

# QA

---

## 一、网商贷贷款金额推荐

### 1. 为什么不用多分类而用二分类的问题来解决?

延伸1. 为什么推荐系统召回阶段常建模为多分类，而排序阶段多建模为二分类?

延伸2. word2vec

1) 模型原理

CBOW (Continuous Bag-of-Word)

Skip-gram

训练优化

negative sampling

hierarchical softmax

### 2. LR、FM、DeepFM、Wide and Deep、ESMM、MMOE的区别、优劣势

2.1 LR (线性回归) 为什么在推荐系统中优于其他机器学习模型 (如GBDT、随机森林、XGboost等)

LR的优点:

缺点:

### 2.2 FM

FM的优点:

缺点:

### 2.3 FFM

FFM的优点:

缺点:

### 2.4 GBDT+LR

GBDT+LR的优点:

缺点:

演进到深度神经网路

DNN的优势

### 2.5 WND

WND的优点:

缺点

## 2.6 DCN

DCN的优点：

缺点：

## 2.7 DeepFM

DeepFM的优点：

## 2.8 DIN

## 2.9 CAN

模型架构

演进到CTCVR联合建模

## 2.10 ESMM

ESMM模型的优点：

## 2.11 ESM2

ESM2的模型结构

损失函数

相关思考

ESMM和ESM2的缺点：

## 2.12 MMOE

MMoE模型结构

思考

MMoE效果归因

## 3. 特征工程怎么做

特征工程：

## 4. 模型实现细节

样本构造

网络结构：

激活函数

损失函数：

优化器：

防止过拟合：

训练过程优化

## 5. 梯度下降

入门计算基本概念

指数加权移动平均数

各种优化器介绍

## 6. 过拟合/欠拟合

欠拟合

过拟合

1) L1正则化 -- 用于特征选择

2) L2正则化

L1、L2的参数 $\lambda$ 如何选择好?

3) early stop

4) Dropout

通过loss曲线诊断神经网络模型

## 7. DNN反向传播公式推导及求解

公式推导

求解

1) 最小二乘法

2) 梯度下降法

3) 牛顿法

★ 牛顿法 VS 梯度下降法

4) 拟牛顿法

梯度下降代码实现

## 8. DNN为什么能够学到特征交叉?

## 9. 激活函数

sigmod、tanh、relu激活函数公式

softmax激活函数

sigmoid激活函数

tanh激活函数

ReLU激活函数

Leaky ReLU

梯度爆炸、梯度消失的原因及解决方法

## 10. AUC/Precision/Recall/F1-score计算

## 11. Focal Loss损失函数

## 12. loss不收敛

## 二、潜在竞对人群识别

图神经网络和知识图谱的区别

信息增益比指标

一致性正则化损失

锐化操作

transformer的多头注意力机制

基于元关系的异质attention

节点相似性度量

基于共同邻居数的指标

基于网络结构信息定义的指标

Adamic/Adar度量

基于全局信息的指标

ROI计算

图聚类系数

半监督学习

图数据增强方法

图的半监督学习和传统半监督学习的差别

图神经网络

GCN

GraphSage

异质图神经网络

HAN、HetGNN、HeGNN

HGT

图谱构成

在大图上进行随机小批次训练的方法——邻居采样方法

树模型

GBDT

思考

XGBoost

XGBoost核心

1. 目标函数加入正则化项

2. 目标函数使用二阶泰勒展开式展开

- 3. 节点分裂准则
- 4. 处理特征缺失和特征稀疏的问题
- 5. 列采样
- 6. 学习率
- 7. XGBoost工程优化——并行列块设计

XGBoost 防止过拟合的方法

XGBoost vs GBDT

随机森林

## 一、网商贷贷款金额推荐

### 1. 为什么不用多分类而用二分类的问题来解决？

本质上多分类和二分类是互通的，多分类问题是可以转成二分类来做的

- 1. 多分类softmax的计算复杂度高\*，转成二分类比较高效
- 2. 多分类要以item作为标签，模型训练的时候学不到item的信息，转成二分类可以把item作为特征和用户特征一起输入到模型中学习，可以学习到用户和item的潜在关系

\*每一次计算，分母都需要将所有类别加一遍，当类别很多时很费时间，而且负样本引起的梯度太大了，导致对正样本的更新力度很小。

### 延伸1. 为什么推荐系统召回阶段常建模为多分类，而排序阶段多建模为二分类？

召回过程是从海量的商品集合里筛选候选子集进入排序，对于召回过程来说，我理解主要的目标还是相关性问题，即召回的结果要和输入query有联系。业界主要是通过挖掘大量负采样\*来提升相关性的效果。而当有大量的负采样一起训练时，其实模型需要做的是将正样本和负采样尽可能的拉开差距，保证正样本反馈的信号相比于负采样是更强的，这其中有个相对程度的对比，而logits本身的大小没有太多物理含义\*\*，所以使用softmax进行多分类可以达到这种效果。

\* 【常用做法是使用负采样的softmax：对batch内进行负采样，比如设定batch\_size为N，其中query\_1-item\_1为1个正样本，再将其他query-items组合成为N-1负样本，从而形成1: N-1样本形式，如此将softmax转换成多个并行的sigmod。二分类器的个数就等于物品总数】

\*\* 【由于没有了原来全部样本求和放在分母上这一部分，所有我们所有神经元的输出和不再是1，不过好在我们只是为了训练embedding而不是真的需要这个softmax输出概率】

在粗精排环节，会对点击及点击后续行为进行精细化学习，尽量将正样本排序至负样本前，即目标在于保证有意义内容排在无意义内容之前。比如候选商品哪些更容易被用户点击或者购买，以及参考候选商品的price来更好的结合业务指标。所以在排序的过程(特别是精排)预估的ctr/cvr/ctovr，这时候的概率是需要关注的，有具体的物理意义的，这个过程使用类似sigmoid的方式输出是比较自然的。

参考：<https://www.zhihu.com/question/491586542>

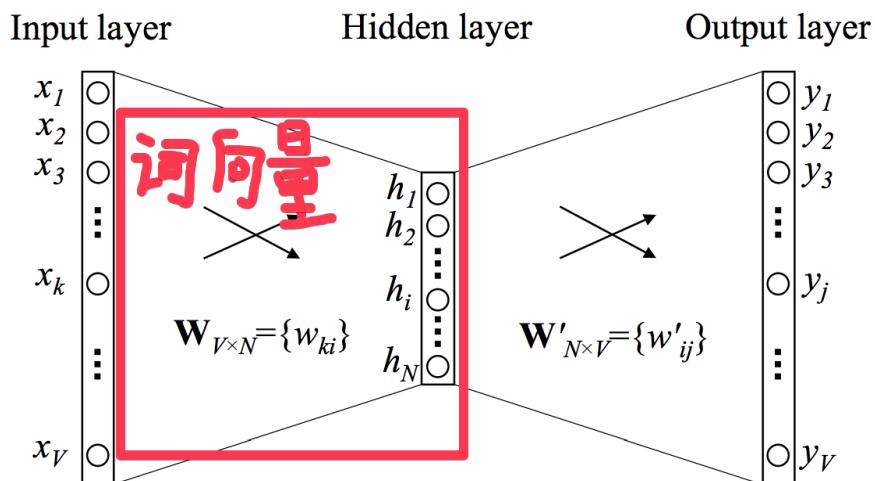
## 延伸2. word2vec

### 1) 模型原理

CBOW (Continuous Bag-of-Word)

由上下文预测目标词

假设vocabulary size 为V, hidden layer 神经元个数为N, 假设只有一个上下文单词，则根据这个上下文单词预测目标词，类似于一个bigram model，如下图所示：



<https://blog.csdn.net/BGoodHabit>

输入是一个one-hot编码的vector (大小为 $V$ )，假设只给定一个上下文word，对于输入编码， $\{x_1, x_2, \dots, x_v\}$ ，只有一个为1，其它都为0。如上图所示，第一层的参数权重 $W_{V \times N}$ ， $W$ 中的每一行是一个 $N$ 维度的向量，代表的就是单词 $w$ 的向量 $v_w$ 表示。从hidden layer到output layer，也有一个不同的权重矩阵 $W' = \{w'_{ij}\}$ ，是一个 $N \times V$ 的矩阵，第 $j$ 列代表了词 $w_j$ 的 $N$ 维度向量，用这个向量和hidden layer输出向量相乘，就得到在 $V$ 中的每个词的分值 $u_j$

$$u_j = {v'_w}^T h \quad (1)$$

然后用 softmax 一个log 线性分类器，得到每个词的分布概率

$$p(w_j | W_I) = y_j = \frac{\exp(u_j)}{\sum_{j'=1}^V \exp(u'_{j'})} \quad (2)$$

其中 $w_I$ 是上下文单词， $w_j$ 是目标词， $y_j$ 是output layer层的第 $j$ 个 神经元 的输出。

$v_w$ 和 $v'_w$ 是词 $w$ 的两个词向量表示， $v_w$ 来自input->hidden的权重矩阵 $W$ 的行，而 $v'_w$ 来自hidden->output的权重矩阵 $W'$ 的列，通过上面的语言模型，第一层权重 $W$ 就是我们学习的词向量矩阵。

首先让我们来看下从hidden->output层的权重 $W'$ 训练过程更新公式，这里就不做详细推导了，若想详细推导可以参考论文：[word2vec Parameter Learning Explained](#)，

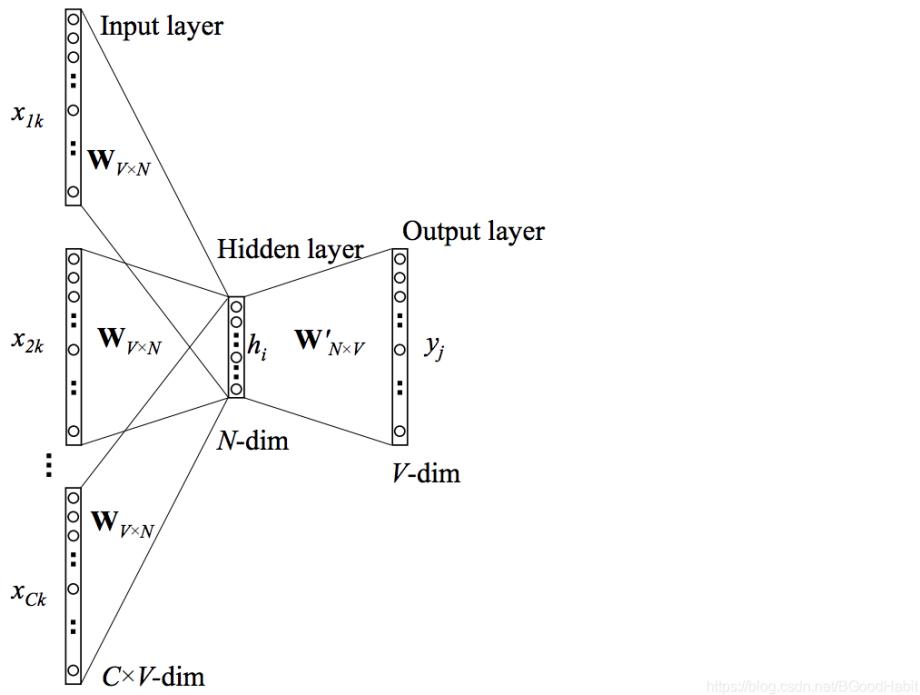
$$\frac{\partial E}{\partial u_j} = y_j - t_j := e_j \quad (3)$$

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial w'_{ij}} = e_j \cdot h_i \quad (4)$$

$$v'_{w_j}^{(new)} = v'_{w_j}^{(old)} - \eta \cdot e_j \cdot h \quad (5)$$

其中E为 loss function， $e_j$ 就是说的残差， $t_j$ 就是真实label (不是1就是0) 从公式 (2) 和 (5) 可知，更新最后一层的权重梯度，我们必须计算词典 $V$ 中的每个词，当 $V$ 很大的时候，最后一层softmax计算量会非常的耗时

上面介绍了只有一个上下文词的情况，当多个上下文词的情况的时候，只需要将多个上下文词的向量求平均作为输入就行，如下图所示：

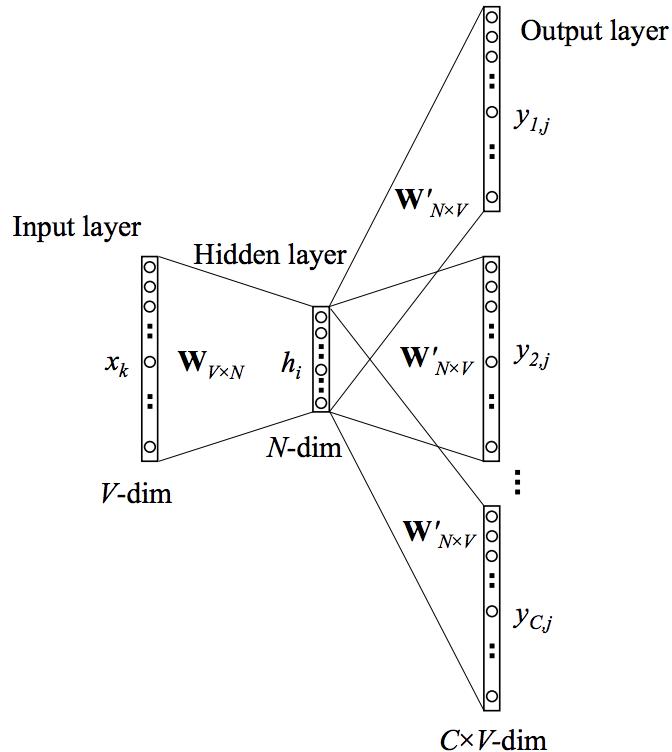


<https://blog.csdn.net/BGoodHabit>

$$\text{其中, } h = \frac{1}{C} W^T (x_1 + x_2 + \dots + x_C)$$

### Skip-gram

skip-gram model训练词向量方式和cbow输入和输出目标反过来， cbow用上下文词预测中心词，而skip-gram 用中心词预测上下文词， 如下图所示：



<https://blog.csdn.net/BGoodHabit>

理解了cbow模型，skip-gram也就不难理解了。有一点需要说明的是，因为一个中心词对应的上下文词可能有多个，所以正样本会有多个pair对<input\_word, output\_context1>, <input\_word, output\_context2>等，所以相同的语料，skip-gram比cbow训练的要更久

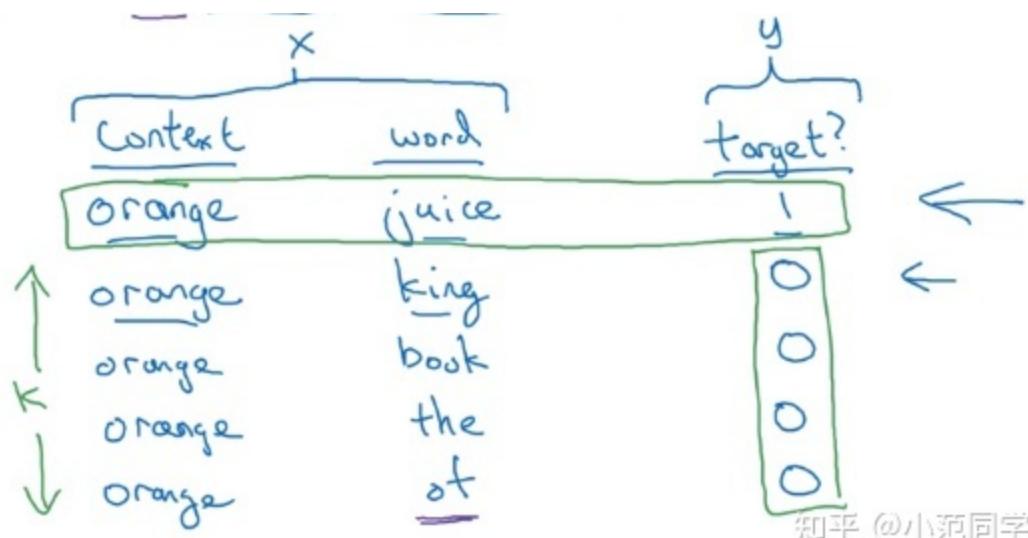
## 训练优化

从上面的公式(2)可以看出，softmax分母那项归一化，每次需要计算所有的V的输出值，才可以得到当前j节点的输出，当V很大的时候， $O(V)$ 的计算代价会非常高。所以在训练word2vec模型的时候，用到了两个tricks，一个是**negative sampling**，每次只采少量的负label，不需要计算全部的V；另外一个是**hierarchical softmax**，通过构建赫夫曼tree来做层级softmax，从复杂度 $O(V)$ 降低到 $O(\log_2 V)$ 。

### negative sampling

负采样的基本思想是，把原问题转换成一个新问题：预测两个词之间是否是上下文-目标词对，如果是词对，则学习的目标为1；否则为0。这样原来“预测目标词是什么”这种多分类任务被转换为“预测候选词是不是目标词”这种二分类任务（这种理解方式有些特别）

上下文（样本特征）和目标词（目标类别）都作为特征输入到模型中，标签为0/1



在整个采样过程中，输入中的“上下文”是一致的。关于负样本数k的选取，建议：

- 小数据集， $k=5\sim 20$ ；
- 大数据集， $k=2\sim 5$ 。

转化成二分类任务后，模型变成了逻辑回归： $P(y = 1|c, t) = \sigma(\Theta_t^T e_c)$

### 如何选择负样本：

- 通过单词出现的频率进行采样：导致一些类似a、the、of等词的频率较高；
- 均匀随机地抽取负样本：没有很好的代表性（无法表征数据集分布）；

- 一种折中做法：每个词（类别）的采样概率为

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{10000} f(w_j)^{3/4}}$$

这种方法处于上面两种极端采样方法之间，即不用频率分布，也不用均匀分布，而采用的是对词频的 $3/4$ 除以词频 $3/4$ 整体的和进行采样的。其中， $f(w_j)$ 是语料库中观察到的某个词的词频。

注意：高词频的词会被更高机率采样到，在推荐领域也有类似的结论。但是这里的 $3/4$ 并不总是适用，跟数据集有关，有时候甚至可以是负值，也就是更多采样冷门样本。参考：[AI Box专栏：Efficient and Effective: 百篇论文概览负采样方法的前世今生](#)

参考：Andrew Ng网课：[双语字幕]吴恩达深度学习deeplearning.ai\_哔哩哔哩\_bilibili

### hierarchical softmax

根据词频构建哈夫曼树，所有的叶子节点构成了词 $V$ ，中间节点则共有 $V-1$ 个，不学习 $V$ 个词的向量，而是对中间节点进行向量学习，而每个叶子上的节点可以通过路径中经过的中间节点去表示。

#### 4.2 叶子节点词的概率表示

上图假设我们需要计算 $w_2$ 的输出概率，我们定义从根节点开始，每次经过中间节点，做一个二分类任务（左边或者右边），所以我们定义中间节点的 $n$ 左边概率为

$$p(n, left) = \sigma(v_n'^T \cdot h)$$

其中 $v_n'$ 是中间节点的向量，那么右边概率

$$p(n, right) = 1 - \sigma(v_n'^T \cdot h) = \sigma(-v_n'^T \cdot h)$$

从根节点到 $w_2$ ，我们可以计算概率值为：

$$\begin{aligned} p(w_2 = w_O) &= \\ p(n(w_2, 1), left) \cdot p(n(w_2, 2), left) \cdot p(n(w_3, 3), right) &= \sigma(v_{n(w_2, 1)}'^T \cdot h) \cdot \sigma(v_{n(w_2, 2)}'^T \cdot h) \cdot \sigma(-v_{n(w_3, 3)}'^T \cdot h) \end{aligned}$$

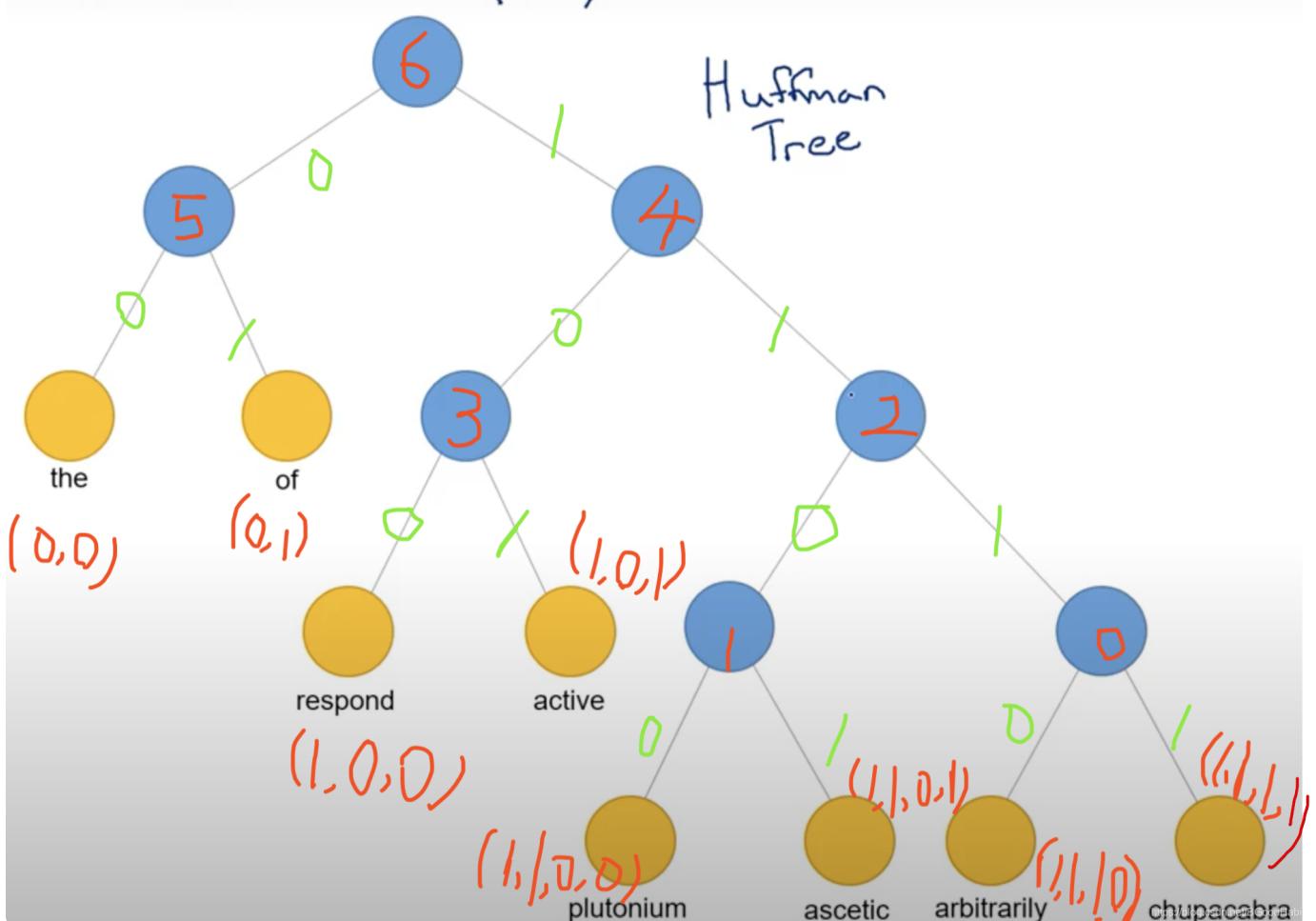
其中  $\sigma$  为 sigmoid 函数

#### 4.3 各叶子节点概率值相加为1

从论文里讲解的公式可以推导得出

$$\sum_{i=1}^V p(w_i = w_O) = 1$$

### 具体case



假设我们的词典有word [the, of ,respond, active, plutonium, ascetic, arbitrarily, chupacabra] 共8个单词。接下对hierarchical softmax训练做一个讲解，主要步骤如下：

### 1) 预处理：构建huffman树

根据语料中的每个word的词频构建赫夫曼tree，词频越高，则离树根越近，路径越短。如上图所示，词典V中的每个word都在叶子节点上，每个word需要计算两个信息：路径（经过的每个中间节点）以及赫夫曼编码，例如：“respond”的路径经过的节点是 (6, 4, 3)，编码label是 (1, 0, 0)

构建完赫夫曼tree后，每个叶子节点都有唯一的路径和编码，hierarchical softmax与softmax不同的是，在hierarchical softmax中，不对V中的word词进行向量学习，而是对中间节点进行向量学习，而每个叶子上的节点可以通过路径中经过的中间节点去表示。

### 2) 模型的输入

输入部分，在cbow或者skip-gram模型，要么是上下文word对应的id词向量平均，要么是中心词对应的id向量，作为hidden层的输出向量

### 3) 样本label

不同softmax的是，每个词word对应的是一个V大小的one-hot label，hierarchical softmax中每个叶子节点word，对应的label是赫夫曼编码，一般长度不超过  $\log_2 V$  。在训练的时候，每个叶子节点的label

统一编码到一个固定的长度，不足的可以进行pad

#### 4) 训练过程

我们用一个例子来描述，假如一个训练样本如下：

input	output
chupacabra [0,0,0,...,1,0,0,...]	active [1, 0, 1]

假如我们用skip-gram模型，则第一部分，根据“chupacabra”词的one-hot编码乘以 $W$ 权重矩阵，得到“chupachabra”的词向量表示，也就是hidden的输出，根据目标词“active”从赫夫曼tree中得到它的路径path，即经过的节点(6, 4, 3)，而这些中间节点的向量是模型参数需要学习的，共有 $V - 1$ 个向量，通过对应的节点id，取出相应的向量，假设是 $w'$  ( $3 \times N$ ) (这里设词向量维度为N)，分别与hidden输出向量相乘，再经过sigmoid函数，得到一个 $3 \times 1$ 的score分值，与实际label [1,0,1]，通过如下公式：

$$path\_score = tf.log(tf.multiply(label, score) + tf.multiply(1 - label, 1 - score))$$

$$loss = tf.reduce\_sum(tf.negative(path\_score))$$

$tf.multiply(label, score)$ 点乘，是获取该叶子节点路径中走右边各分支对应的概率分值， $tf.multiply(1 - label, 1 - score)$ 获取路径中走左边各分支对应的概率分值，相加就是该路径[1,0,1]对应的[右分支—左分支—右分支]分别对应的概率分值 $[score_1, score_2, score_3]$ ，由于小于1的分值相乘，会容易溢出，则取log，乘法变加法，loss则是与我们期望的概率值越大相反，需要取负用 $tf.negative$ 实现，计算出loss，则就可以用优化器来计算模型中各参数的梯度进行更新了

在tensorflow中的实现如下：

```
▼ Python | 复制代码
1 # label [None, max_path_length], score [None, max_path_length]
2 path_score = tf.log(tf.multiply(label, score) + tf.multiply(1-label, 1-score))
3 loss = tf.reduce_sum(tf.negative(path_score))
```

参考：

<https://zhuanlan.zhihu.com/p/456081303>

<https://blog.csdn.net/BGoodHabit/article/details/106163130>

## 2. LR、FM、DeepFM、Wide and Deep、ESMM、MMOE的区别、优劣势

参考：<https://zhuanlan.zhihu.com/p/432817787>

## 2.1 LR（线性回归） 为什么在推荐系统中优于其他机器学习模型（如GBDT、随机森林、XGboost等）

在推荐领域，效果很好的大部分都是类别特征（也就是01特征），浮点型特征很少。

所以你说GBDT、随机森林、XGboost这些模型的效果会很好吗？很难说，因为这些模型的长处往往都在浮点型特征，也就是连续型特征。这些树模型会设计规则对这些连续特征进行分段，如果大部分特征都是01特征，那还怎么分段呢？

LR可以简单理解成若干个特征的加权和。W表示权重，所有  $W_i X_i$  最后累加在一起，得到一个预测的概率。这意味着模型其实是“记住”了每个特征和最终结果的关系，我们把模型拟人化，把它看成一个机器人的话。机器人看到样本有特征A并且点击了，于是特征A的权重提升一点，样本有特征B但是没点击，于是把特征B的权重降低一些。模型就是在这样一个策略当中找到一个最佳的平衡。也就是说，一些容易被记忆的特征往往发挥比较好的效果。因此，LR模型在推荐领域发挥作用，本质上就是靠的“记性”。因为它可以记住那些类别特征以及类别特征的组合，所以它往往比那些看起来更高端的树模型效果要好。

LR 模型训练表达式：

$$Y = W^T X = w_0 + \sum_{i=1}^n w_i x_i$$

只有一次项有时候效果不好，尤其是在特别稀疏的场景当中，刻画能力不够。我们做特征的时候经常会把两项特征组合起来做成新的组合特征，由于我们这样操作引入了新的特征，找到了新的特征组合，所以能够挖掘出之前无法得到的信息，因此模型也会有更好的效果。

当我们把特征进行二项组合之后，会得到这样的式子：

$$\hat{y} = w_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^n w_{ij} x_i x_j$$

这里的 $x_i$ 和 $x_j$ 分别表示两个不同的特征取值，对于n维的特征来说，这样的组合应该一共有 $C_n^2$ 种，这个值是 $\frac{n(n-1)}{2}$ ，也就意味着我们需要同样数量的权重参数。但是有一个小问题，我们前面已经说过，由于特征可能非常稀疏，导致n非常大，比如上百万，这里两两特征组合的特征量级大约是n的平方，那么因此带来的参数数量就是一个天文数字。想想看对于一个上百亿甚至更多参数空间的模型来说，我们需要多少训练样本才可以保证完全收敛？这是不可想象的。

此时，计算复杂度为  $O(n^2)$

多元线性回归模型的损失函数为：

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 = \frac{1}{2} (X\theta - Y)^T (X\theta - Y)$$

### LR的优点：

1. 计算高效（相比SVM、GBDT等），特征较少且稠密的情况下，参数空间比较小，LR模型可以迅速收敛
2. 可解释性强，可以查阅得到所有特征的权重，解释究竟是什么特征发挥了作用

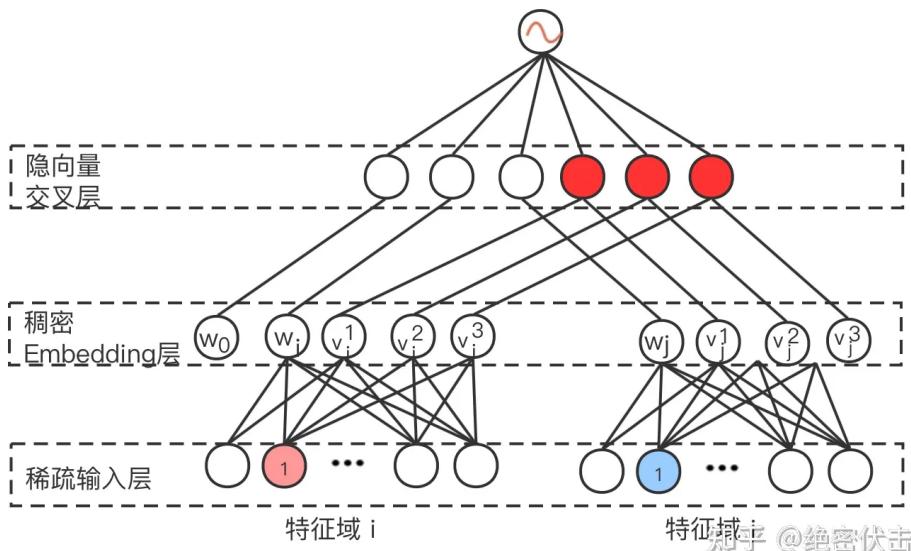
### 缺点：

1. 大量人工特征工程，需要依靠经验来做特征工程；或者暴力地将两两特征进行特征工程，但是两两特征之间都需要学习一个参数，当特征维度很大时学习的参数会特别多，计算量很大；
2. 特征稀疏的场景表现不好：只能学习在我们的训练过程中出现过的特征组合，比如说训练中有贷款和利息的组合，我们可以得到他们之间的关系系数，但是没有贷款和购物的组合，在预测的时候就会认为他们之间的关系是0，也就是没有关系。显然不合理

## 2.2 FM

**旨在解决稀疏数据下的特征组合问题。**FM模型可以看成底层为特征维度为 $n$ 的离散特征输入，经过Embedding层后，对Embedding层线性部分（LR）和非线性部分（特征交叉部分）累加后输出。

**FM为每个特征学习了一个隐向量，在特征交叉时，使用两个特征隐向量的内积作为特征交叉的权重。**



FM的数学形式如下所示：

$$\hat{y}(\mathbf{x}) = w_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^n \sum_{j=i+1}^n \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j \quad (1)$$

在实际的二分类问题中，只需要经过一个sigmoid变换后，便可以得到最终输出正例的概率。和二阶多项式模型（POLY2）相比，其主要的区别是用两个向量的内积  $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$  取代了单一的权重系数  $w_{ij}$ ，模型复杂度从  $O(n^2)$  降低到  $O(kn)$ （ $k$  为隐向量维度， $n \gg k$ ）。具体地说，FM为每个特征学习了一个隐向量，在特征交叉时，使用两个特征隐向量的内积作为特征交叉的权重。

公式（1）是FM通用的拟合公式，可以采用不同的损失函数用于解决回归、二分类、多分类等问题，比如采用MSE（Mean Square Error）损失函数来求解回归问题，也可以采用交叉熵损失函数来求解分类问题。在进行二分类时，FM的输出需要经过sigmoid变换，这与逻辑回归是一样的。直观上看，FM的复杂度是  $O(kn^2)$ ，但是通过下面公式（2）的化简，FM的复杂度可以降低到  $O(kn)$ 。由此可见，FM可以在线性时间对新样本做出预测。

$$\sum_{j=i+1}^n \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j = \frac{1}{2} \sum_{f=1}^k \left( \left( \sum_{i=1}^n v_{i,f} x_i \right)^2 - \sum_{i=1}^n v_{i,f}^2 x_i^2 \right) \quad (2)$$

我们再来看一下FM的训练复杂度，利用SGD随机梯度下降训练模型，模型各个参数的梯度如下所示。

$$\frac{\partial \hat{y}}{\partial \theta} = \begin{cases} 1 & \text{if } \theta \text{ is } w_0 \\ x_i & \text{if } \theta \text{ is } w_i \\ x_i \sum_{j=1}^n v_{j,f} x_j - v_{i,j} x_i^2 & \text{if } \theta \text{ is } v_{i,j} \end{cases} \quad (3)$$

其中， $v_{j,f}$  是隐向量  $\mathbf{v}_j$  的第  $f$  个元素。由于  $\sum_{j=1}^n v_{j,f} x_j$  只与  $f$  有关，而与  $i$  无关，在每次迭代过程中，只需要计算一次所有  $f$  的  $\sum_{j=1}^n v_{j,f} x_j$ ，就能够方便的得到所有  $v_{j,f}$  的梯度。由于计算所有  $f$  的  $\sum_{j=1}^n v_{j,f} x_j$  的复杂度是  $O(kn)$ ，因此计算所有  $v_{j,f}$  梯度的复杂度是  $O(kn)$ 。计算所有  $w_i$  的复杂度是  $O(n)$ ，因此FM训练参数的复杂度就是  $O(kn)$ 。总结起来，FM可以在线性时间完成训练和预测，是一种非常高效的模型。

上面介绍了FM训练和预测的高效性，而本质上，FM引入隐向量的做法，与矩阵分解用隐向量代表用户和物品的做法异曲同工。但是，FM将矩阵分解隐向量的思想进行了进一步扩展，从单一的用户、物品隐向量扩展到了所有特征上。

隐向量的引入使FM能够更好地解决数据稀疏性问题。举例来说，在某广告推荐场景，样本有两个特征，分别是城市（Country）和广告类别（Ad\_type），某训练样本的特征组合是（USA, Movie）。在POLY2模型中，只有当USA和Movie同时出现在一个训练样本中，模型才能学到这个特征组合对应的权重；而在FM模型中，USA的隐向量也可以通过（USA, Game）样本进行更新，Movie的隐向量也可以通过（China, Movie）样本进行更新，这大大降低了模型对数据稀疏性的要求。甚至对于一个从未出现过的特征组合（China, Game），由于模型之前分别学习过China和Game的隐向量，具备了计算该特征组合的能力，这是POLY2无法实现的。相比POLY2，FM虽然丢失某些特征组合的精确记忆能力。但是泛化能力大大提高。

相比之后深度学习模型复杂的网络结构导致难以部署和线上服务，FM较容易实现的模型结构使其线上推断的过程相对简单，也更容易线上部署和服务。因此，FM模型至今仍然在主流推荐场景拥有一席之地。

### Tips

- 一般只考虑二维的特征交叉。三重特征的交叉往往没有意义，并且会过于稀疏。
- 在实际的应用场景当中，我们并不需要设置非常大的K，因为特征矩阵往往非常稀疏，我们可能没有足够多的样本来训练这么大量的参数，并且限制K也可以一定程度上提升FM模型的泛化能力。

### FM的优点：

- 减小了参数空间，提升了学习效率

通过将参数矩阵进行因子分解，参数个数由  $\frac{n(n-1)}{2} + n + 1$  减少成  $kn + n + 1$ ，模型训练复杂度也由  $O(mn^2)$  变为  $O(mnk)$ 。m为训练样本数。对于训练样本和特征数而言，都是线性复杂度。

## 2. 对稀疏数据有更好的学习能力，提升了模型预估能力

**单特征参数 (k维隐向量 添加 TeX 公式 ) 的学习要比交叉项参数 添加 TeX 公式 学习得更充分。**

在稀疏场景中，很多特征组合可能在训练过程中都不会出现。比如在刚才的例子当中用户A和电影B的组合，可能用户A在电影B上就没有过任何行为，那么这个数据就是空的，我们也不可能训练出任何参数来。而在FM中，对于 $\langle A, B \rangle$ 的特征组合向量，只需要在训练过程中出现过有其中一个特征的任意特征组合，如 $\langle A, C \rangle$ 、 $\langle B, C \rangle$ 、 $\langle A, F \rangle$ 等，都可以学习到A和B对应的隐向量  $v_A, v_B$ ，从而将  $v_A, v_B$  点乘就能得到 $\langle A, B \rangle$ 交叉特征的系数，不需要在训练过程中出现 $\langle A, B \rangle$ 组合。

也就是说，单特征隐向量  $v_i$  相比交叉项参数  $w_{ij}$  可以从更多样本中得到学习训练，并且在预测中可以得到未出现过的特征组合向量。

**缺点：**

1. 每个特征只引入了一个隐向量，不同类型特征之间交叉没有区分性。FFM模型正是以这一点作为切入进行改进。

**参考：**

- <https://www.cnblogs.com/techflow/p/13967844.html>
- 详细因子分解公式推导：[https://blog.csdn.net/anshuai\\_aw1/article/details/83747171](https://blog.csdn.net/anshuai_aw1/article/details/83747171)

## 2.3 FFM

相比FM模型，FFM通过引入特征域感知（field-aware）这一概念，把相同性质的特征归于同一field，使模型的表达能力更强。

$$\hat{y}_{FFM}(x) = w_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^n \sum_{j=i+1}^n \langle \mathbf{v}_{i,f_j}, \mathbf{v}_{j,f_i} \rangle x_i x_j$$

**FFM的隐向量为  $v_{\langle \text{当前特征}id, \text{ 进行特征交叉的对象域}id \rangle}$ ，因此一个特征会对应一组隐向量，假设其他特征对应的域个数为J，则一个特征有(J-1)个隐向量；**

**二次项的系数是通过与特征field相关的隐向量点积得到的，二次项共有 添加 TeX 公式 个。**

公式(4)是FFM的数学形式。和FM的区别在于隐向量由原来的 $v_{i,f}$ 变成了 $v_{i,f_j}$ ，其中 $f_j$ 表示特征 $x_j$ 对应的特征域，这意味着每个特征对应的不是唯一一个隐向量，而是一组隐向量。FM可以看成是FFM的特例，是把所有特征都归属到一个field中。这里解释下上面提到的有关域(field)的概念，以图4所示的训练样本为例。

Clicked	Publisher (P)	Advertiser (A)	Gender (G)
Yes	ESPN	Nike	Male 知乎 @绝密伏击

图4 FFM训练样本示例

其中，Publisher、Advertiser、Gender是三个特征域，ESPN、Nike、Male分别是3个特征域的特征值。由此可见，特征域是指具有相似属性的特征集合。

如果按照FM原理，上面的隐向量特征组合一共只有3项，分别是 $\langle v_{ESPN}, v_{Nike} \rangle$ 、 $\langle v_{ESPN}, v_{Male} \rangle$ 、 $\langle v_{Nike}, v_{Male} \rangle$ 。而在FFM中，隐向量特征组合一共有6项，分别是 $\langle v_{ESPN}, A, v_{Nike}, P \rangle$ 、 $\langle v_{ESPN}, G, v_{Male}, P \rangle$ 、 $\langle v_{Nike}, P, v_{ESPN}, G \rangle$ 、 $\langle v_{Nike}, G, v_{Male}, A \rangle$ 、 $\langle v_{Male}, P, v_{ESPN}, G \rangle$ 、 $\langle v_{Male}, A, v_{Nike}, G \rangle$ 。

在FFM模型的训练过程中，需要学习 $n$ 个特征在 $f$ 个特征域上的 $k$ 维隐向量，参数数量是 $n \cdot k \cdot f$ 个。在训练和预测方面，FFM的二次项并不能像FM那样简化，因此复杂度为 $kn^2$ 。

相比FM模型，FFM引入了特征域的概念，为模型引入了更多信息，提升了模型的表达能力。但是与此同时，FFM的计算复杂度上升到 $kn^2$ ，远大于FM的 $kn$ 。在实际应用中，需要在模型效果和计算开销之间进行权衡。

### FFM的优点：

- 在高维稀疏性数据集中表现很好。
- 相对FM模型精度更高，特征刻画更精细。

### 缺点：

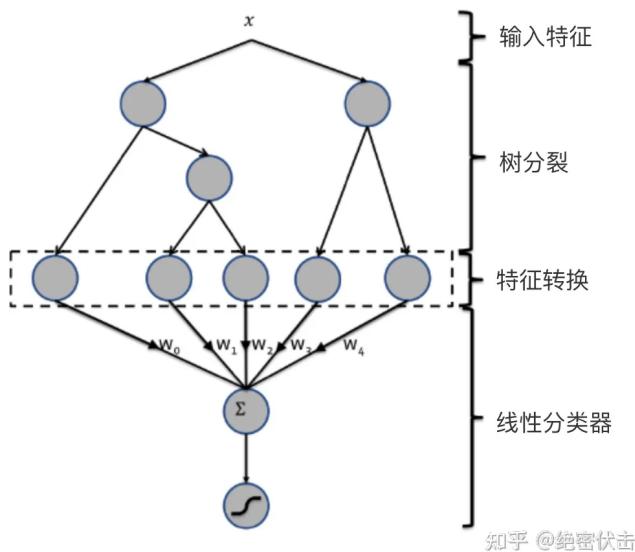
- 时间开销大。FFM时间复杂度为 $O(kn^2)$ ，FM时间复杂度为 $O(kn)$ 。
- 参数多容易过拟合，必须设置正则化方法，以及早停的训练策略。

更高维的特征组合对于FM和FFM来说会出现组合爆炸的问题，无论是参数数量还是训练复杂度都过高，难以在实际应用中实现。那么如何突破FM和FFM二阶特征交叉的限制，进一步加强模型的组合能力，就

成了推荐模型发展的方向。下面介绍的**GBDT+LR**的组合模型在一定程度上解决了高阶特征交叉的问题，在工程上也具有可操作性。

## 2.4 GBDT+LR

Facebook通过GBDT自动构建组合特征，再将组合特征输入到LR中，可有效的处理高阶特征组合和筛选的问题。



GBDT (Gradient Boost Decision Tree) 是一种常用的非线性模型，由多颗回归树组成，每次迭代都在减少残差的梯度方向新建一颗决策树，迭代多少次就会生成多少颗决策树。GBDT的思想使其具有天然优势可以发现多种有区分性的特征以及特征组合，决策树的路径可以直接作为LR输入特征使用，省去了人工特征组合的步骤。

那么GBDT是如何生成组合特征向量的？图6是生成特征向量的过程。

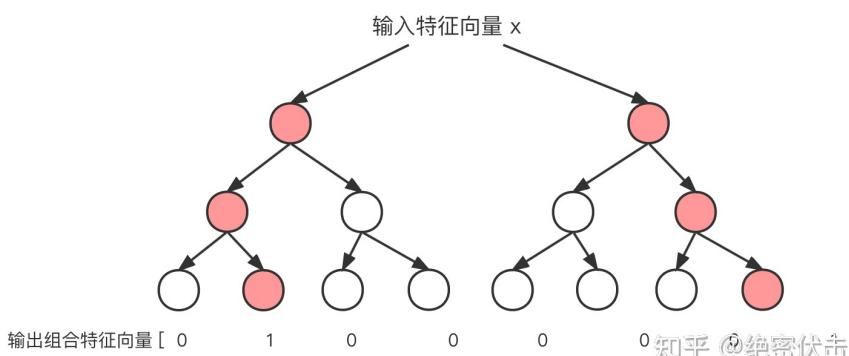


图6是GBDT生成组合特征的例子，GBDT由两棵子树组成，每棵子树有4个叶子节点，输入一个训练样本 $x$ 后，落在子树1的第2个叶子节点上，生成特征向量[0,1,0,0]，随后落在子树2的第4个叶子节点上，生成特征向量[0,0,0,1]，最后连接所有特征向量，形成最终的组合特征向量[0,1,0,0,0,0,1]。

### (1) 为什么建树采用多棵树?

一棵树的表达能力较弱，不足以表达多个有区分性的特征组合，多棵树的表达能力更强。GBDT每棵树都在学习前面树学习的不足。

### (2) 为什么建树采用GBDT而非RF?

RF也是多棵树，但从效果上实践证明不如GBDT。GBDT前面的树，特征分裂主要体现对多数样本有区分度的特征；后面的树，主要体现的是经过前 $N$ 棵树后，残差仍较大的少数样本。优先选择在整体上有区分度的特征，再选用少数针对少数样本有区分度的特征，更加符合特征选择思想。

#### GBDT+LR的优点：

1. 可有效的处理高阶特征组合和筛选的问题
2. 模型简单，易实现

#### 缺点：

2. GBDT无法实时训练，更新时间长

## 演进到深度神经网路

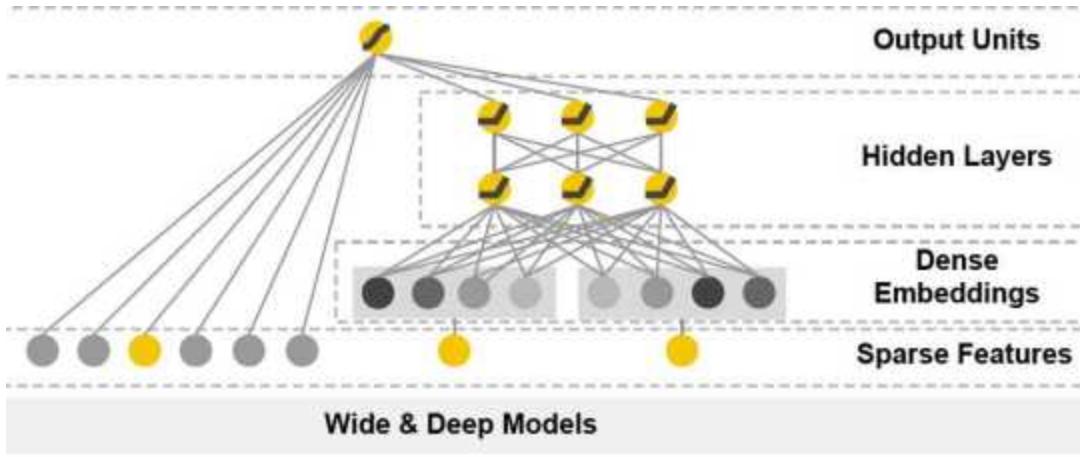
#### DNN的优势

1. Dnn网络通过深层网络的训练拟合，可以学习到特征之间高阶且凝练抽象的信息，比如说可以学习到基金和贷款都是属于金融类的信息。
2. dnn会根据训练目标对网络训练误差进行反馈修正，特别是对于涉及数据量比较大的场景来说，对网络参数的拟合效果更好，深度神经网络的表现能力也就会更好。
3. 对于分布比较广泛、波动性比较高的数据，dnn的表现也比较好。

## 2.5 WND

WDL模型，Wide部分模型结构是LR，主要用于处理大量的人工交叉特征，作用是让模型具有较强的记忆能力；Deep部分模型结构是神经网络，善于挖掘挖掘潜在的隐藏模式，作用是让模型具有泛化能力，这样的特点使得WDL能够兼容复杂的人工交叉特征，同时学习到更复杂的高阶交叉。

Wide部分体现特征间的乘性组合，Deep部分体现特征域间的加性组合。



**Figure 1: The spectrum of Wide & Deep models.**

Wide部分是通用的线性模型，其输出为  $y = \mathbf{w}^T x + b$ ，同时Wide部分对输入特征进行交叉，方式如下所示。

$$\phi_k(\mathbf{x}) = \prod_{i=1}^d x_i^{c_{ki}} \quad c_{ki} \in \{0, 1\} \quad (1)$$

其中如果第*i*个特征参与了交叉，则  $c_{ki}$  为1，否则为0。

Deep部分是一个多层神经网络，对于高维稀疏特征，首先转化为稠密的Embedding，Embedding的大小一般为10-100，然后concat到一起输入到MLP中，每一层的计算方式如下所示。

$$a^{(l+1)} = f \left( W^{(l)} a^{(l)} + b^{(l)} \right) \quad (2)$$

最终，WDL将Wide输出和Deep输出组合在一起，表示如下。

$$P(Y=1|\mathbf{x}) = \sigma \left( \mathbf{w}_{wide}^T [\mathbf{x}, \phi(\mathbf{x})] + \mathbf{w}_{deep}^T a^{(l_f)} + b \right) \quad (3)$$

### Tips:

Deep部分输入的是全量的特征向量，包括数值型特征和类别型特征。需要强调的是**类别特征做Embedding时，不同特征是分开做Embedding，Embedding size可以不一样**。设想如果所有类别特征学习一个Embedding，如果Embedding size太小，便不足以刻画更多信息，而Embedding size太大，会导致模型参数急剧膨胀。

对于WDL的训练，Wide部分使用FTRL优化器，Deep部分使用AdaGrad优化器。

## 2. 为什么Wide部分和Deep部分使用不同的优化器

前面有提到Wide部分和Deep部分使用不同的优化器。Wide部分使用FTR优化器是为了产生稀疏解，那么为什么Deep部分不用考虑稀疏性？

Wide部分采用了两类id特征的乘积，当两类稀疏特征进行组合时，会让原本非常稀疏的multi-hot特征向变得更加稀疏。正因为如此，Wide部分的权重数量是海量的。为了能够达到上线的要求，采用FTRL过滤掉那些稀疏特征无疑是非常好的工程经验。

而对于Deep部分，输入的要么是年龄、安装APP数量这些数值特征，要么是已经降维并稠密化的Embedding向量，这样Deep部分就不存在严重的稀疏特征问题，自然可以选择更适合深度学习训练的AdaGrad优化器。

### WND的优点：

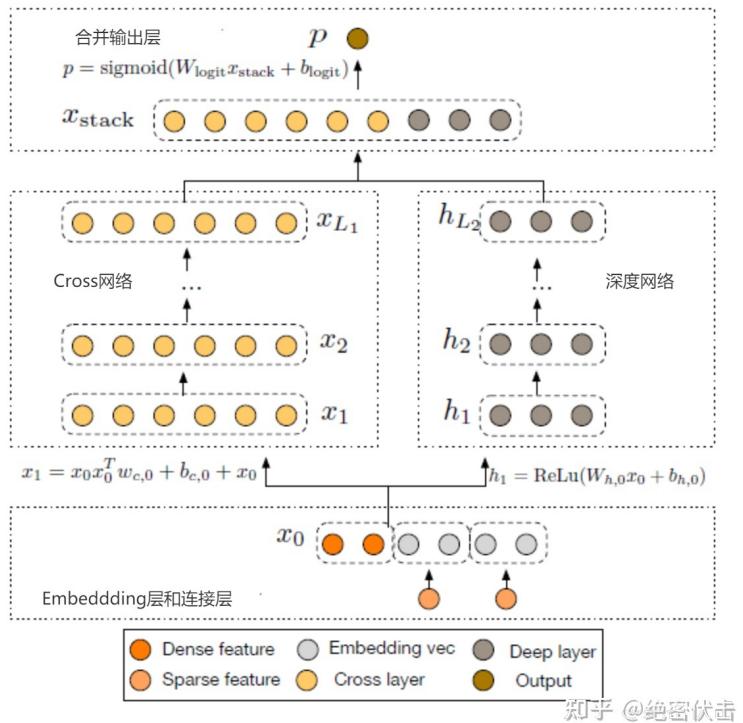
- 有效地将LR的优点和DNN的优点结合起来，可以同时学习到低阶（wide）和高阶（deep）的特征信息，相比DNN提升率模型对重要特征的记忆能力，相比传统机器学习提升了模型的泛化能力。
- WDL模型开启了深度学习在高维稀疏场景的落地，对于推荐和计算广告领域来说，它的特征极大的大规模和稀疏。而对于数亿级的稀疏特征，和其它所有特征使用全局的Embedding，Embedding维度都选取上千左右，那么参数空间可以一下达到万亿级，这对于计算能力的要求非常高。而WDL模型对于这类高维稀疏特征，采用**分组Embedding**的思路，不需要每个id特征的Embedding大小都是上千，高维稀疏的可以缩小到几十的Embedding大小，这样整体的参数空间将大幅缩小。

### 缺点

Wide侧的人工特征交叉的工程仍无法避免。

## 2.6 DCN

针对WDL的Wide部分进行改进，提出了Deep&Cross模型（简称DCN）。其主要思路是**使用Cross网络替代原来的Wide网络，以此提升模型的交叉能力。**



知乎 @绝密伏击

和WDL模型相比，DCN使用Cross网络替代了原来简单的LR网络，增加了特征之间的交互力度，使用多层交叉层对输入特征进行交叉。第  $l+1$  层和第  $l$  层之间的关系如下所示。

$$x_{l+1} = x_0 x_l^T w_l + b_l + x_l = f(x_l, w_l, b_l) + x_l \quad (4)$$

cross网络是一个残差网络

其中  $x_l$  和  $x_{l+1}$  分别是第  $l$  层和第  $l+1$  层的输出向量。可以看到，公式 (4) 中的特征交叉部分  $f$  对特征向量做了外积操作。交叉层的操作可以表示为图6的形式。

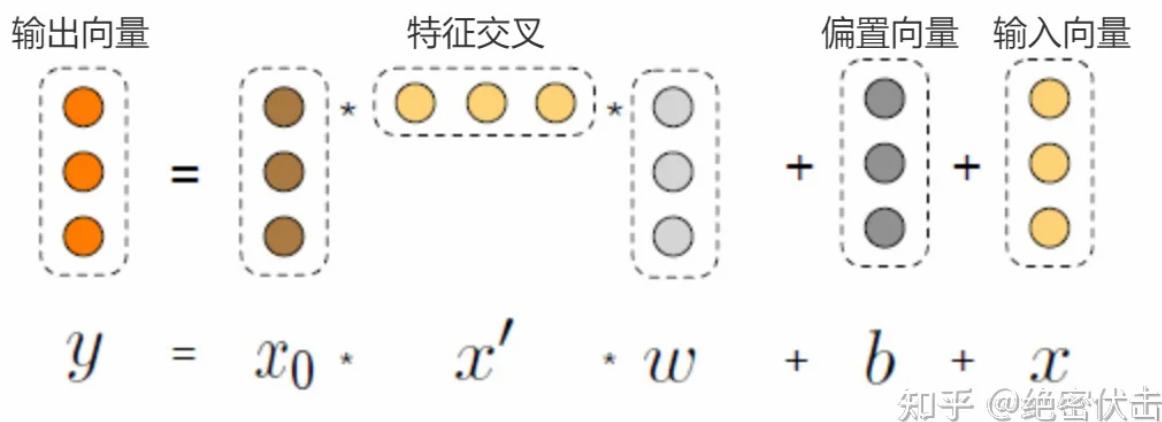


图6 DCN交叉层操作

公式 (2) 对应的图形化表示如Fig 2, 图中所有的向量维度都是一样的。利用  $X_0$  与  $X'$  做向量外积得到所有的元素交叉组合, 层层叠加之后便可得到任意有界阶组合特征, 当cross layer叠加  $l$  层时, 交叉最高阶可以达到  $l + 1$  阶, 参考[5]可以举例说明:

为了方便起见, 首先将  $b$  设置为零向量, 令  $X_0 = \begin{bmatrix} x_{0,1} \\ x_{0,2} \end{bmatrix}$ , 那么

$$\begin{aligned}
X_1 &= X_0 X'_0 W_0 + X_0 \\
&= \begin{bmatrix} x_{0,1} \\ x_{0,2} \end{bmatrix} [x_{0,1} x_{0,2}] \begin{bmatrix} w_{0,1} \\ w_{0,2} \end{bmatrix} + \begin{bmatrix} x_{0,1} \\ x_{0,2} \end{bmatrix} \\
&= \begin{bmatrix} x_{0,1}^2, x_{0,1} x_{0,2} \\ x_{0,2} x_{0,1}, x_{0,2}^2 \end{bmatrix} \begin{bmatrix} w_{0,1} \\ w_{0,2} \end{bmatrix} + \begin{bmatrix} x_{0,1} \\ x_{0,2} \end{bmatrix} \\
&= \begin{bmatrix} w_{0,1} x_{0,1}^2 + w_{0,2} x_{0,1} x_{0,2} \\ w_{0,1} x_{0,2} x_{0,1} + w_{0,2} x_{0,2}^2 \end{bmatrix} + \begin{bmatrix} x_{0,1} \\ x_{0,2} \end{bmatrix} \\
&= \begin{bmatrix} w_{0,1} \textcolor{red}{x}_{0,1}^2 + w_{0,2} \textcolor{red}{x}_{0,1} x_{0,2} + x_{0,1} \\ w_{0,1} \textcolor{red}{x}_{0,2} x_{0,1} + w_{0,2} \textcolor{red}{x}_{0,2}^2 + x_{0,2} \end{bmatrix}
\end{aligned} \tag{3}$$

继续计算  $X_2$ , 有:

$$\begin{aligned}
X_2 &= X_0 X'_1 W_1 + X_1 \\
&= \begin{bmatrix} x_{0,1} \\ x_{0,2} \end{bmatrix} \left[ \begin{bmatrix} w_{0,1} x_{0,1}^2 + w_{0,2} x_{0,1} x_{0,2} + x_{0,1} \\ w_{0,1} x_{0,2} x_{0,1} + w_{0,2} x_{0,2}^2 + x_{0,2} \end{bmatrix} \right] \begin{bmatrix} w_{1,1} \\ w_{1,2} \end{bmatrix} \\
&\quad + \begin{bmatrix} w_{0,1} x_{0,1}^2 + w_{0,2} x_{0,1} x_{0,2} + x_{0,1} \\ w_{0,1} x_{0,2} x_{0,1} + w_{0,2} x_{0,2}^2 + x_{0,2} \end{bmatrix} \\
&= \begin{bmatrix} w_{0,1} x_{0,1}^3 + w_{0,2} x_{0,1}^2 x_{0,2} + x_{0,1}^2 \\ w_{0,1} x_{0,1}^2 x_{0,2} + w_{0,2} x_{0,1} x_{0,2}^2 + x_{0,1} x_{0,2} \end{bmatrix} \begin{bmatrix} w_{1,1} \\ w_{1,2} \end{bmatrix} \\
&\quad + \begin{bmatrix} w_{0,1} x_{0,1}^2 + w_{0,2} x_{0,1} x_{0,2} + x_{0,1} \\ w_{0,1} x_{0,2} x_{0,1} + w_{0,2} x_{0,2}^2 + x_{0,2} \end{bmatrix} \\
&= \begin{bmatrix} w_{0,1} w_{1,1} \textcolor{red}{x}_{0,1}^3 + w_{0,2} w_{1,1} \textcolor{red}{x}_{0,1}^2 \textcolor{red}{x}_{0,2} + w_{1,1} \textcolor{red}{x}_{0,1}^2 + w_{0,1} w_{1,2} \textcolor{red}{x}_{0,2} \textcolor{red}{x}_{0,1}^2 + w_{0,2} w_{1,2} \textcolor{red}{x}_{0,2}^2 \textcolor{red}{x}_{0,1} \\ w_{0,1} w_{1,1} \textcolor{red}{x}_{0,1}^2 \textcolor{red}{x}_{0,2} + w_{0,2} w_{1,1} \textcolor{red}{x}_{0,1} x_{0,2}^2 + w_{1,1} \textcolor{red}{x}_{0,1} x_{0,2} + w_{0,1} w_{1,2} \textcolor{red}{x}_{0,2}^2 \textcolor{red}{x}_{0,1} + w_{0,2} u \end{bmatrix} \\
&\quad + \begin{bmatrix} w_{0,1} \textcolor{red}{x}_{0,1}^2 + w_{0,2} \textcolor{red}{x}_{0,1} x_{0,2} + x_{0,1} \\ w_{0,1} \textcolor{red}{x}_{0,2} x_{0,1} + w_{0,2} \textcolor{red}{x}_{0,2}^2 + x_{0,2} \end{bmatrix}
\end{aligned}$$

从公式 (3) (4) 的标红处可以看出, 当cross layer叠加  $l$  层时, 交叉最高阶可以达到  $l + 1$  阶, 并且包含了所有的交叉组合, 这是DCN的精妙之处。

### 复杂性分析：

一般来说，要对特征进行高阶显式交叉，一定会加大模型的参数量。在DCN中，假设Cross Layer有 $L_c$ 层，最底层输入 $X_0$ 为 $d$ 维。因为每一层仅有 $W, b$ 参数，所以模型参数量会额外增加 $d \times L_c \times 2$ 个。

因为 $X_0 X_l' W_l = X_0 (X_l' W_l)$ ，先计算 $X_l' W_l$ 得到一个标量，然后再与 $X_0$ 相乘，在时间与空间上计算效率都得到提升，最终 $L_c$ 层Cross Layer的时空复杂度均为 $O(dL_c)$ ，也就是说时空复杂度随着输入与层数线性增长，这是非常好的性质。

让我们更进一步分析一下Cross Layer的设计理念，通常来说当两个向量的外积之后，往往想到的是再利用一个矩阵来对结果进行压缩变换，假设向量外积之后为 $d \times n$ 维矩阵，那么为了将结果变换为 $d$ 维向量，需要使用的参数矩阵为 $n \times d$ 维，不仅参数量变多了，而且矩阵相乘的运算复杂度高达三次方。

参数量过多很容易造成过拟合，参数量的适当精简反而可以提高模型的泛化能力与鲁棒性。如Cross Layer中使用向量而不是矩阵来对结果进行变换，本质上是通过参数共享的模式减少参数量。共享参数能够对样本外数据仍有较好的泛化能力，且能够对噪声数据带来的参数变化进行纠正。

### DCN的优点：

1. 多层交叉层组成的DCN网络在WDL模型中wide部分的基础上进行特征的自动交叉组合，避免了更多依赖人工特征交叉。
2. DCN相比DeepFM，可以任意阶组合特征。DeepFM的FM部分仅限于二阶特征交叉。

### 缺点：

1. cross网络的参数个数为 $d \times L \times 2$ ，DeepFM中FM部分并没有增加额外的参数。同时计算复杂度fm部分为线性的 $O(d \times k)$ ，cross部分计算复杂度为 $O(d^3 \times L)$ 。因此，DCN训练难度应该会更大。

## 2.7 DeepFM

DeepFM将wide&deep的LR部分替换成FM，可以自动表征特征间的2阶组合。

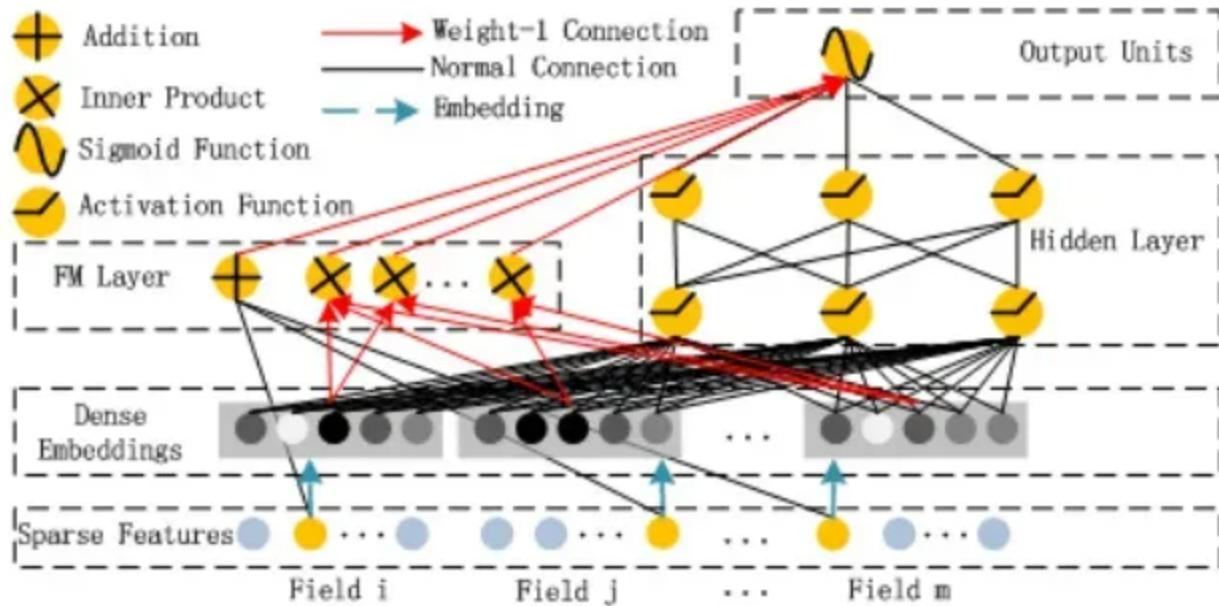


Figure 1: Wide & deep architecture of DeepFM. The wide and deep component share the same input raw feature vector, which enables DeepFM to learn low- and high-order feature interactions simultaneously from the input raw features.

知乎 @小小小小AI



Wide&Deep 和 DeepFM的主要区别有两点：

1. 左边选用的广度模型不同，wide&deep是线性模型和dnn的融合，LR部分需要人工特征工程来完成特征组合；deepfm是fm和dnn的融合，FM部分自动表征特征域的2阶组合。
2. 两部分的输入是分开的，deepfm的两部分是共享同一个输入和embedding的。（DeepFM中每一个feature field经过embedding层转化为一个隐向量，多个特征field concat成一个密集向量分别作为FM部分和Deep部分的输入。FM部分将每个field的隐向量两两组合，最后在输出层和Deep部分输出concat成最终的输出层。）

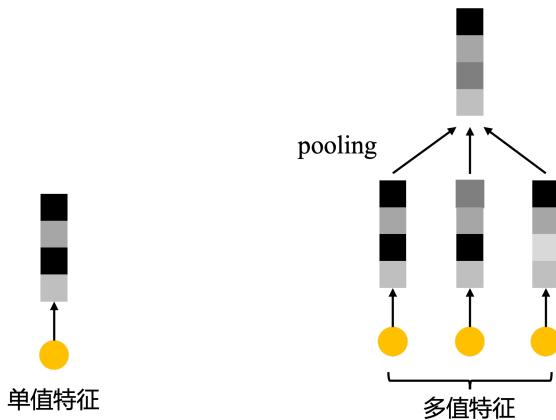
**DeepFM的优点：**

1. 端到端的模型，自动表征特征域的2阶组合，比Wide&Deep更加高效。
2. fm和dnn两部分共享底层的embedding输入，神经网络的误差反馈可以同时根据这两部分信息对参数进行更新，使Embedding层的信息表达更加准确，最终可以提升推荐效果。

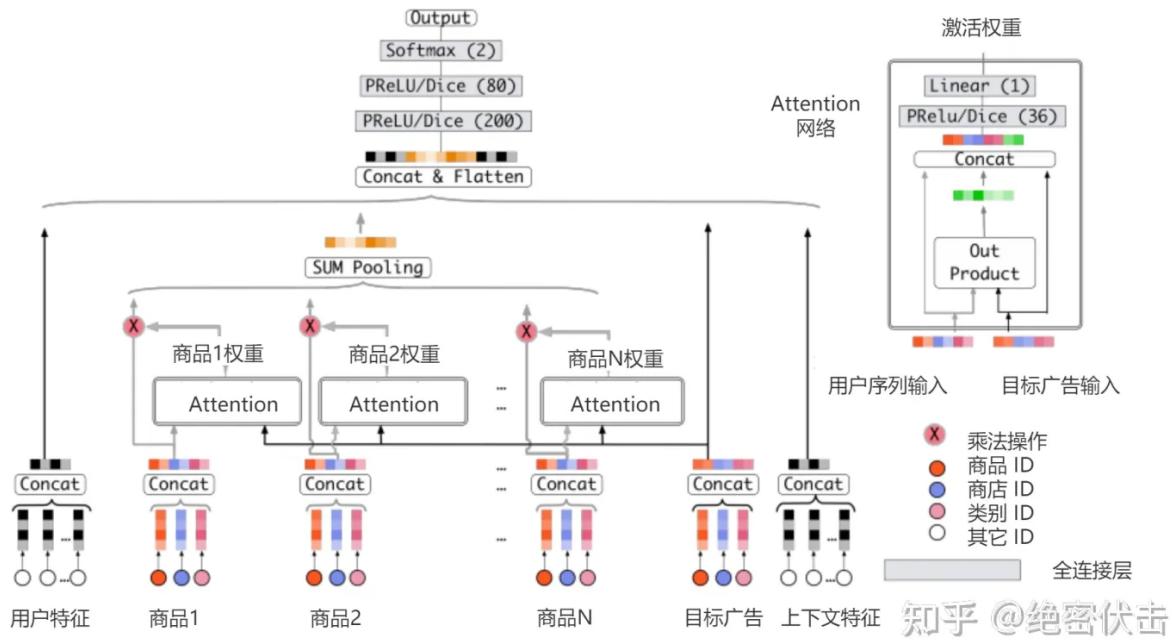
	非线性	深度神经网络	特征工程	低阶特征	高阶特征	共享输入和embedding
DeepFM	✓	✓	✗	✗	✓	✓
Wide & deep	✓	✓	✗	✗	✓	✗
DNN	✓	✓	✗	✗	✓	-
FM	✓	✗	✓	✓	✗	-
LR	✗	✗	✓	✓	✗	-

## 2.8 DIN

在DIN之前，对于用户兴趣的表达就是直接把所有历史点击做sum/average pooling，如下图所示。这样导致所有历史行为都没有区分，实际上用户当前的兴趣应该只和历史上某些行为是关联的。（每个item的权重应该由目标广告（商品）与该item之间的相关性决定。）



在实际业务中通过人工特征组合的方法，用ad的shop属性去匹配用户历史行为的shop list，如果命中了就说明历史有过直接行为，用行为id和频次来表示特征组合。举个简单例子，candidate的shop属性是Adidas，用户在以下shop里买过东西，Adidas、Nike、Adidas、Sony，则组合特征就是Adidas\_2（2为Adidas出现的频次）。【这种将目标与历史点击行为相匹配的过程，称为hard attention。如果在普通的DNN中使用这种交叉特征，效果应该不比DIN差多少。】Adidas\_2这个特征描述了用户对candidate的兴趣程度，但是Adidas和Nike应该是相似的shop，但是和Sony关联不大，而hard attention的方法无法刻画这种潜在较弱的相关关系。



为了解决上面hard attention的问题，阿里巴巴将其扩展到soft attention，就是DIN里面的动态兴趣表达，如下式所示。

$$V_u = \sum_{i=1}^N w_i * V_i = \sum_{i=1}^N g(V_i, V_a) * V_i \quad (1)$$

其中  $V_u$  是用户Embedding向量， $V_a$  是目标广告的Embedding向量， $V_i$  是用户第*i*次的行为Embedding， $g(V_i, V_a)$  是Attention函数，可以看到目标广告Embedding会和每次用户行为Embedding计算权重，最后的加权和就是用户的兴趣表达。DIN模型结构如图4所示，其中Attention的实现是一个小的神经网络，输入是用户历史行为某个Embedding向量以及目标广告Embedding向量，经过元素减、乘操作后，与原Embedding向量一同连接后形成全连接层的输入，最后通过单神经元输出层生成注意力得分。

#### attention网络最后的输出层没有加sigmod激活函数

在具体实现时，DIN并没有保证注意力得分相加为1，这样做的好处是可以量化用户的历史点击对目标广告的贡献程度。比如用户历史浏览商品中90%是clothes，10%是electronics，对于给定的目标商品T-shirt和phone，T-shirt激活了大部分属于clothes的历史行为，因此可能获得比phone更大的兴趣强度，而如果做归一化处理，可能最后T-shirt和phone对于clothes的Attention值没太大差别。

上面红字有误！！！ attention网络一般使用softmax激活函数。

虽然softmax和attention的目的非常吻合，但也存在几个问题。

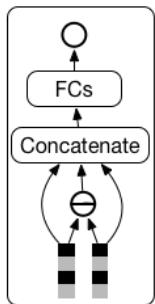
- (1) softmax具有强化区分度的性质，这对于用户兴趣广泛的场景，会带来损伤，它会加强用户头部兴趣，打压非头部兴趣，从而导致用户原有的广泛兴趣在提取过程中被削窄，影响最终推荐效果。

- (2) softmax强制归一化，导致和target不相似的item也会分配到一定权重贡献于兴趣表征，从而对最终的兴趣表征造成损伤。

针对softmax在用户兴趣提取中的问题，采用softplus激活函数进行改进。softplus的优势包括：

- (1) 不存在强制归一化操作，因此对和target不相关的item，不会强制分配权重从而干扰最终兴趣表征；
  - (2) 经过softplus的输出接近于输入，不会进一步强化区分度，从而保留了用户的头部和非头部兴趣，有利于推荐结果的多样性，更适用于用户兴趣广泛的场景。

## 核心：attention单元



## Activation Unit

- $\oplus$  element-wise +
- $\ominus$  element-wise -
- $\otimes$  element-wise  $\times$

上图比较清晰的展示了activation unit模块是如何工作的，以用户历史行为的商品id序列与候选广告id举例子。

假设用户历史行为：

- 商品id序列为 $ug = [ug_1, ug_2, \dots, ug_n]$ , 经过embedding层后对应的向量为 $uge = [uge_1, uge_2, \dots, uge_n]$ 。
  - 商铺id序列为 $us = [us_1, us_2, \dots, us_n]$ , 经过embedding层后对应的向量为 $use = [use_1, use_2, \dots, use_n]$ 。

### 候选广告：

1. 商品id为ag， 经过embedding层后对应的向量为age。
  2. 商品id为as， 经过embedding层后对应的向量为ase。

则activation unit的做法为：

1. 把用户历史行为的商品向量与候选广告商品向量做 $\ominus$ （向量对应元素相减），即 $\text{aug} = [\text{age} \ominus \text{uge}_1, \text{age} \ominus \text{uge}_2, \dots, \text{age} \ominus \text{uge}_n]$
  2. 把用户历史行为的商铺向量与候选广告商铺向量做 $\ominus$ ，即 $\text{aus} = [\text{ase} \ominus \text{use}_1, \text{ase} \ominus \text{use}_2, \dots, \text{ase} \ominus \text{use}_n]$
  3. 把1, 2步中得到的结果向量 $\text{aug}$ ,  $\text{aus}$ 与用户商品向量 $\text{uge}$ , 用户商铺向量 $\text{use}$ , 候选广告商品向量 $\text{ag}$ , 候选广告商铺向量 $\text{as}$ , 做拼接concatenate, 输入一个全连接网络。

参考：含实现源码 <https://blog.csdn.net/u012328159/article/details/123043033>

## 2.9 CAN

CAN通过建模特征之间的协同关系，使其具备笛卡尔积特征交叉的效果，同时具备一定的信息共享。实验发现，在CTR预估问题里，把待预估的商品信息（如item id）和用户历史行为序列做笛卡尔积，形成一个新的序列，对其Embedding后pooling效果很好，在DIN和DIEN的基础上再有比较明显的提升。

**Co-action**指的是特征之间的协同效应，对于目标的预测有显著的影响。比如：在电商场景下，用户最近一次历史点击item和目标item就是一对很强的Co-action特征，对于预测用户是否会点击目标item非常有帮助。

**笛卡尔积特征ID交叉和之前的特征交叉的区别：**

1. 笛卡尔积特征ID交叉：是原始未编码的特征ID进行交叉生成一个新的特征后再进行独立的one-hot编码和embedding，然后在输入到网络中。这算是新增的特征，后续在网络训练中与原始特征无关
2. 之前的特征交叉：原始特征one-hot编码并embedding后输入到网络中在进行交叉特征，训练时交叉特征的Embedding会随本身的学习以及原始特征embedding更新而更新。

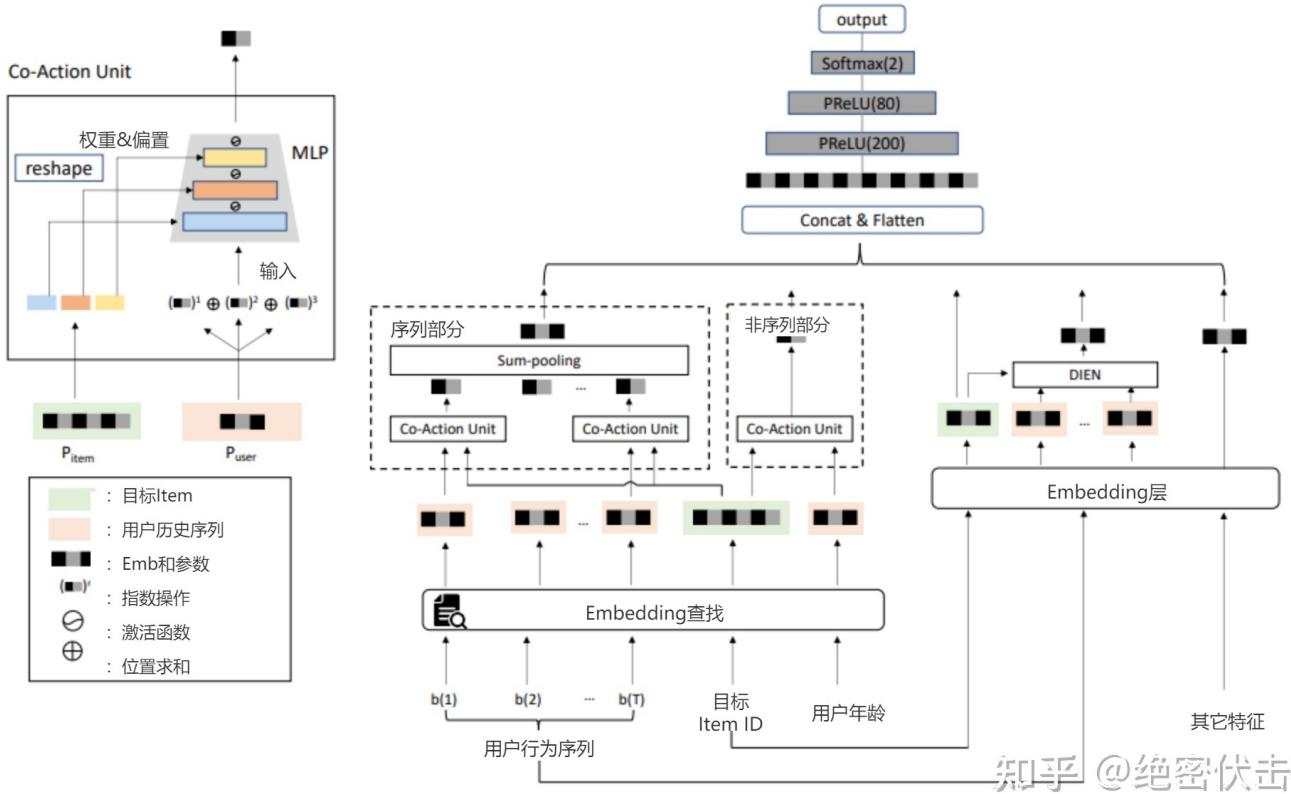
但在具体实现中存在问题：

1. 笛卡尔积特征交叉的效果虽然好，但是会产生大量稀疏的特征，学习复杂度高，这也是传统逻辑回归用于手工特征交叉建模的主要问题所在。
2. 传统的自动特征交叉并不是直接且独立地从数据中学习来的，而是通过于是特征A和B的嵌入来间接表达和学习的，虽然解决了稀疏性问题，带来了一定的泛化性，但是在一些Co-Action非常强烈的场景中，这样的泛化可能是过度的。原本信息较强的特征交叉可能会被特征A和B的嵌入更新而减弱。**因此，模型层面的交叉，如果不涉及到独立交叉特征嵌入的学习，本质上都是伪交叉，可能学不好高阶的交叉信息。**

## 模型架构

CAN网络和正常的深度CTR网络几乎一样，所不同的是多了个CAN Unit结构。**CAN Unit本质上是一种特征提取器，输入是要建模的co-action特征对，输出是建模后的co-action向量。**这样，常规的embedding向量和co-action向量拼接起来经过多层MLP，就能输出点击率预估值。

整个CAN模型的架构如下图所示，左侧是建模co-action的部分，右侧是DIEN模型结构。



知乎 @绝密伏击

在左侧网络中，部分潜在强co-action效应的特征输入Co-action Unit：从 $X_{user}, X_{item}, x_{user}, x_{item}$ 中选出部分特征作为Co-action Unit的输入来建模强Co-action效应。这部分用户和物品的特征会独立的映射到另一种向量  $p_{user}, p_{item}$ 上。 $p_{user}$ 和右侧的 $e_{user}$ 是独立的两种参数。物品侧同理。即经过图中左侧Paramter lookup后输出的向量。从画出的图可以看出，作者想建模的co-action特征包括两大类：sequence类，如：目标item id和用户历史行为item id；non-sequence类，如：用户年龄age和目标item id。对于序列特征，每个历史行为item和目标item建模得到的co-action向量经过sum pooling得到最终的序列向量。

### Co-action Unit

一方面拓展了特征 $p_{item}$ 的向量维度数，并切分成多个slot向量。不同slot向量可以通过reshape的方式来充当MLP网络不同层参数的角色，另一方面则通过对 $p_{user}$ 手动构造高阶的特征来作为MLP的输入。二者通过多层的MLP来实现co-action效应的建模。

$M$ 表示item unique ID的数量， $N$ 是user unique ID的数量(个人认为应该用 $N$ 表示，原文用 $M$ 表示)。 $T$ 是item端拓宽的向量维度数， $D$ 是user端的向量维度数， $D < T$ 。当然，user和item可以角色对调。之所以这么做，是因为广告系统中，候选的item数量很少，只占全部item的一小部分，比用户点击历史中的item少的多。(我推测，由于候选item被更多地共享了，所以要拓宽其参数空间，让其学习过程更独立)。

具体而言，MLP第1层每个神经元的连接数和某个user的向量 $p_{user} \in \mathbb{R}^D$ 是一样的，都是 $D$ 维；而 $p_{item} \in \mathbb{R}^T$ 充当着多层MLP参数的容器，即：所有MLP层总共的维度为 $T$ 。为了建模co-action，分为几个步骤：

- $p_{item} \in \mathbb{R}^T$ 通过reshape的方式转成MLP层的权重矩阵和偏置向量参数。即：

$$P_{item} = \text{concatenate}(\text{flatten}(\mathbf{w}^{(i)}), b^{(i)}_{i=0, \dots, K-1})$$

$\mathbf{w}^{(i)}$ 是MLP第*i*层的权重矩阵，而 $b^{(i)}$ 是第*i*层的偏置向量。把 $\mathbf{w}^{(i)}$ 矩阵拍扁成向量，并和 $b^{(i)}$ 向量拼在一起，这样就形成了第*i*层网络的参数的向量化形式，所有*K*层网络的参数拼接在一起，则形成了所有*K*层网络参数的向量化形式，共*T*维度。这个过程反向执行，其实就是reshape操作，就能将*T*维向量转成MLP网络的各层参数。比如三层网络，每层参数数量一样，即：三层网络的权重矩阵都为 $D \times D$ 维，偏置都为*D*维，则

$$T = 3 \times (D \times D + D)$$

- $p_{user} \in \mathbb{R}^D$ 作为MLP的输入，进行多层MLP前向传播，不同层之间有激活函数。即：

$$\begin{aligned} h^{(0)} &= p_{user} \\ h^{(i)} &= \sigma(\mathbf{w}^{(i-1)} \otimes h^{(i-1)} + b^{(i-1)}), i = 1, 2, \dots, K-1 \\ H(p_{user}, p_{item}) &= h^{(K)} \end{aligned}$$

⊗表示矩阵乘法。 $H$ 代表最终输出的feature co-action向量。如果 $p_{user}$ 是用户点击序列中的item，每个历史点击item都会和目标item建模co-action输出1个向量，那么最终的向量表示是这个序列所有item的co-action向量的sum-pooling结果。

## 演进到CTCVR联合建模

### 1. CTR/CVR/CTCVR的定义

$pCTR = p(\text{点击} | \text{曝光})$

$pCVR = p(\text{转化} | \text{曝光, 点击})$

$pCTCVR = pCTR * pCVR$ , 即  $p(\text{点击, 转化} | \text{曝光}) = p(\text{点击} | \text{曝光}) * p(\text{转化} | \text{曝光, 点击})$

--> CTCVR和CVR的区别是, CTCVR是在全量曝光样本空间中学习到的, 从曝光到购买的最终概率, 即点击且转化概率; CVR是在点击样本空间中学到的转化率。

如何通过提高CTCVR来提高GMV, 是电商推荐系统的核心目标。

### 2. 存在的问题:

传统的CVR预估存在着两个主要的问题: 样本选择偏差和稀疏数据。

1. 样本选择偏差是指传统的CVR预估只用了点击后的样本来训练CVR模型, 但训练好的模型是在整个样本空间中去做推断的。而因为训练样本只是样本空间中的一个很小子集, 所以训练样本的特征相对于整个样本空间的特征是有偏的。样本选择偏差会降低模型的泛化性能。
2. 稀疏数据是指一方面有点击行为的用户也仅仅只占所有用户的一小部分, 可训练数据量小, 导致模型很难达到拟合的状态。

## 2.10 ESMM

ESMM模型借鉴多任务学习的思路, 在完整的样本数据空间同时学习点击率(CTR)和转化率(CVR)。将CVR预估问题, 转成CTR预估和CTCVR预估问题。

Esmm模型由两个子网络组成, 左边的子网络用来拟合 $pCVR$ , 右边的子网络用来拟合 $pCTR$ , 同时, 两个子网络的输出相乘之后可以得到 $pCTCVR$ 。模型损失函数对应的等于ctr部分的损失加上ctcvr部分的损失。

- CTR任务在学什么

基于点击样本训练CVR任务中, 训练时的样本空间和推理时的存在分布不一致问题, 是因为训练时的样本经过了筛选, 筛选的根据是用户是否点击, 相当于在构造训练样本时存在bias, 而在ESMM模型中, 点击行为作为CTR任务来学习, 相当于在学构造样本时的bias, 从而使CVR任务学得更好。

### ESSM模型的优点:

1. 在整个样本空间中进行建模, 可以在一定程度上消除了样本选择偏差的问题;

2. 底层共享embedding向量，使得CVR子任务也能够从曝光未点击的样本中学习，缓解训练数据稀疏的问题。

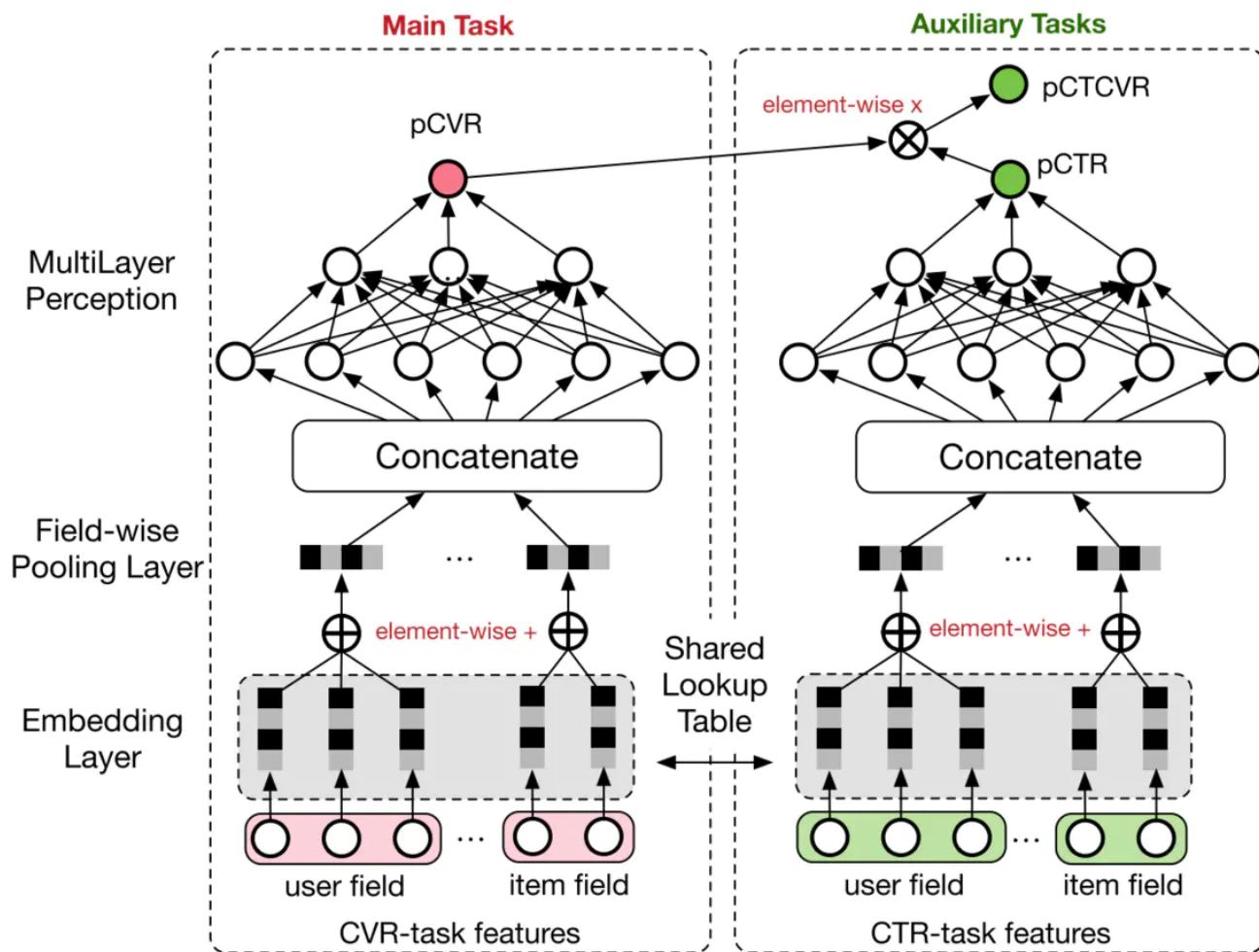
缺点：（仍然由数据稀疏导致的以下两点问题）

1. 正样本稀疏

ESMM模型虽然缓解了数据稀疏问题，但由于用户在曝光-点击-购买这个行为链路上，点击和购买行为相比曝光非常稀疏，因此带来正样本稀疏问题，是否需要对正负样本不平衡问题做一些处理？

2. 训练由CTR任务主导

点击行为远少于曝光，购买行为远少于点击，导致损失函数由CTR任务主导，是否需要对损失函数进行权重的调整？



参考：<https://www.jianshu.com/p/d0bbc679191f>

## 2.11 ESM2

用户在电商场景中从曝光到购买，存在一条行为路径，ESMM模型将该行为路径抽象为曝光-点击-购买，实际上用户在点击到购买这个过程之间，往往存在其它行为，比如收藏、加入心愿单或购物车等，

如图4(a)所示，而加入购物车或心愿单这两种行为占比较大，因此ESM2将此行为路径抽象为如图4(b)所示，曝光-点击-加入购物车/加入心愿单/其它行为-购买。当用户产生了加入购物车或心愿单行为，其后续的购买概率远高于其它行为的后续购买概率，因此ESM2进一步将此链路抽象为曝光-点击-有特定行为/无特定行为-购买，如图4(c)所示，其中特定行为包括决定性行为和其它行为，图4(c)中虚线表示无特定行为。

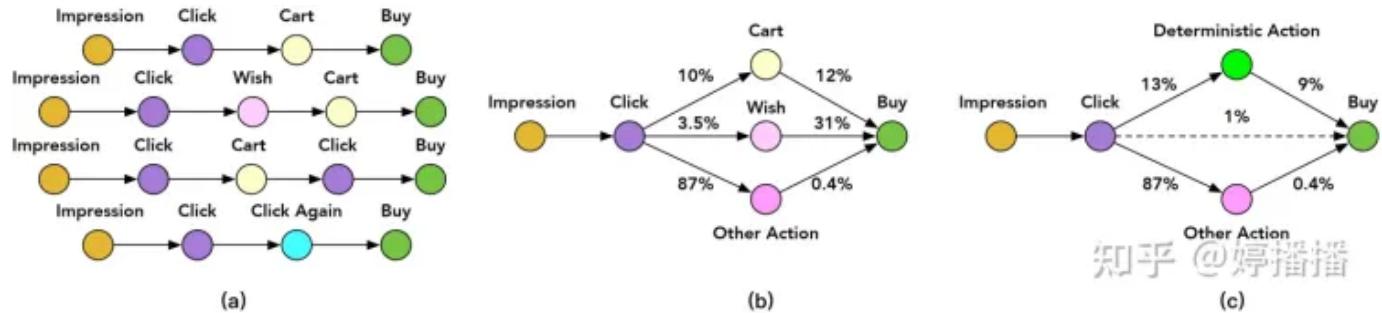


图4 电商场景用户从曝光到购买的行为路径

类似ESMM中的CTCVR概率，ESM2中从曝光到购买的最终概率，是由该行为链路上的对应概率相乘得到。

- CTAVR：表示从曝光到特定行为时点击且有特定行为的转化率
- CVR：表示从点击到购买时点击且购买的概率
- CTCVR：表示从曝光到购买时点击且购买的概率

**CTR, CTAVR, CTCVR均是基于曝光样本空间，因此在建模过程中，将通过对这三个任务进行直接建模，实现对CVR的基于曝光样本的间接学习。**

## ESM2的模型结构

ESM2的模型结构设计思路同ESMM：(1) 底层embedding共享；(2) 不同任务之间，根据用户从曝光到购买的行为链路，对基于曝光样本且有依赖关系的任务进行级联，对基于曝光样本的任务直接学习，从而解决样本选择偏差问题。

ESM2的模型结构如图5所示，包括三部分：embedding共享模块(Shared Embedding Module, 简写SEM)，预测分解模块(Decomposed Prediction Module, 简写DPM)和序列行为合成模块(Sequential Composition Module, 简写SCM)。

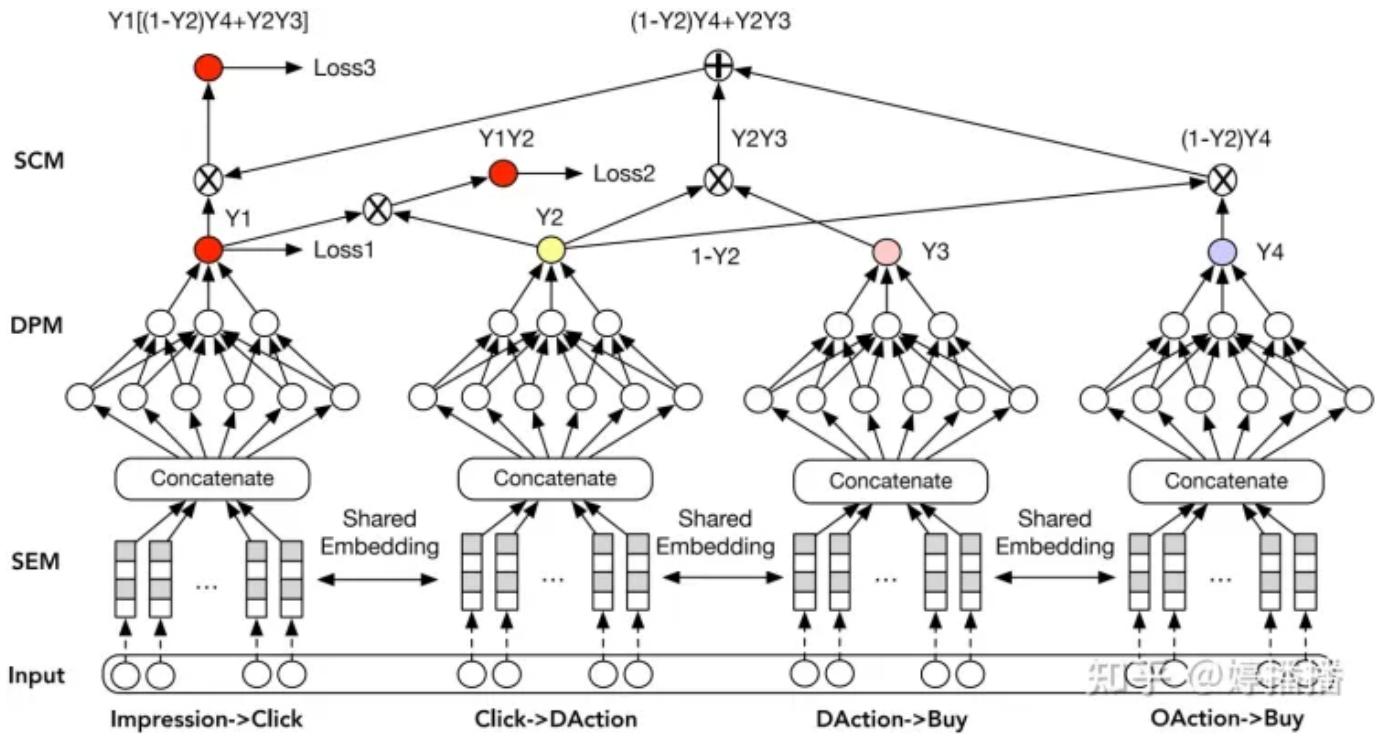


图5 ESM2模型结构

1. SEM实现了不同任务之间底层embedding共享，使模型可以充分利用样本量大的任务进行学习，缓解数据稀疏问题。
2. DPM根据用户行为链路，将最终购买行为分解为4部分，从曝光到点击，从点击到有特定行为，从有特定行为到购买，从无特定行为到购买，每部分都有对应的任务进行学习。
3. SCM根据行为之间的依赖性和DPM中已有的分解行为，对共享曝光样本空间的任务进行级联，从而使CVR任务基于曝光样本空间间接学习，解决样本选择偏差问题。

## 损失函数

ESM2的损失包括三部分，CTR损失，CTAVR损失和CTCVR损失，即上图正红色所示的三部分。

DPM分解的4个任务中，有3个任务是基于曝光样本空间，分别是CTR任务(从曝光到点击)，CTAVR任务(从曝光到特定行为)，CTCVR任务(从曝光到购买)，因此这三个任务可以基于曝光样本计算出对应的损失，分别如式子(7)(8)(9)所示，其中，C表示点击label空间，A表示特定行为label空间，B表示购买label空间，+表示正样本，-表示负样本。

$$L_{ctr} = - \sum_{i \in C_+} \log p_i^{ctr} - \sum_{j \in C_-} \log(1 - p_j^{ctr}) \quad (7)$$

$$L_{ctavr} = - \sum_{i \in A_+} \log p_i^{ctavr} - \sum_{j \in A_-} \log(1 - p_j^{ctavr}) \quad (8)$$

$$L_{ctcvr} = - \sum_{i \in B_+} \log p_i^{ctcvr} - \sum_{j \in B_-} \log(1 - p_j^{ctcvr}) \quad (9)$$

整体损失由这三分部加权求和得到，如式子(10)所示。

$$L(\Theta) = w_{ctr} \times L_{ctr} + w_{ctavr} \times L_{ctavr} + w_{ctcvr} \times L_{ctcvr} \quad (10)$$

## 相关思考

- 思考1：行为链路中增加一个过程节点，为什么可以缓解样本稀疏？

虽然从点击到购买的样本是固定的，比如有100条，但当增加中间行为节点时，中间行为的数量高于购买，因此可以增加样本，以中间节点为加入购物车为例，当从点击到加入购物车的样本有1000条时，则点击-加入购物车-购买这条链路上，从点击到购买的样本量则有1000条，比未加入该中间节点的样本增加，从而缓解样本稀疏问题。

- 思考2：增加中间节点可以增加样本量，为什么不尽可能的在点击和购买之间多增加一些行为？

- 如果把中间行为拆分得足够细，比如把特定行为拆分成收藏、加入购物车、加入心愿单等，则每条链路上的数据量减少。另外，拆分得足够细和合并这些行为到一种行为，样本量总量不变。
- 如果是再链路中再增加一层行为，则会使链路变长，建模变得更复杂，ESM2的建模相比ESMM的复杂度增加，再在ESM2的基础上增加建模复杂度，其性价比有限。

具体公式推导：<https://zhuanlan.zhihu.com/p/524256107>

## ESMM和ESM2的缺点：

每个任务单独学一个模型，每个任务对应单独的模型无法利用任务之间的关联性，而且资源消耗大，后期维护成本高。

## 2.12 MMOE

多任务学习方法：

- 如ESMM、ESM2等模型，每个任务单独学一个模型；
- Share-bottom多任务DNN模型：为了节约成本和后续的可迭代性，hard参数共享机制被广泛采用，

也就是多任务学习中share-bottom的DNN模型结构。该模型受不同任务之间的相关性影响。当任务相关性强时，其共性强，模型中的share bottom部分可以有效提取其共性，因此可以实现较好的效果。但是随着不同任务之间的相关性减弱，hard参数共享机制将对效果提升有限。通过实验论证：不同任务之间存在一定的拉扯，而share-bottom结构在同时学习这些任务时，存在负迁移作用，使得效果受到影响。

MMoE模型的提出，有效解决了任务之间共性和特性的拉扯，使模型兼具任务共性和特性的有效表征。模型结构采用soft参数共享机制，相比share-bottom模型结构，在不增加参数量和成本消耗的情况下，可发挥不同task共同学习的优势，对可持续性迭代友好。

## MMoE模型结构

MMoE模型基于share bottom多任务DNN模型结构，对share bottom部分进行优化，引入专家(expert)概念，并结合门网络的使用，每个task运用gate得到各个expert的权重后加权求和得到task的输入，进行各个task层的学习。Gate的运用使得模型可以自动学习不同task的expert组合，对任务之间的共性和特性自动调整参数进行学习，从而使模型有效地挖掘任务之间的关系，提高表征能力。

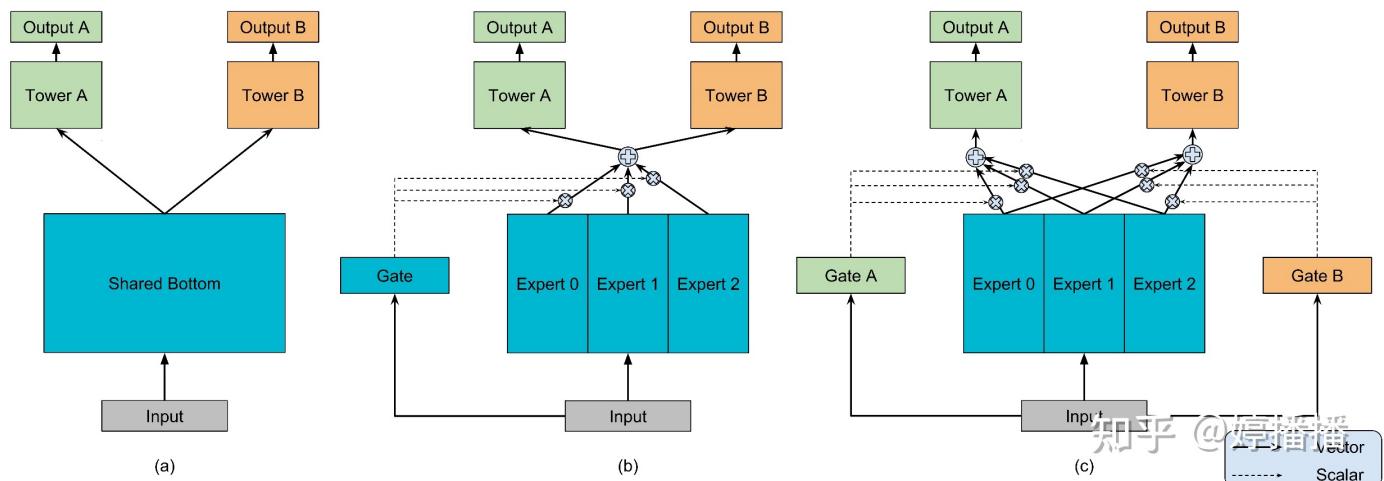


图3 MMoE模型结构 (a) share-bottom DNN多任务模型(hard参数共享模型) (b) 单gate MoE模型 (c) 多gate MoE模型

### gate

每个task有对应的gate，当所有task共享一个gate时，结构退化成单gate Moe模型，如图3(b)所示，这时模型难以学习不同task之间的差异性，和share-bottom多任务DNN模型结构没有本质差异。

Gate的输入和网络的输入保持一致，论文中对gate输入进行线性变化，在实际实现过程中，可根据具体需要，设计gate网络，引入非线性变化，提高gate网络的表征能力。

Gate的输入为模型的输入，经过gate网络变换后进行softmax操作，输出各个expert的权重，如式子(2)所示，其中  $W_{gk}$  为矩阵，也就是对输入进行线性变化。gate的网络可以根据业务实际情况设计，可以采用论文中线性的方式，也可以使用几层简单的全连接操作。

$$g^k(x) = \text{softmax}(W_{gk}x) \quad (2)$$

## 使用softmax激活函数

### expert

MMoE的模型结构中expert部分，每个expert对应独立的网络层，在反向传播时参数各自独立更新。不同expert的网络层相同，也可以不同，但最后一层的维度需保持相同，才可根据式子(3)进行加权组合得到task的输入。

根据当前task的gate得到的权重，对expert进行线性加权组合，得到task的输入，如式子(3)所示，其中  $f_i$  表示第*i*个expert层的操作。

$$y_k = h^k(f^k(x)), f^k(x) = \sum_{i=1}^n g^k(x)_i f_i(x) \quad (3)$$

- 特征表征经过不同的expert层后，得到更高级的特征表达，**expert层的作用是特征提取**。在模型对应的特征空间里，不同expert对应不同的子空间。在推荐场景，user对item的喜好源自不同的维度，比如标题、封面、具体内容、背景音乐、人物颜值等等，**当不同expert学到这些维度的特征子空间时，则每个expert进行了不同维度的特征提取**，这也是“专家”这个概念的含义和目的。

## 思考

1. expert网络结构一样，输入特征一样，是否会导致每个expert学出来的参数趋于一致，从而失去了ensemble的意义？

**【答】：**在网络参数随机初始化的情况下，不会发生问题中提到的问题。核心原因在于**数据存在multi-view**，只要每一个expert网络参数初始化是不一样的，就会导致每一个expert学到数据中不同的view（paddle官方实现就犯了这个致命错误）。微软的一篇论文中提到因为数据存在multi-view，训练多个DNN时，即使一样的特征，一样的超参数，只要简单的把参数初始化设置不一样，这多个DNN也会有差异。

-> 模型结构的设计中，和这个出发点类似的有attention中的multi-head，希望可以通过不同的head提取到不同子空间的兴趣，从而得到更充分的兴趣表征。

### 2. gate坍缩

在训练过程中，会出现gate学到的权重极度不均衡问题，各个task坍缩到单个expert上，即对task而言，少数expert的权重接近1，其它expert权重接近0，导致task只依赖于单个expert，从而无法发挥不同

expert的作用。极端情况，当某个expert的权重为1时，则模型退化成share-bottom结构。我同事在业务中使用MMoE也出现了同样的问题。

-> 产生这个问题的原因在于expert的学习速度不一样，导致gate在生成expert权重时，学得快的expert权重极高，学得慢的expert权重低。

-> 解决这个问题的方法在Youtube 2019年的论文[8]中被提出，即对gate使用dropout，使不同的expert的学习速度差异减小，加快学得慢的expert的学习速度。

### 3. 是否需要对gate输入stop gradient

对gate输入stop gradient，则使gate的学习更聚焦，可以减少对主网络学习的干预，embedding的更新只受主网络的影响，避免gate网络和主网络同时对embedding更新带来的一些不稳定甚至是拉扯问题，直接的反应就是模型收敛更快，可以比较快地达到稳定状态，这一点在运用中使用了对比实验论证。

**停止gate对embedding的梯度，在加快网络收敛的同时，可能会带来效果上的损失，也就是模型的效果上限降低。**理想情况下，主网络和gate网络对embedding的更新，方向会保持一致，不存在拉扯问题，所以当对gate输入stop gradient时，相当于去除了gate对embedding的影响贡献。具体这里是否会导致效果的下降，以及是否有更多的理论支撑，欢迎有兴趣的朋友一起来讨论。

## MMoE效果归因

### 1. 模型容量

MMoE的expert往往使用和share-bottom相同的结构，此时模型bottom层的容量将成倍增加。当每个expert采用和原有base中share-bottom层一样的参数量时，线上可获得一定收益，这是由模型容量增大带来的。

### 2. 更大地发挥模型能力，接近模型上限

在模型容量保持一致的情况下，和share-bottom DNN模型对比，MMoE模型的能力上限没有提高。

MMoE之所以能带来效果的提升，是因为模型在实际上很难达到理论的上限，只是不断地接近上限，而MMoE的模型结构设计，使模型在学习过程中能比share-bottom这种结构，更接近上限。

## 3. 特征工程怎么做

- 类别特征：用户的基本信息（如性别、年龄、省份、职业）；客群、是否新老客、是否花呗准入、是否有车/有房/有孩/已婚

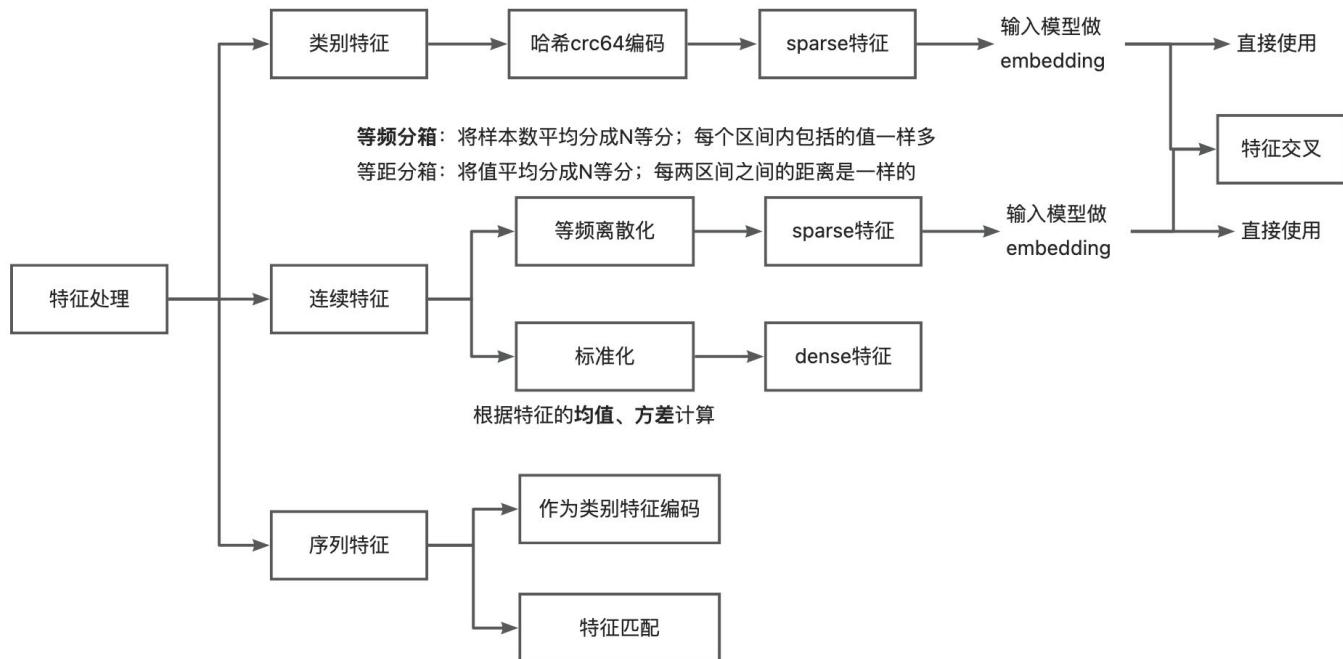
特别地，**预测的金额档位是类别特征**

- 连续特征：1) 基础数值特征：额度、利率、余额授信比、可贷额度；2) 统计类特征：【行为】近N天来访/点击/支用次数/支用金额、【收入支出】支付宝消费/转入金额、银行流水、

花借呗消费 【资产】 财富域基金/理财、 【负债】 微贷域支用/余额；3) 模型打分：支用率预估、营销活跃度

- 序列特征：网商-历史支用金额序列、借呗-历史支用金额序列

## 特征工程：



- 连续特征标准化：**1) 加快DNN梯度下降求最优解的速度（未归一化前特征的差距较大，导致梯度下降需要多次迭代才能收敛，甚至无法收敛）；2) 提高精度，如果一个特征的值域范围很大，在计算过程中主要取决于这个特征，其他特征的作用不明显。
- 序列特征的处理：**在贷款类的场景中，用户的历史支用序列对用户资金需求的刻画尤为重要。因此，把历史支用序列作为Category特征进行Embedding加入到模型，另外统计用户当前支用与历史支用序列的匹配次数，由此反映用户支用习惯。
- 连续特征离散化的优点：**1) 处理之后的特征对于异常数据具有较强的鲁棒性，模型会更稳定。2) 离散化相当于引入了非线性，提升模型的表达能力，增强拟合能力。3) 离散化之后方便进行特征交叉（两个连续特征之间不方便做特征交叉），通过使得模型要拟合的值大幅度降低，也降低了模型的复杂度。
- 稀疏特征分组embedding：**借鉴wnd的做法，分域做embedding，不同域选择不同的size。优点：  
1) 减少特征差异对模型的影响；2) 减小模型的参数空间，提升训练速度。【比如说性别的维度只有两维，而文本信息的维度可能有上千维，维度差异非常大，如果都放在一个embedding里都训练到相同的100维，对年龄来说会带来运算量的浪费，对文字来说会带来精度的损失。】

## 4. 模型实现细节

### 样本构造

借鉴word2vec负样本采样的思想，一条样本会分别对应28个档位，变成28条样本，用户实际支用对应的金额档位作为正样本，其他档位作为负样本。特别地，因为每个用户有可贷额度的限制，因此只保留预测金额档位 $\leq$ 可贷额度的那些正负样本。

### 网络结构：

输入层–embedding层–3层相同神经元的隐层–输出层

--> 相关论文实验表明：钻石型网络结构好，（比菱形网络、锥形网络表现好）

### 激活函数

隐层：relu，最后输出层：sigmod

-- 为什么DNN用relu的表现比tanh好？

按照上面的表示，在DNN中，输入  $x$ ，第一层的输出就是  $f_1 = f[W_1x + b_1]$ ，第二层的输出就是  $f_2 = f[W_2f[W_1x + b_1] + b_2]$ ，第三层的输出就是  $f_3 = f[W_3f[W_2f[W_1x + b_1] + b_2] + b_3]$ 。设最终损失为  $L$ ，我们来尝试从第三层开始，用BP算法推导一下损失对参数  $W_1$  的偏导数<sup>Q</sup>，看看会发生什么。

为了简洁起见，略过求导过程，最后的结果为  $\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} W_3 \frac{\partial f_2}{\partial a_2} W_2 \frac{\partial f_1}{\partial a_1} \frac{\partial a_1}{\partial W_1}$ 。我们常常说原始神经网络<sup>Q</sup>的梯度消失问题，这里的  $\frac{\partial f_3}{\partial a_3}$ 、 $\frac{\partial f_2}{\partial a_2}$  就是梯度消失的“罪魁祸首”。例如sigmoid的函数，它的导数的取值范围是  $(0, 0.25]$ ，也就是说对于导数中的每一个元素，我们都有  $0 < \frac{\partial f_3}{\partial a_3} \leq 0.25$ ， $0 < \frac{\partial f_2}{\partial a_2} \leq 0.25$ ，小于1的数乘在一起，必然是越乘越小的。这才仅仅是3层，如果10层的话，根据  $0.25^{10} \approx 0.000000954$ ，第10层的误差相对第一层卷积的参数  $W_1$  的梯度将是一个非常小的值，这就是所谓的“梯度消失”。

ReLU函数的改进就是它使得  $\frac{\partial f_3}{\partial a_3} \in \{0, 1\}$ ， $\frac{\partial f_2}{\partial a_2} \in \{0, 1\}$ ， $\frac{\partial f_1}{\partial a_1} \in \{0, 1\}$ ，这样的话只要一条路径上的导数都是1，无论神经网络是多少层，这一部分的乘积都始终为1，因此深层的梯度也可以传递到浅层中。

### 损失函数：

## 二分类常用的交叉熵损失函数

$$H(p, q) = \sum_x p(x) \log q(x)$$

p表示真实标记的分布，q则为训练后的模型的预测标记分布，交叉熵损失函数可以衡量p与q的相似性。

具体地，

### (1) 二分类

在二分的情况下，模型最后需要预测的结果只有两种情况，对于每个类别我们的预测得到的概率为p和 $1 - p$ ，此时表达式为(log的底数是e)：

$$L = \frac{1}{N} \sum_i L_i = \frac{1}{N} \sum_i -[y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)]$$

其中：

-  $y_i$  —— 表示样本*i*的label，正类为1，负类为0

-  $p_i$  —— 表示样本*i*预测为正类的概率

### (2) 多分类

多分类的情况实际上就是对二分类的扩展：

$$L = \frac{1}{N} \sum_i L_i = -\frac{1}{N} \sum_i \sum_{c=1}^M y_{ic} \log(p_{ic})$$

其中：

-  $M$  —— 类别的数量

-  $y_{ic}$  —— 符号函数（0或1），如果样本*i*的真实类别等于*c*取1，否则取0

-  $p_{ic}$  —— 观测样本*i*属于类别*c*的预测概率

## 为什么分类问题用交叉熵损失而不用均方误差损失？

参考：<https://blog.csdn.net/blogshinelee/article/details/103518097>

均方误差损失函数：  $MSELoss(y, \hat{y}) = \frac{1}{n} \sum_{n=1}^N (y_n - \hat{y}_n)^2$

### 1. 从损失函数角度来看，

- 交叉熵只与label类别有关， $\hat{y}_p$ 越趋近于1越好
- 均方误差不仅与 $\hat{y}_p$ 有关，还与其他项有关，它希望 $\hat{y}_1, \dots, \hat{y}_{p-1}, \hat{y}_{p+1}, \dots, \hat{y}_K$ 越平均越好，即在 $\frac{1-\hat{y}_p}{K-1}$ 时取得最小值

在这个前提下，均方误差损失可能会给出错误的指示，比如猫、老虎、狗的3分类问题，label为[1,0,0]，在均方误差看来，预测为[0.8,0.1,0.1]要比[0.8,0.15,0.05]好，即认为平均总比有倾向性要好，但这有

悖我们的常识。

## 2. 从反向传播来看,

对于最后一层的误差 (以多分类softmax激活函数为例, 对正样本y=1 预估)

交叉熵损失:  $\frac{\partial L}{\partial \hat{z}_p} = \frac{\partial L}{\partial \hat{y}_p} \cdot \frac{\partial \hat{y}_p}{\partial z_p} = \hat{y}_p - 1$ ,  $\hat{y}_p$  越接近于1, 偏导越接近于0, 即分类越正确越不需要更新权重, 分类越错误越需要对权重进行更新。

均方误差损失:

$\hat{y}_p$  不仅与  $z_p$  有关, 还与  $\{z_k | k \neq p\}$  有关, 这里仅看  $z_p$ , 则有

$$\frac{\partial \hat{y}_p}{\partial z_p} = \hat{y}_p(1 - \hat{y}_p)$$

所以  $\frac{\partial L}{\partial \hat{z}_p} = \frac{\partial L}{\partial \hat{y}_p} \cdot \frac{\partial \hat{y}_p}{\partial z_p} = -2\hat{y}_p(1 - \hat{y}_p)^2$

$\hat{y}_p = 0$  分类错误, 但不更新权重, 显然不合理。

## KL散度

-- 相对熵, 表示2个概率分布之间的差异性, 差异越大, 相对熵越大。

$$D(p||q) = H(p, q) - H(p) = \sum_i p(i) * \log \frac{q(i)}{p(i)}$$

- KL散度具有非对称性:  $D(p||q) \neq D(q||p)$
- KL散度大于等于0

在机器学习中, 需要评估label和predict之间的差距, 可以使用KL散度。但是因为训练集中的标签分布是已知确定的, 即  $H(y)$  不变, 所以在优化过程中只需关注交叉熵即可。即, 此时KL散度 = 交叉熵, 所以一般使用交叉熵作为loss。

--> 在真实分布变化的情况下, 同时拟合真实分布和预测分布, 并且使预测分布尽可能与真实分布一致时, 使用KL散度。

## 优化器:

Adam

## 防止过拟合：

early stopping、L2正则化、dropout、增大minibatch的数量，减少fc层的节点数量

-- 过拟合表现及原因：方差大，偏差小

## 训练过程优化

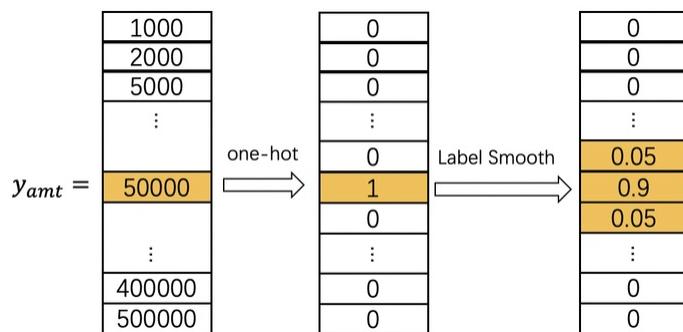
1. **FM+attention机制**：在训练过程中发现“历史平均支用-可贷额度”、“花借呗待还款金额-可贷额度”等组合特征重要性较高，为了让模型学习到更多特征交互的隐含信息，将一些金额类的连续型特征做自动分桶后，通过FM实现特征之间自动交叉的目的。同时加入Attention机制使得特征交叉的过程中能够找到较优的组合。通过FM+Attention，模型评估的top1 acc+1.7%。

- **样本加权**：将支用金额作为加权weight对样本进行加权，让模型更关注大额样本的信息。

-- 实际训练过程中，直接金额加权由于权重太大，会知道导致样本全预测到高金额，所以又做了一定的折中，最后采用的样本加权方式是sqrt(金额档位)。

参考：<https://aws.amazon.com/cn/blogs/china/characteristic-engineering-of-data-centric-ai-lecture-2/>

- 金额变成了类目，数值属性丢失（即当真实label是5W的时候，模型训练的时候无法知道，预测6W的概率要比10W高）。基于此，这里采用**Label Smooth**的策略。假设 $y_{amt}$ 为原真实金额档位标签，则Label Smooth的过程如下，



- loss选择适用于样本分布不平衡情况下的focal loss（正负样本比例在1:28以内，不算很严重的样本不均衡的情况，不需要特别处理）

## 5. 梯度下降

**梯度下降算法的原理**：目标函数  $J(\theta)$  关于参数  $\theta$  的梯度将是损失函数 (loss function) 上升最快的方向。而我们要最小化loss，只需要将参数沿着**梯度相反的方向**前进一个步长，就可以实现目标函数 (loss function) 的下降。

参数更新公式如下：

$$\theta \leftarrow \theta - \eta \cdot \nabla J(\theta)$$

梯度下降算法又可以分为批量梯度下降算法（Batch Gradient Descent），随机梯度下降算法（Stochastic Gradient Descent）和小批量梯度下降算法（Mini-batch Gradient Descent）。

- 对于**批量梯度下降算法**，其  $J(\theta)$  是在整个训练集上计算的，如果数据集比较大，可能会面临内存不足问题，而且其**收敛速度一般比较慢**。
- **随机梯度下降算法**是另外一个极端， $J(\theta)$  是针对训练集中的一个训练样本计算的，又称为在线学习，即得到了一个样本，就可以执行一次参数更新。所以**其收敛速度会快一些，但是有可能出现目标函数值震荡现象，因为高频率的参数更新导致了高方差**。
- **小批量梯度下降算法**是折中方案，选取训练集中一个小批量样本（一般是2的倍数，如32, 64, 128等）计算，这样可以保证训练过程更稳定，而且采用批量训练方法也可以利用矩阵计算的优势。这是目前最常用的梯度下降算法。

★ 梯度下降算法中一个重要的参数是**学习速率**，适当的学习速率很重要：学习速率过小时收敛速度慢，而过大时导致训练震荡，而且可能会发散。理想的梯度下降算法要满足两点：**收敛速度要快；而且能全局收敛**。

## 入门计算基本概念

### 指数加权移动平均数

其针对的序列数据，比如  $t$  时刻的观测值为  $x(t)$ ，那么评估  $t$  时刻的移动平均值为：

$$v(t) \leftarrow \beta \cdot v(t-1) + (1 - \beta) \cdot x(t)$$

其中  $v(t-1)$  是上一时刻的移动平均值，其实也可以看成历史积累量，一般  $v(0) = 0$ ，而  $\beta$  是一个系数，其在0~1之间，可以看到移动平均值是按比例合并历史量与当前观测量。现在我们展开上面的式子：

$$v(0) = 0$$

$$v(1) = \beta \cdot v(0) + (1 - \beta) \cdot x(1)$$

$$v(2) = \beta \cdot v(1) + (1 - \beta) \cdot x(2) = \beta(1 - \beta) \cdot x(1) + (1 - \beta)^2 \cdot x(2)$$

$$v(t) = \beta \cdot v(t-1) + (1 - \beta) \cdot x(t) = \sum_{i=1}^t \beta^{t-i} (1 - \beta) \cdot x(i)$$

展开之后，我们可以直观的看到对于每个  $x$  值，其权重是不一样的，实际上是指数递减的，从当前往后指数递减，所以距离时刻较近的数据会对当前值影响较大，这样计算的好处是平均数会比较平稳。由于权重指数衰减，所以移动平均数只是计算比较相近时刻数据的加权平均数，一般可认为这个时间范围为  $\frac{1}{1-\beta}$ ，比如  $\beta = 0.9$ ，你可以近似认为只是平均了10时刻之内的数据而已，因为再往前权重太小了，基本没影响了。如果你熟悉级数的话，也可以计算出权重和是近似为1的，这是在无穷的情况下，但是在前期计算时，权重之和是会小于1的，为了解决这个问题，指数加权移动平均数会引入偏差修正：

$$v(t) \leftarrow \frac{v(t)}{1 - \beta^t}$$

其实很好理解，就是前期计算时要放大一下计算结果，而后期不需要，此时  $1 - \beta^t$  的值也接近1了。

## 各种优化器介绍

### 1. FTRL

主要用于CTR预测的在线训练，成千上万维度导致大量稀疏特征。一般希望模型参数更加稀疏，但是简单的L1正则无法真正做到稀疏，一些梯度截断方法（TG）的提出就是为了解决这个问题，在这其中FTRL是兼备精度和稀疏性的在线学习方法。FTRL的基本思想是将接近于0的梯度直接置零，计算时直接跳过以减少计算量。

参考：<https://blog.csdn.net/songbinxu/article/details/79633101>

### 2. Adam

$$m \leftarrow \beta_1 \cdot m + (1 - \beta_1) \cdot \nabla J(\theta)$$

$$s \leftarrow \beta_2 \cdot s + (1 - \beta_2) \cdot \nabla J(\theta) \odot \nabla J(\theta)$$

$$m \leftarrow \frac{m}{1 - \beta_1^t}$$

$$s \leftarrow \frac{s}{1 - \beta_2^t}$$

$$\theta \leftarrow \theta - \frac{\eta}{\sqrt{s + \varepsilon}} \odot m$$

**m和s的初始值**

可以看到前两项和Momentum和RMSprop是非常一致的，由于m的初始值一般设置为0，在训练初期其可能较小，第三和第四项主要是为了放大它们。最后一项是参数更新。其中超参数的建议值是:  $\beta_1 = 0.9, \beta_2 = 0.999, \varepsilon = 1e - 8$ 。Adm是性能非常好的算法，在TensorFlow其实现如

**优点:**

- a. 实际上只需要保存梯度的均值，所以基本不需要额外的内存
- b. 不需要人工设定全局学习率 $\eta$

### 3. AdamGrad

在训练迭代过程，其学习速率是逐渐衰减的，经常更新的参数其学习速率衰减更快，这是一种自适应算法。

$$s \leftarrow s + \nabla J(\theta) \odot \nabla J(\theta)$$

$$\theta \leftarrow \theta - \frac{\eta}{\sqrt{s + \varepsilon}} \odot \nabla J(\theta)$$

**优点:**

- a. 每个参数都有自己的学习率。
- b. 训练初期s (gt平方累计和) 较小，学习率较大，能够加速训练；后期较大，学习率较小，能够约束梯度。【比较陡的方向s比较大，其学习速率将衰减得更快，这有利于参数沿着更接近坡底的方向移动，从而加速收敛。】
- c. 适合处理稀疏梯度。【对于稀疏梯度，该平方和一般会比较小，使得参数的学习率偏大】

**缺点:** AdaGrad训练后期学习速率很小，导致训练过早停止，因此在实际中AdaGrad一般不会被采用

### 4. Adadelta

- a. 训练初中期，加速效果不错，很快
- b. 训练后期，反复在局部最小值附近抖动

RMSprop优化器虽然可以对不同的权重参数自适应的改变学习率，但仍要指定超参数  $\eta$ ，  
AdaDelta优化器对RMSProp算法进一步优化：AdaDelta算法额外维护一个状态变量  $\Delta x_t$ ，并使用  $RMS[\Delta x]_t$  代替 RMSprop 中的学习率参数  $\eta$ ，使AdaDelta优化器不需要指定超参数  $\eta$

AdaDelta 的计算公式如下：

$$E[g^2]_t = \rho * E[g^2]_{t-1} + (1 - \rho) * g_t^2$$

$$E[\Delta x^2]_{t-1} = \rho * E[\Delta x^2]_{t-2} + (1 - \rho) * \Delta x_{t-2}^2$$

$$RMS[g]_t = \sqrt{E[g^2]_t + \epsilon}$$

$$RMS[\Delta x]_{t-1} = \sqrt{E[\Delta x^2]_{t-1} + \epsilon}$$

$$\Delta x_t = \frac{RMS[\Delta x]_{t-1}}{RMS[g]_t}$$

$$\Delta \theta_t = -\Delta x_t * g_t$$

$g_t$ 、 $\Delta x$  的初始值都为0，这就导致算法开始时得到的  $E[g^2]_t$  和  $E[\Delta x^2]_{t-1}$  都很小，因此，在实际工作中，我们往往会：

用  $\frac{RMS[g]_t}{1-\rho_1^t}$  代替  $RMS[g]_t$

用  $\frac{RMS[\Delta x]_t}{1-\rho_2^t}$  代替  $RMS[\Delta x]_t$

$\rho_1^t$  和  $\rho_2^t$  中的  $t$  表示  $t$  次方，因为  $\rho_1$  和  $\rho_2$  小于1，因此  $t$  越大，两个式子中的分母越接近于1

当  $t$  很小时，分母较小，因此可以放大  $RMS[g]_t$  和  $RMS[\Delta x]_t$ ，从而解决上述问题

参考：<https://zhuanlan.zhihu.com/p/31630368>

## 6. 过拟合/欠拟合

### 欠拟合

**高偏差**，模型训练不足，对数据拟合不够

- 解决欠拟合的方法：模型训练前期出现欠拟合情况是正常的，后期欠拟合通过增加网络复杂度或者在模型中增加特征解决

### 过拟合

**高方差**，模型在训练集上拟合效果很好，在测试集上泛化效果很差

- 造成原因主要有以下几种：

- 训练数据集样本单一、样本不足。如果训练样本只有负样本，然后那生成的模型去预测正样本，这肯定预测不准。所以训练样本要尽可能的全面，覆盖所有的数据类型。
- 训练数据中噪声干扰过大。噪声指训练数据中的干扰数据。过多的干扰会导致记录了很多噪声特征，忽略了真实输入和输出之间的关系。
- 模型过于复杂。模型太复杂，已经能够“死记硬背”记下了训练数据的信息，但是遇到没有见过的数据的时候不能够变通，泛化能力太差。我们希望模型对不同的数据都有稳定的输出。模型太复杂是过拟合的重要因素。

- 解决过拟合的方法：

- 获取和使用更多的数据（数据集增强）——解决过拟合的根本性方法
- L1/L2正则化
- 提前终止(Early stopping)
- Dropout
- 降低特征的数量——删除冗余特征

### 1) L1正则化 -- 用于特征选择

优点：

- 得到的参数的先验概率分布满足拉普拉斯分布；（拉普拉斯分布在极值点（0点）处是一个尖峰，因此 $w=0$ 的可能性较大）
- 鲁棒性更强，对异常值不敏感；
- 适合获得稀疏解，可用于特征选择等场景；

在原始的损失函数后面加上一个L1正则化项，即全部权重  $w$  的绝对值的和，再乘以 $\lambda/n$ 。则损失函数变为：

$$C = C_0 + \frac{\lambda}{n} \sum_i |w_i|$$

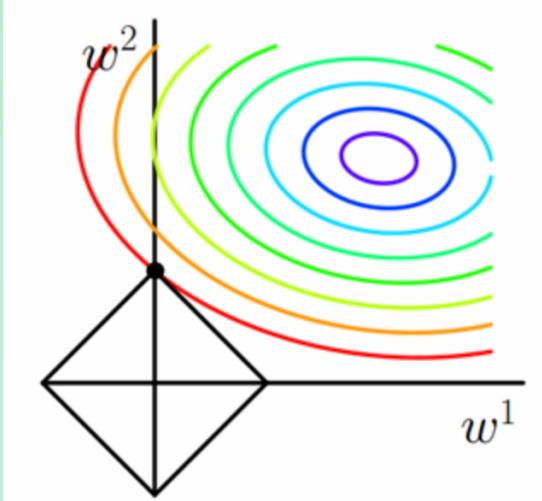
从梯度下降的过程可以理解，L1正则化使得权重  $w$  往0靠，使网络中的权重尽可能为0，也就相当于减小了网络复杂度，防止过拟合。

这也就是L1正则化会产生更稀疏的解的原因。此处稀疏性指的是最优化中的一些参数为0。L1正则化的稀疏性质已经被广泛地应用于特征选择机制，从可用的特征子集中选择出有意义的特征。

---

另一种直观的理解，加入L1正则项后，此时我们的任务变成在L1约束下求出J0取最小值的解。

- 考虑二维的情况，即只有两个权值  $w_1$  和  $w_2$ ，此时  $L = |w_1| + |w_2|$  对于梯度下降法，求解  $J_0$  的过程可以画出等值线，同时 L1 正则化的函数  $L$  也可以在  $w_1$ 、 $w_2$  的二维平面上画出来。如下图：



- 上图中等值线是  $J_0$  的等值线，黑色方形是  $L$  函数的图形。在图中，当  $J_0$  等值线与  $L$  图形首次相交的地方就是最优解。上图中  $J_0$  与  $L$  在  $L$  的一个顶点处相交，这个顶点就是最优解。注意到这个顶点的值是  $(w_1, w_2) = (0, w_2)$ 。可以直观想象，因为  $L$  函数有很多突出的角（二维情况下四个，多维情况下更多）， $J_0$  与这些角接触的机率会远大于与  $L$  其它部位接触的机率，而在这些角上，会有很多权值等于 0，这就是为什么 L1 正则化可以产生稀疏模型，进而可以用于特征选择。
- 而正则化前面的系数  $\lambda$ ，可以控制  $L$  图形的大小。 $\lambda$  越小， $L$  的图形越大（上图中的黑色方框）； $\lambda$  越大， $L$  的图形就越小，可以小到黑色方框只超出原点范围一点点，这是最优点的值  $(w_1, w_2) = (0, w_2)$  中的  $w_2$  可以取到很小的值。

## 2) L2正则化

优点：

- 得到的参数  $w$  的先验概率分布满足高斯分布（高斯分布在极值点（0点）处是平滑的，这也是  $w$  趋向于 0 但不等于 0 的原因）
- 提高模型泛化能力
- 让优化求解变得稳定很快速（这是因为加入了 L2 范式之后，满足了强凸：保证函数在每点有一个二次下界）

通常被称为权重衰减（weight decay），就是在原始的损失函数后面再加上一个 L2 正则化项，即全部权重  $w$  的平方和，再乘以  $\lambda/2n$ 。则损失函数变为：

$$C = C_0 + \frac{\lambda}{2n} \cdot \sum_{\text{所有 } i} w_i^2$$

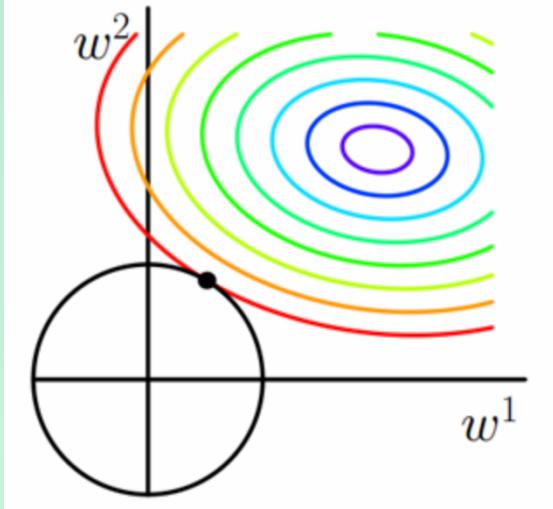
在梯度下降过程中，权重  $w$  将逐渐减小，趋向于 0 但不等于 0。这也就是权重衰减（weight decay）的由来。

L2 正则化起到使得权重参数  $w$  变小的效果，为什么能防止过拟合呢？因为更小的权重参数  $w$  意味着模型的复杂度更低，对训练数据的拟合刚刚好，不会过分拟合训练数据，从而提高模型的泛化能力。

-- 若参数很大，那么只要数据偏移一点点，就会对结果造成很大的影响；但如果参数足够小，数据偏移得多一点也不会对结果造成什么影响，即抗扰动能力强

直观理解：

- 同理，又L2正则化损失函数： $J = J_0 + \lambda \sum_w w^2$ ,同样可画出其在二维平面的图像，如下：



- 二维平面下L2正则化的函数图形是个圆，与方形相比，被磨去了棱角。因此 $J_0$ 与 $L$ 相交时使得 $w_1$ 或 $w_2$ 等于零的机率小了许多，这就是为什么L2正则化不具有稀疏性的原因。

## L1、L2的参数 $\lambda$ 如何选择好？

- 以L2正则化参数为例：从公式(8)可以看到， $\lambda$ 越大， $\theta_j$ 衰减得越快。另一个理解可以参考L2求解图， $\lambda$ 越大，L2圆的半径越小，最后求得代价函数最优点时各参数也会变得很小；当然也不是越大越好，太大容易引起欠拟合。L1同理
- 经验

从0开始，逐渐增大 $\lambda$ 。在训练集上学习到参数，然后在测试集上验证误差。反复进行这个过程，直到测试集上的误差最小。一般的说，随着 $\lambda$ 从0开始增大，测试集的误分类率应该是先减小后增大。交叉验证的目的就是为了找到误分类率最小的那个位置。建议一开始将正则项系数 $\lambda$ 设置为0，先确定一个比较好的learning rate。然后固定该learning rate，给 $\lambda$ 一个值（比如1.0），然后根据validation accuracy，将 $\lambda$ 增大或者减小10倍，增减10倍是粗调节，当你确定了 $\lambda$ 的合适的数量级后，比如 $\lambda=0.01$ ，再进一步地细调节，比如调节为0.02, 0.03, 0.009之类。

L1正则、L2正则详细推导：

<https://zhuanlan.zhihu.com/p/72038532>

<https://www.cnblogs.com/zingp/p/10375691.html>

<https://blog.csdn.net/rosefun96/article/details/103845052>

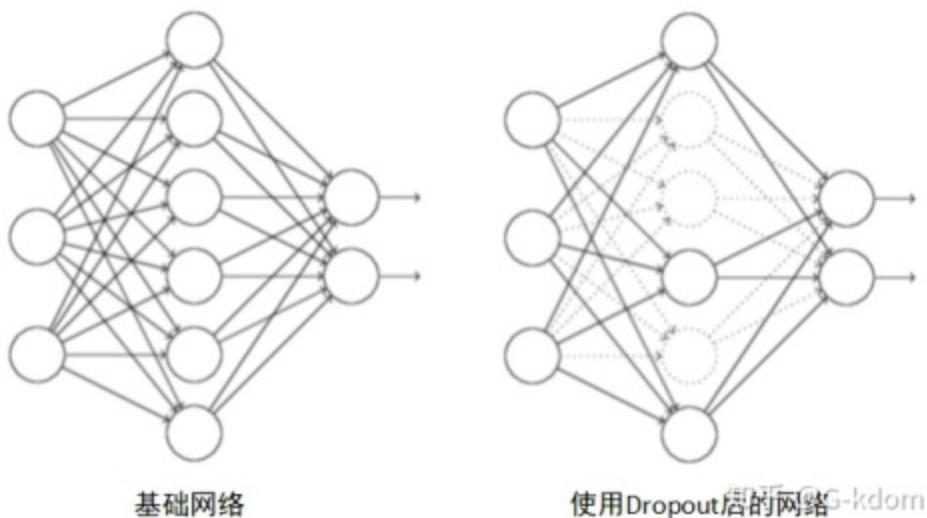
### 3) early stop

截断训练轮数epoch, 作用是当模型在每一个Epoch结束时（一个Epoch集为对所有的训练数据的一轮遍历）在验证集上的性能（accuracy）不再增加的时候就停止训练，从而达到充分训练的作用，又避免过拟合。

Early Stopping缺点：可能会存在停止训练时优化损失函数的值还比较大。

### 4) Dropout

在训练网络时用的一种技巧（trick），相当于在隐藏单元增加了噪声。Dropout 指的是在训练过程中每次按一定的概率（比如50%）随机地“删除”一部分隐藏单元（神经元）。所谓的“删除”不是真正意义上的删除，其实就是将该部分神经元的激活函数设为0（则激活函数的输出为0），让这些神经元不计算而已。



具体地，在每轮迭代中，网络中的每个神经元以  $p$  的概率被丢弃（每一层的 dropout 概率可能不尽相同，经验值输入层的  $p=0.2$ ，而隐藏层的  $p=0.5$ ）。当训练完成后，模型需要用全部神经元进行预测。

Dropout为什么有助于防止过拟合呢？

- (a) Dropout 可以天然理解为不同模型架构的集成方法，它提供了一种非常廉价的 Bagging 集成近似方法。每次更新时关注的神经元都不相同，重点更新的权重也不相同，因此最后集成在一起就能达到正则化的效果。
- (b) 它消除或者减弱了神经元节点间的联合，降低了网络对单个神经元的依赖，从而增强了泛化能力。

### 5) batch\_size

一般来说，在合理的范围之内，越大的 batch size 使下降方向越准确，震荡越小

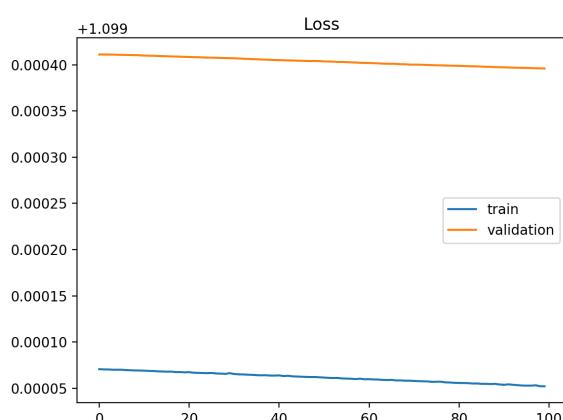
batch size的问题一般是较大会有比较好的效果，一是更快收敛，二是可以躲过一些局部最优点。batch size增大，处理相同的数据量速度加快，越大的 batch size 使下降方向越准确，震荡越小；但是也不是一味地增加batch size就好，太大的batch size 容易陷入局部最优的情况，泛化性不好。较小的 batch size 引入的随机性更大，难以达到收敛，但是可能不会陷入局部最优，另一方面，当类别较多时，可能会使得网络有明显的震荡，所以，Batch size过小会使Loss曲线振荡的比较大。

因此基于上述两种情况，batch size要调试到合适的数值；过大的batchsize会让网络收敛到不好的局部最优点；过小的batchsize训练速度慢，训练不收敛；具体的batch size需要根据训练集数据内容和数量进行调试。大小一般按照2的次幂规律选择，这是为了硬件计算效率考虑的。

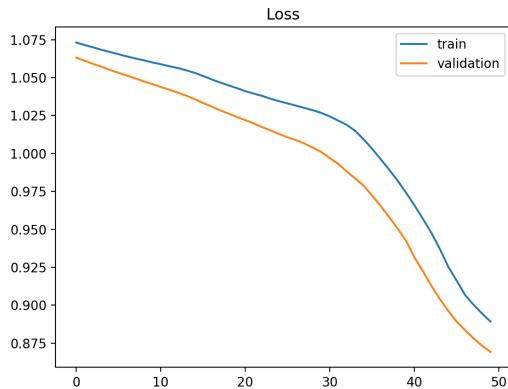
## 通过loss曲线诊断神经网络模型

### 欠拟合

- 1) 仅根据training loss就可以判断。这个training loss下降的非常平缓以致于好像都没有下降，这说明模型根本没有从训练集学到什么东西。

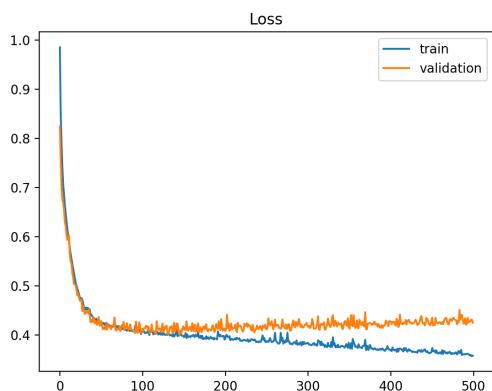


- 2) 在训练结束时候training loss还在继续下降，这说明还有学习空间，模型学习不够充分就结束了。



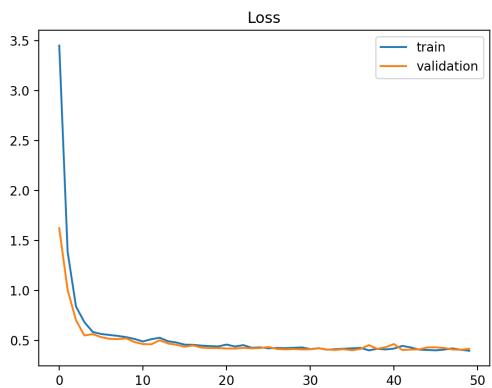
## 过拟合

training loss一直在不断地下降，而validation loss在某个点开始不再下降反而开始上升了，这就说明overfit，我们应该在这个拐点处停止训练。



## 完美拟合

Good fit是我们的目标，它在loss曲线上的特点是training loss和validation loss都已经收敛并且之间相差很小很小。如下图所示，模型在20轮过后，两个loss曲线都开始收敛，而且两者之间并没有肉眼的差距。通常training loss会更小，这样他们之间就会有个gap，这个gap叫做generalization gap。



# 7. DNN反向传播公式推导及求解

## 公式推导

参考: [https://blog.csdn.net/xuan\\_liu123/article/details/83660316](https://blog.csdn.net/xuan_liu123/article/details/83660316)

## 求解

### 1) 最小二乘法

最小二乘法的思想就是求下列二乘公式的极值。通过直接矩阵运算求出参数w的最优解。

误差方程为:

$$\text{Error}(w|X, y) = (Xw - y)^T (Xw - y)$$

这个公式就是二乘公式

其最优解为:

$$w = (X^T X)^{-1} X^T y$$

具体来说, 最小二乘法的矩阵公式是  $(A^T A)^{-1} A^T b$ , 这里的  $A$  是一个矩阵,  $b$  是一个向量. 如果有离散数据点,  $(x_1, y_1), \dots, (x_n, y_n)$ , 而想要拟合的方程又大致形如  $a_0 + a_1 x^1 + a_2 x^2 + \dots + a_m x^m$ , 那么,  $A$  就是一个  $n \times (m + 1)$  的矩阵, 第  $i$  行的数据点分别是  $[x_i^0, x_i^1, \dots, x_i^m]$ , 而  $b$  则是一个向量, 其值为  $[y_1, \dots, y_n]^T$ . 而又已知, 计算一个矩阵的逆是相当耗费时间的, 而且求逆也会存在数值不稳定的情况 (比如对希尔伯特矩阵求逆就几乎是不可能的). 因而这样的计算方法有时不值得提倡.

参考: <https://www.zhihu.com/question/20822481>

### 2) 梯度下降法

梯度下降法可以看作是 更简单的一种 求最小二乘法最后一步解方程 的方法。不直接求解整个矩阵, 而是通过每个样本的梯度更新w, 通过多次迭代后从而找到w的最优解。

### 3) 牛顿法

牛顿法主要应用在两个方面, 1: 求方程的根; 2: 最优化。 (应用1跟参数最优求解无关, 这里不讨论。)

应用2: 最优化 -> 李航《机器学习》P219

优化求解过程中, 需要求解  $p_k = -H_k^{-1} g_k$ , 其中需要计算逆矩阵  $H_k^{-1}$ , 计算比较复杂。

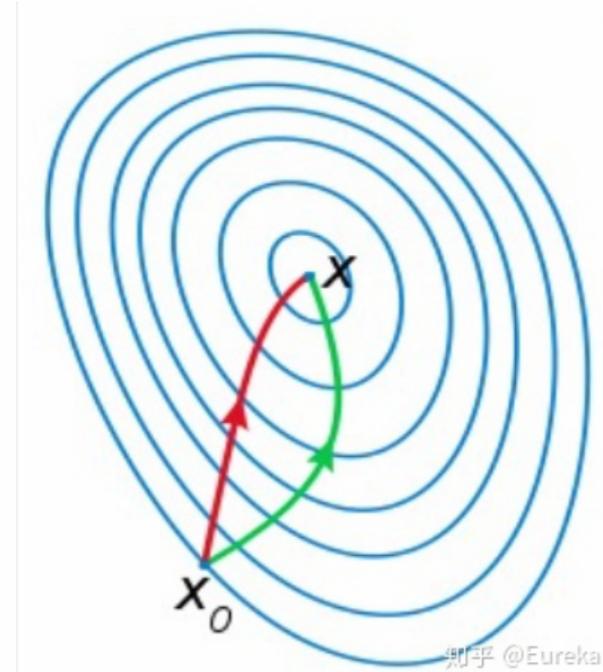
## ★ 牛顿法 VS 梯度下降法

梯度下降法和牛顿法相比，两者都是迭代求解，不过梯度下降法是梯度求解，而牛顿法是用二阶的海森矩阵的逆矩阵求解。相对而言，使用牛顿法收敛更快（迭代更少次数）。但是每次迭代的时间比梯度下降法长。

$$\text{梯度下降法: } x^{(k+1)} = x^{(k)} - \lambda \nabla f(x^k)$$

$$\text{牛顿法: } x^{(k+1)} = x^{(k)} - \lambda (H^{(k)})^{-1} \nabla f(x^k)$$

如下图是一个最小化一个目标方程的例子，红色曲线是利用牛顿法迭代求解，绿色曲线是利用梯度下降法求解。



至于为什么牛顿法收敛更快，通俗来说梯度下降法每次只从你当前所处位置选一个坡度最大的方向走一步，牛顿法在选择方向时，不仅会考虑坡度是否够大，还会考虑你走了一步之后，坡度是否会变得更大。所以，可以说牛顿法比梯度下降法看得更远一点，能更快地走到最底部。更多的可见：

### 4) 拟牛顿法

在牛顿法的迭代中，需要计算海森矩阵的逆矩阵  $H^{-1}$ ，这一计算比较复杂，考虑用一个  $n$  阶矩阵  $G_k = G(x^{(k)})$  来近似代替  $H_k^{-1} = H^{-1}(x^{(k)})$ 。这就是拟牛顿法的基本想法。

过程太复杂，选择放弃。。。

### 梯度下降代码实现

1. 牛客网AI5 使用梯度下降对逻辑回归进行训练
2. 算法题：求n的开方（二分法、梯度下降法、牛顿法）

## 8. DNN为什么能够学到特征交叉？

DNN全连接本质是将输入向量的所有元素进行加权求和。

DNN学习的是隐式的特征交叉，虽然是每个特征（位）在每一层乘以不同权重，但是彼此之间是有关联的，DNN能够通过一定深度捕捉到隐式特征共现。（每一层神经元体现特征域间的加性组合，通过激活函数的非线性变换后再通过传播到下一层，就形成了隐形的特征交叉）

$$w_{21} \cdot \frac{1}{1 + e^{-w_{11}x_1}} + w_{22} \cdot \frac{1}{1 + e^{-w_{12}x_2}} = \frac{w_{22}(1 + e^{-w_{11}x_1}) + w_{21}(1 + e^{-w_{12}x_1})}{(1 + e^{-w_{11}x_1})(1 + e^{-w_{12}x_2})}$$

分母  $(1 + e^{-w_{11}x_1})(1 + e^{-w_{12}x_2})$  就隐含了特征交叉的信息。

## 9. 激活函数

激活函数的作用是在神经网络中给神经元引入非线性。

### sigmod、tanh、relu激活函数公式

#### softmax激活函数

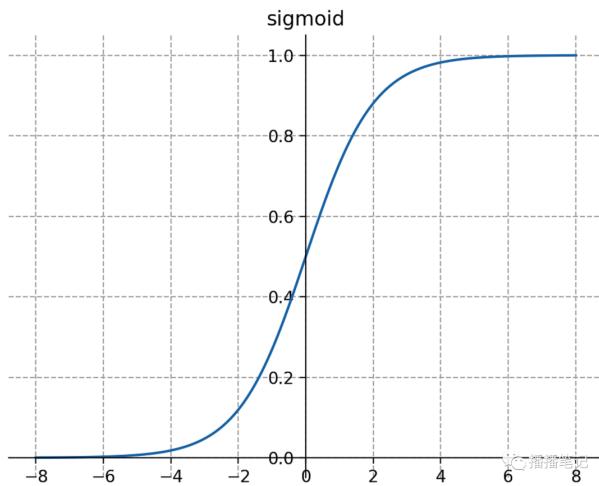
也称为归一化指数函数。

作用：将 $(-\infty, +\infty)$ 的几个实数映射到 $(0,1)$ 之间且之和为1，以获得某种概率解释。

数学意义：对向量进行归一化，凸显其中最大的值并抑制远低于最大值的其他分量。

--> 反向传播推导：<https://www.cnblogs.com/chumingqian/articles/11534581.html>

#### sigmoid激活函数



**特点：**

- (1) 连续光滑、严格单调；
- (2) 输出范围为(0,1), 以(0, 0.5)为对称中心；
- (3) 当输入趋于负无穷时, 输出趋近于0, 当输入趋于正无穷时, 输出趋近于1;
- (4) 输入在0附近时, 输出变化趋势明显, 输入离0越远, 变化趋势越平缓且逐渐趋于不变。

**优点：**

- (1) 输出范围为(0,1), 适合作为概率的使用；
- (2) 求导方便, 如式子(5)所示, 不需要额外的计算量。

$$f'(x) = f(x)(1 - f(x)) = -(f(x) - \frac{1}{2})^2 + \frac{1}{4} \in (0, \frac{1}{4}] \quad (5)$$

**缺点：**

- (1) 当输入离0较远时, 输出变化非常平缓, 容易陷入梯度饱和状态, 导致梯度消失问题;  
-- 梯度饱和: Sigmoid函数, tanh函数都存在两端软饱和的问题, 即随着特征值越大或越小, 在该处的导数越接近于0。这样直接带来的问题便是在反向传播过程中, 可能部分权重更新的非常小, 如果神经网络层数比较深的话, 可能会导致部分权重基本不更新, 也就是我们讲的梯度消失的问题。
- (2) 以(0, 0.5)为对称中心, 原点不对称, 容易改变输出的数据分布;
- (3) 导数取值范围为(0, 0.25] (推导过程见式子(5)), 连乘后梯度呈指数级减小, 所以当网络加深时, 浅层网络梯度容易出现梯度消失;
- (4) 输出总是正数, 使反向传播时参数w的梯度全正或全负, 梯度下降出现zigzag形状, 导致网络收敛速度慢, 详细原因见分析1;
- (5) 计算包括指数项, 耗时多。

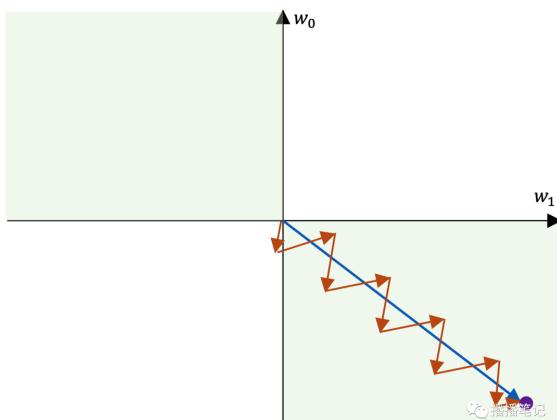
## 分析1 sigmoid激活函数zigzag现象（震荡现象）

### 1 参数w的梯度方向特点

参数w的梯度为  $\frac{\alpha L}{\alpha w^1} = \frac{\alpha L}{\alpha a^2} \frac{\alpha a^2}{\alpha z^1} \frac{\alpha z^1}{\alpha w^1} = \frac{\alpha L}{\alpha a^2} f'(z^1) a^1, a^1 = x$ ，由于sigmoid的导数  $f'$  恒为正， $a^1$  为上一层隐层经过sigmoid的输出，同样恒为正，因此参数w的梯度方向取决于  $\frac{\alpha L}{\alpha a^2}$  的梯度方向，而对同一层的所有参数w而言， $\frac{\alpha L}{\alpha a^2}$  是相同的，所以同一层的参数w在更新时要么同时增大，要么同时减小，没有自己独立的正负方向。

### 2 为什么参数的梯度方向一致容易造成zigzag现象

当所有梯度同为正或者负时，参数在梯度更新时容易出现zigzag现象。zigzag现象如图4所示，不妨假设一共两个参数， $w_0$  和  $w_1$ ，紫色点为参数的最优解，蓝色箭头表示梯度最优方向，红色箭头表示实际梯度更新方向。由于参数的梯度方向一致，要同正，要同负，因此更新方向只能为第三象限角度或第一象限角度，而梯度的最优方向为第四象限角度，也就是参数  $w_0$  要向着变小的方向， $w_1$  要向着变大的方向，在这种情况下，每更新一次梯度，不管是同时变小(第三象限角度)还是同时变大(第四象限角度)，总是一个参数更接近最优状态，另一个参数远离最优状态，因此为了使参数尽快收敛到最优状态，出现交替向最优状态更新的现象，也就是zigzag现象。



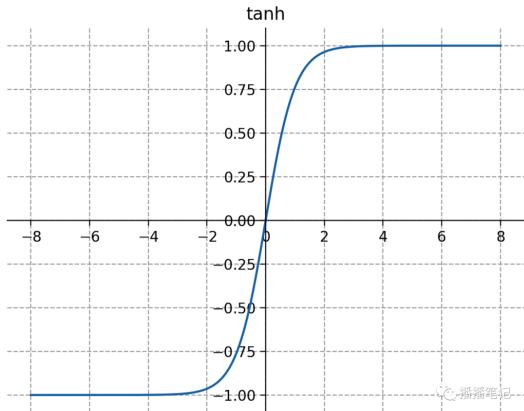
sigmoid函数在什么情况下会出现zigzag现象呢？从上面的分析可以看出，当梯度更新的最优方向不满足所有参数梯度正负向一致时，也就是有的参数梯度正向，有的参数梯度负向，容易出现zigzag现象。

两个参数时，出现zigzag现象的梯度更新最优方向为图4中的绿色背景部分，即第二象限和第四象限。在深度学习中，网络参数非常巨大，形成高维的空间，梯度更新的最优方向非常容易出现不同参数的梯度正负向不一致的情况，也就更容易造成zigzag现象。

抛开激活函数，当参数量纲差异大时，也容易造成zigzag现象，其中的原因和sigmoid函数造成的zigzag现象，是相通的。

## tanh激活函数

tanh 函数形状和 sigmoid 函数很相似，由 tanh 和 sigmoid 函数的定义有  $\tanh(x) = 2\text{sigmoid}(2x) - 1$ ，因此tanh函数可以由sigmoid函数伸缩平移得到，所以tanh函数一些特点和sigmoid函数相同。



### 特点：

- (1) 连续光滑、严格单调；
- (2) 输出范围为(-1,1)，以(0, 0)为对称中心，均值为0；
- (3) 输入在0附近时，输出变化明显；输入离0越远，输出变化越小最后输出趋近于1不变。

### 优点：

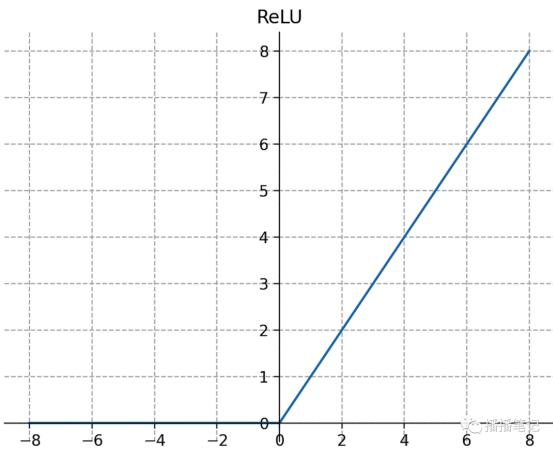
- (1) 输出关于原点对称，0均值，因此输出有正有负，可以规避zigzag现象；
- (2) 原点对称本身是一个很好的优点，有益于网络的学习。

### 缺点：

- (1) 存在梯度饱和的问题，导致梯度消失。

(2) 当网络层数增加时，存在梯度消失问题，tanh的导数计算为  $f' = \frac{4e^{2x}}{(e^{2x}+1)^2}$ ，取值范围为(0,1]，虽然取值范围比sigmoid导数更广一些，可以缓解梯度消失，但仍然无法避免随着网络层数增多梯度连乘导致的梯度消失问题。

## ReLU激活函数



ReLU函数，在负轴上也存在梯度消失的问题，不过负轴为零也带来了一些好处：即这样会使得被激活的神经元非常稀疏，这和我们熟知的认知神经科学中关于人类大脑学习时生物神经元激活的稀疏性具有相似性。

ReLU函数的分段性使其具有如下优点：

- (1) 输入 $>0$ 时保持梯度为恒定值不衰减，从而缓解梯度消失问题；
- (2) 输入 $<0$ 时导数为0，当神经元激活值为负值时，梯度不再更新，**增加了网络的稀疏性，从而使模型更具鲁棒性**，这点类似dropout但不完全等同dropout（分析2）；
- (3) 计算速度快，ReLU函数的导数是if-else的实现逻辑，计算非常方便快速。

**缺点：**

- (1) 输入 $>0$ 时梯度为1，可能导致爆炸问题；
- (2) 输入 $<0$ 时导数为0，一旦神经元激活值为负，则神经元进入永久性dead状态，梯度不再更新，导致梯度消失问题，学习率过大容易导致所有神经元都进入dead状态，所以**需设置较小的学习率**；
- (3) ReLU的输出均值大于0，容易改变输出的分布，可以通过**batch normalization**缓解这个问题。

相比sigmoid和tanh，ReLU可明显改善梯度消失的问题，且计算高效，因此在业界被广泛使用。

## 分析2 ReLU稀疏性和dropout的区别

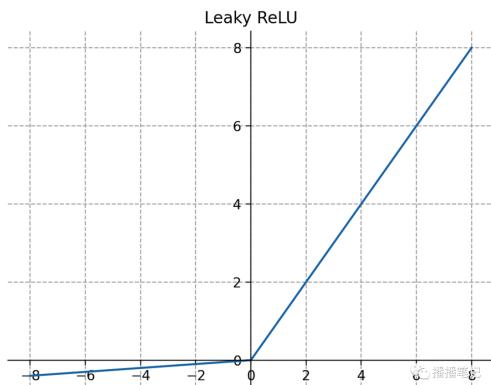
ReLU对神经元的选择，类似dropout，但两者原理不同。

- ReLU根据神经元的输出进行选择，多层神经元的选择相当于在原始输入数据层增加稀疏度，这也是ReLU增加模型鲁棒性的原因，增加数据稀疏度可以起到去除噪音的效果。
- Dropout是动态去除一些神经元到下一层的连接，相当前于动态L2正则化，通过打压隐层连接参数W实现稀疏性。

## relu函数为分段线性函数，为什么能使网络增加非线性表达？

- 首先什么是线性的网络，如果把线性网络看成一个大的矩阵M。那么输入样本A和B，则会经过同样的线性变换MA, MB（这里A和B经历的线性变换矩阵M是一样的）。
- 的确对于单一的样本A，经过由relu激活函数所构成神经网络，其过程确实可以等价是经过了一个线性变换M1，但是对于样本B，在经过同样的网络时，由于每个神经元是否激活（0或者Wx+b）与样本A经过时情形不同了（不同样本），因此B所经历的线性变换M2并不等于M1。因此，relu构成的神经网络虽然对每个样本都是线性变换，但是不同样本之间经历的线性变换M并不一样，所以整个样本空间在经过relu构成的网络时其实是经历了非线性变换的。
- 还有一种解释就是，不同样本的同一个feature，在通过relu构成的神经网络时，流经的路径不一样（relu激活值为0，则堵塞；激活值为本身，则通过），因此最终的输出空间其实是输入空间的非线性变换得来的。

## Leaky ReLU



Leaky ReLU函数中的参数  $\alpha$  非常小，即负半轴的输入输出的线性斜率小。其优点是通过在负半轴引入非0线性输出，解决了ReLU的神经元dead问题；同时也带来了参数  $\alpha$  需要手工调整的缺点。理论上 Leaky ReLU激活函数效果好，但其优势在实际场景中并未得到证明，因此在具体业务中使用不多。

## 梯度爆炸、梯度消失的原因及解决方法

假设有4层网络，在反向传播时，损失函数对参数的求导

$$\frac{\partial L}{\partial w^1} = \frac{\partial L}{\partial a^4} \frac{\partial a^4}{\partial z^4} \frac{\partial z^4}{\partial a^3} \frac{\partial a^3}{\partial z^3} \frac{\partial z^3}{\partial a^2} \frac{\partial a^2}{\partial z^2} \frac{\partial z^2}{\partial a^1} = \frac{\partial L}{\partial a^4} \cdot f'(z^4) w^3 f'(z^3) w^2 f'(z^2) w^1 f'(z^1) \cdot a^1$$

,

### 1、梯度消失

**原因：** (1) 不合适的损失函数（使用sigmoid或tanh）。当网络层数过深时，多个激活函数相乘导致梯度越来越小；每一层输入值过大时，也会引起梯度饱和导致梯度消失； (3) 参数w初始化不合理，值  $<=1$ 。

**解决方法：**

- (1) 更换激活函数：如Relu/leakyReLU等。
- (2) batch normalization：**BN就是通过对每一层的输出规范为均值和方差一致的方法**，消除了权重参数放大缩小带来的影响，进而解决梯度消失和爆炸的问题，或者可以理解为BN将输出从饱和区拉倒了非饱和区。
- (3) 残差网络：将某一层的输入直接加到后面第N层的输出上
- (4) LSTM的“门（gate）”结构

LSTM全称是长短期记忆网络（long–short term memory networks），LSTM的结构设计可以改善RNN中的梯度消失的问题。LSTM通过它内部的“门”可以在接下来更新的时候“记住”前几次训练的“残留记忆”。

## 2、梯度爆炸

**原因：** (1) 激活函数使用Relu/Leaky Relu，当输入特征值都为正的情况下，参数w初始化成很大的值，值 $>=1$

**解决方法：**

- (1) 梯度截断

设置一个梯度剪切阈值，然后更新梯度的时候，如果梯度超过这个阈值，那么就将其强制限制在这个范围之内。这可以防止梯度爆炸。

- (2) L1、L2正则化：正则项可以限制权重过大

- (3) batch normalization

参考：<https://zhuanlan.zhihu.com/p/72589432>

### \*Batch Normalization

在神经网络中，有这样的一个问题：**Internal Covariate Shift**。

假设第一层的输入数据经过第一层的处理之后，得到第二层的输入数据。这时候，第二层的输入数据相对第一层的数据分布，就会发生改变，所以这一个batch，第二层的参数更新是为了拟合第二层的输入数据的那个分布。然而到了下一个batch，因为第一层的参数也改变了，所以第二层的输入数据的分布相比上一个batch，又不太一样了。然后第二层的参数更新方向也会发生改变。层数越多，这样的问题就越明显。

但是为了保证每一层的分布不变的话，那么如果把每一层输出的数据都归一化0均值，1方差不就好了？但是这样就会完全学习不到输入数据的特征了。不管什么数据都是服从标准正太分布，想想也会觉得有

点奇怪。所以BN就是增加了两个自适应参数，可以通过训练学习的那种参数。这样把每一层的数据都归一化到  $\beta$  均值， $\gamma$  标准差的正态分布上。

【将输入分布变成正态分布，是一种去除数据绝对差异，扩大相对差异的一种行为，所以BN层用在分类上效果好的。对于Image-to-Image这种任务，数据的绝对差异也是非常重要的，所以BN层可能起不到相应的效果。】

## BN应该放在哪里？

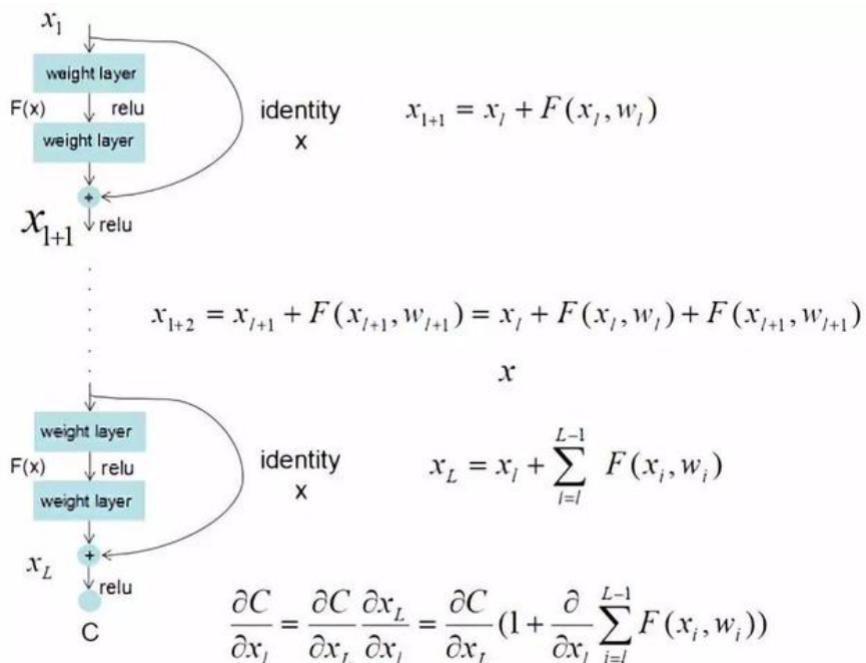
- Sigmoid：如果先BN再Sigmoid，由于BN后方差接近于1，均值接近于0，使得BN后的数据接近于Sigmoid的线性区域，降低了激活函数的非线性能力，这种情况下建议Sigmoid+BN。
- Relu：如果先Relu再BN，Relu后部分神经元已经失活，失活的神经元将对BN的归一化产生影响，这种情况下建议BN+Relu。

## \*残差网络

解决了梯度消失和网络退化的问题。

### 1) 为什么可以解决梯度消失问题？

不管括号内右边部分的求导参数有多小，因为左边的1的存在，并且将原来的链式求导中的连乘变成了连加状态（正确？），都能保证该节点参数更新不会发生梯度消失或梯度爆炸现象。



### 2) 为什么可以解决网络退化问题？

## -> 网络退化问题

举个例子，假设已经有了一个最优化的网络结构，是18层。当我们设计网络结构的时候，我们并不知道具体多少层次的网络时最优化的网络结构，假设设计了34层网络结构。那么多出来的16层其实是冗余的，我们希望训练网络的过程中，模型能够自己将这16层冗余层训练为恒等映射，也就是经过这层时的输入与输出完全一样。但是往往模型很难将这16层恒等映射的参数学习正确，那么就不如最优化的18层网络结构的性能，这就是随着网络深度增加，模型会产生退化现象。它不是由过拟合产生的，而是由冗余的网络层学习了不是恒等映射的参数造成的。

-> 为什么能解决？

我们发现，假设该层是冗余的，在引入ResNet之前，我们想让该层学习到的参数能够满足  $h(x)=x$ ，即输入是x，经过该冗余层后，输出仍然为x。但是可以看见，要想学习  $h(x)=x$  恒等映射时的这层参数时比较困难的。ResNet想到避免去学习该层恒等映射的参数，使用了如上图的结构，让 $h(x)=\text{Relu}(F(x)+x)$ ；这里的F(x)我们称作残差项，我们发现，要想让该冗余层能够恒等映射，我们只需要学习 $F(x)=0$ 。学习  $F(x)=0$  比学习  $h(x)=x$  要简单，因为一般每层网络中的参数初始化偏向于0，并且ReLU能够将负数激活为0，这样在相比于更新该网络层的参数来学习 $h(x)=x$ ，该冗余层学习  $F(x)=0$  的更新参数能够更快收敛。

参考：[https://blog.csdn.net/weixin\\_42253689/article/details/110450298](https://blog.csdn.net/weixin_42253689/article/details/110450298)

## 10. AUC/Precision/Recall/F1-score计算

1、查准率：Precision = TP/(TP+FP)

2、查全率：Recall = TP/(TP+FN)

3、F1-score = 2PR/(P+R)

-> F1-score表示的是precision和recall的调和平均评估指标

F-Measure：

$$F_{\beta} = \frac{(\beta^2 + 1) PR}{\beta^2 \cdot P + R}$$

## 4、ROC曲线

根据学习器预测的结果对样本从小到大进行排序，然后依次设定阈值（将每个样本的打分作为阈值对样本划分），每一次算出一个真阳率和假阳率，以假阳率为横坐标，以真阳率为纵坐标，打点画线。

-- ROC曲线越靠近左上角，模型的准确性就越高。最靠近左上角的ROC曲线上的点是分类错误最少的最好阈值。

- 真阳率 $TPR = TP/(TP+FN)$ , 代表分类器预测的**正类**中实际正实例占所有正实例的比例。
  - 假阳率 $FPR = FP/(FP+TN)$ , 代表分类器预测的**正类**中实际负实例占所有负实例的比例。

## 5、AUC

**含义：**Roc曲线下的面积，介于0.1和1之间。

从计算概率的角度理解AUC可以理解为：随机抽出一对样本（一个正样本，一个负样本），然后用训练得到的分类器来对这两个样本进行预测，预测得到正样本的概率大于负样本概率的概率。

->AUC值越大，当前分类算法越有可能将正样本排在负样本前面，从而能够更好地分类。

计算

首先按照预测的概率从小到高排列：

class	label	pre
A	0	0.1
C	1	0.3
B	0	0.4
D	1	0.8

显然比C小的就是A

比D小的就是A, B

C的位置是2 D的位置是4

$$2+4 = A+C+A+B+C+D \quad \dots \dots \dots (1)$$

而我们的目标是：

$$1+2=A+A+B \quad \dots \dots \dots \quad (2)$$

因此需要将(1)式减去 C+C+D

显然就是将正样本进行排列然后用等差数列求和公式即：

$$\frac{M(M+1)}{2}$$

所以最后的求解公式为：

$$AUC = \frac{\sum_{i \text{ is positiveClass}} rank_i - \frac{M(1+M)}{2}}{M \times N} = \frac{(4+2) - \frac{2 \times (2+1)}{2}}{2 \times 2} = 0.75$$

## 11. Focal Loss损失函数

它使容易分类的样本权重降低，而对难分类的样本权重增加。

数学定义： Focal loss 调变因子（modulating factor）乘以原来的交叉熵损失。

公式为：  $FL(p_t) = -(1 - p_t)^\gamma \log(p_t)$

$(1-p_t)^\gamma$  为调变因子，这里  $\gamma \geq 0$ ，称为聚焦参数。

从上述定义中可以提取出Focal Loss的两个性质：

- 1) 当样本分类错误时， $p_t$ 趋近于0，调变因子趋近于1，使得损失函数几乎不受影响。另一方面，如果示例被正确分类， $p_t$ 将趋近于1，调变因子将趋近于0，使得损耗非常接近于0，从而降低了该特定示例的权重。
- 2) 聚焦参数 ( $\gamma$ ) 平滑地调整易于分类的示例向下加权的速率。

### FL(Focal Loss)和CE（交叉熵损失）的比较

当  $\gamma=2$  时，与概率为0.9的示例相比，概率为0.9的示例的损失比CE低100倍，损失将降低1000倍。

顶部的图描述了不同  $\gamma$  值下的 FL。当  $\gamma=0$  时，FL 等于 CE 损耗。这里我们可以看到，对于  $\gamma=0$  (CE 损失)，即使是容易分类的例子也会产生非平凡的损失震级。这些求和的损失可以压倒稀有类 (很难分类的类)。

## 12. loss不收敛

表现：

- 1) loss震荡；-- 激活函数不合理；学习率设置过大
  - 选用sigmoid作为激活函数，会出现梯度zigzag的现象

- 选用relu函数作为激活函数的同时使用了softmax或者带有exp的函数做分类层的loss函数，造成梯度饱和，梯度回传消失，所有的weight会变成NAN，从此之后weight就会一直保持NAN，所以loss就飞起来了；
- 选用relu函数作为激活函数的同时学习率设置过大，导致参数更新变化过大变成负值，神经元进入dead状态，引起梯度消失；

2) loss训练小幅度下降未达理想水平，在验证集上基本不变；-- 过拟合

## 二、潜在竞对人群识别

### 图神经网络和知识图谱的区别

-- ipad笔记

### 信息增益比指标

- 对连续特征计算信息增益的方法：

将连续的特征进行离散化，然后再计算信息增益。-- 常用的离散化策略是二分法。

具体方法：<https://blog.csdn.net/u012328159/article/details/79396893>

### 一致性正则化损失

基于平滑假设和聚类假设，具有不同标签的数据点在低密度区域分离，并且相似的数据点具有相似的输出。那么，如果对一个未标记的数据应用实际的扰动，其预测结果不应该发生显著变化，也就是输出具有一致性。

用于半监督学习，通过在未标记数据上构造添加扰动后的预测结果  $\tilde{y}$  与正常预测结果  $y$  之间的无监督正则化损失项，提高模型的泛化能力。

数学化表达如下：

$$D[p_{\text{model}}(y|\text{Augment}(x), \theta), p_{\text{model}}(y|\text{Augment}(x), \theta)]$$

其中， $D$  为度量函数，一般采用 KL 散度或 JS 散度，当然也可使用交叉熵或平方误差等。 $\text{Augment}(\cdot)$  是数据增强函数，会添加一些噪声扰动， $\theta$  为模型参数，多个模型的参数可以共用，或者通过一些转换建立联系(Mean-Teacher)，如 EMA，也可以相互独立(Dual-Student)。

常用的有度量函数：

(1) **熵正则化/熵最小化**：常用于伪标签的半监督模型中，加入熵正则化项是为了得到更高置信的伪标签。

计算的是无标签数据的预测概率的熵， $M$  为预测类别， $n-(l+1)+1$  为无标签样本数

-- 衡量的是模型对样本分类的区分程度，区分越好，熵越小。

$$\lambda \sum_{i=l+1}^n \sum_{m=1}^M P(m|x_i; \theta) \ln P(m|x_i; \theta)$$

<https://blog.csdn.net/u012420553/article/details/100997590>

(2) **均方误差**：约束数据增强前后的值差异

(3) **KL散度**：约束数据增强前后的概率分布

-- 有论文实验表明，MSE 的表现优于 KL 散度。相对于 KL 散度，MSE 的计算方式使得模型对于无标签数据的预测错误，敏感度更低。

-- 论文建议，MSE 和 KL 散度各有优劣，二者的选取与数据集的实际分布特征关系很大，在实践中不妨进行对比测试。

## 锐化操作

类似softmax的作用，为了突出显示预测值大的类别。对于S平均之后的结果再进行锐化操作，使得各类别上的概率值差别更为明显。

计算未标记数据的标签分布：

1. 未标记数据的总预测值：对  $S$  个数据增强的预测结果取均值

$$\bar{\mathbf{Z}}_i = \frac{1}{S} \sum_{s=1}^S \tilde{\mathbf{Z}}_i^{(s)}$$

2. 对于第  $i$  个未标记节点的第  $j$  类 label 的真实分布概率应该为： (softmax计算：指数归一化)

$$\overline{\mathbf{Z}}'_{ij} = \overline{\mathbf{Z}}_{ij}^{\frac{1}{T}} \Bigg/ \sum_{c=0}^{C-1} \overline{\mathbf{Z}}_{ic}^{\frac{1}{T}}, (0 \leq j \leq C-1)$$

其中  $0 < T \leq 1$  充当“温度”，用于控制分类分布的锐度。当  $T$  趋近于 0 时，锐化的标签分布将接近一个 one-hot 分布。【 $T$  越小，越锐利】

## transformer 的多头注意力机制

-- ipad 笔记

## 基于元关系的异质 attention

以<起始节点类型，边类型，终止节点类型>为一组元关系

## 节点相似性度量

节点相似性分析的一个典型应用就是链路预测，它是指如何通过已知的各种信息预测给定网络中尚不存在连边的两个节点之间产生连接的可能性。基于节点相似性进行链路预测的基本假设就是如果两个节点之间的相似性越大，它们之间存在连接的可能性就越大。

## 基于共同邻居数的指标

### 共同邻居相似度 (Common Neighbours Similarity)

对于两个顶点  $x, y \in V$  而言，如果它们的共同邻居数越多，表示它们的相似度越高，反之，相似度越低。

$$CN(x, y) = |N(x) \cap N(y)| = \sum_{u \in N(x) \cap N(y)} 1.$$

### Jaccard 相似度 (Jaccard Similarity)

如果将两个节点  $x$  和  $y$  的邻居分别作为两个集合  $N(x), N(y)$ ，

$J(x, y) = CN(x, y) / TN(x, y)$  就可以作为顶点  $x$  和  $y$  的 Jaccard 相似度指标，其相似度是通过邻居来衡量的。

Jaccard 指标：两个顶点共同邻居数比上他们的所有邻居数之和。

## 基于网络结构信息定义的指标

考虑了共同邻居的度信息。

## 好友度量 (Friend Measure)

$$\text{Friend-measure}(x, y) = \sum_{u \in N(x)} \sum_{v \in N(y)} \delta(u, v),$$

其中  $\delta$  用于判断  $u, v$  之间是否有边相连接。如果相连接，则取值为 1，否则取值为 0。

好友之间联系紧密，说明两节点之间越相似。

## Adamic/Adar度量

$$A(x, y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{\ln |N(u)|},$$

度小的共同邻居节点的贡献大于度大的共同邻居节点的。

根据共同邻居的节点的度给每个节点赋予一个权重值，即为邻居节点的度的对数分之一。如果  $x, y$  的共同邻居  $u$  拥有较多的邻居，则降低权重，否则增加权重。

## 基于全局信息的指标

(1) 局部路径指标。它在共同邻居指标的基础上考虑了三阶邻居的贡献，定义如下：

$$S = A^2 + \alpha A^3,$$

$A$  为网络的邻接矩阵， $(A^n)_{xy}$  给出了节点  $x$  和  $y$  之间长度为  $n$  的路径数。当  $\alpha=0$  时，LP 指标就等于共同邻居指标。

(2) Katz 指标。它考虑的是所有路径数，且对越短的路径赋予越大的权重，定义为：

$$s_{xy} = \sum_{l=1}^{\infty} \beta^l (A^l)_{xy},$$

其中  $\beta$  为权重衰减因子。对应的相似性矩阵如下：

$$S = \beta A + \beta^2 A^2 + \beta^3 A^3 + \dots = (\mathbf{I} - \beta A)^{-1} - \mathbf{I}.$$

为了保证数列的收敛性， $\beta$  的取值必须小于邻接矩阵  $A$  最大特征值的倒数。

## ROI计算

简单理解：投资回报率 = 利润/成本 = (收入-成本) / 成本

$$\text{余额} ROI = \frac{\left( (\text{人均余额增量}) * (0.144 - 0.08) * \text{留存时长}/360 \right) - \text{预估总成本}}{\text{预估总成本}}$$

ROI = (贷款金额\*利率\*时间/360-成本) /成本

## 图聚类系数

$$c(i) = \frac{|triangles(i)|}{|triangles(i)| + |triples(i)|} = \frac{1}{1 + \frac{|triples(i)|}{|triangles(i)|}}$$

## 半监督学习

<https://yuque.antfin-inc.com/qvpsyf/hf2imr/ric66f?singleDoc#> 《半监督模型调研》

## 图数据增强方法

- 结构变换：移除边、增加边、翻转边、删除点、由random walks产生的子图、个性化PageRank扩散；
- 特征变换：特征masking、特征dropout

## 图的半监督学习和传统半监督学习的差别

主要在于数据增强的方法上，

1. 图数据样本之间存在相互依赖，节点的改变会影响该节点以及它的邻域，甚至它邻域的邻域。结构数据样本是相互独立的。

-- 图数据需要并行数据增强，不能直接在原图上进行数据增强

2. 多层次数据增强，可作用在节点特征上，也可作用在边关系上。结构数据只作用在节点特征上。
3. 图数据是不规则的，邻居节点之间没有顺序，因此不存在翻转、平移等操作

因此，我们无法直接将图像/NLP领域的数据增强技术直接应用于图数据。

## 图神经网络

<https://yuque.antfin-inc.com/qvpsyf/hf2imr/ric66f?singleDoc#> 《半监督模型调研》

## 1.1 频域方法和空域方法

**频域方法：**图的频域卷积是在傅里叶空间完成的，我们对图的拉普拉斯矩阵进行特征值分解，特征分解更有助于我们理解图的底层特征，能够更好的找到图中的簇或者子图，典型的频域方法有ChebNet, GCN等。但是图的特征值分解是一个特别耗时的操作，具有 $O(n^3)$ 的复杂度，很难扩展到海量节点的场景中。

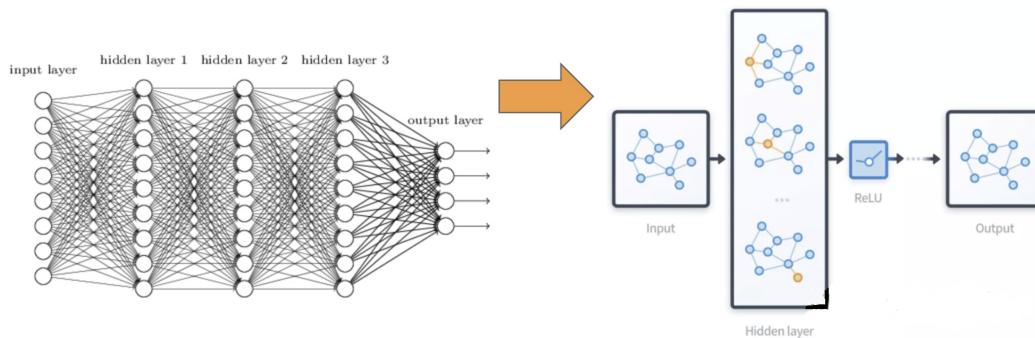
**空域方法：**空间方法作用于节点的邻居节点，使用 $K$ 个邻居节点来计算当前节点的属性。基于空域的方法的计算量要比频域方法小很多，时间复杂度约为 $O(|E|d)$ ，本文要介绍的GraphSAGE就是经典的基于空域的图模型。

## GCN

相比较于神经网络最基本的网络结构全连接层（MLP），特征矩阵乘以权重矩阵，图神经网络多了一个邻接矩阵。

$$H = \sigma(XW)$$

$$H = \sigma(AXW)$$



在实际计算中，加入对自身节点的考虑并对A进行归一化处理，所以将A改写为（拉普拉斯矩阵）：

$$\hat{A} = \bar{D}^{-1/2}(A + I)\bar{D}^{-1/2}$$

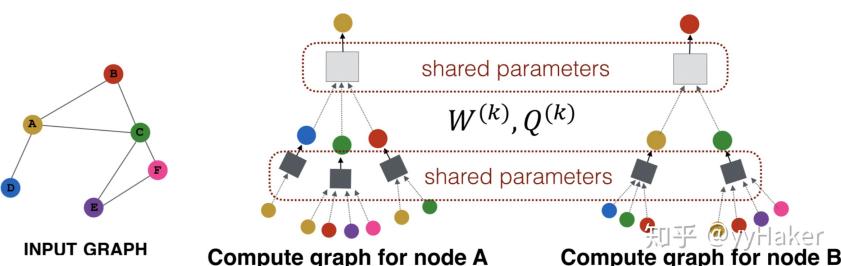
$D$  为顶点的度矩阵（对角矩阵），对角线上元素依次为各个顶点的度。

GCN的核心（AGG）：

$$H = \sigma(\hat{A}XW)$$

假设我们构造一个两层的GCN，激活函数分别采用ReLU和Softmax，最后得到的结果为

$$Z = f(X, A) = softmax(\hat{A} \text{ReLU}(\hat{A}XW^{(0)})W^{(1)})$$



通过若干层GCN每个node的特征从X变成了Z，但是，无论中间有多少层，node之间的连接关系，即A，都是共享的。邻接矩阵可以看成是一种掩码（MASK），每一行值为0的位置表示其对当前节点的聚合特征没有影响，值为1的位置表示需要聚合的节点。邻接矩阵反映了整个图的连接情况，即图神经网络的感知域（对应CNN中的感知域概念，这也是为什么GCN叫图卷积网络的原因）。

$\hat{A} * W$  相当于卷积核的作用。

模型可学习的参数是  $W^{(l)}$ ， $H^{(l)}W^{(l)}$  进行了线性变换，对于空间聚合后的特征进行了feature augment。矩阵  $W^{(l)}$  的两个维度分别是：矩阵  $H^{(l)}$  的第二个维度 以及 根据特征增强需要设计的维度（是超参数）。很显然，这个矩阵维度与顶点数目或者每个顶点的度无关，也就是说模型在每一层共享了用于特征增强的参数矩阵。

为什么GCN要用拉普拉斯矩阵？  $L^{sys} = D^{-1/2}LD^{-1/2}$

- (1) 拉普拉斯矩阵是对称矩阵，可以进行特征分解（谱分解），这就和GCN的spectral domain对应上了  
-> 方便矩阵求解
- (2) 拉普拉斯矩阵只在中心顶点和一阶相连的顶点上（1-hop neighbor）有非0元素，其余之处均为0 ->  
这正是邻接矩阵的含义
- (3) 通过拉普拉斯算子与拉普拉斯矩阵进行类比

CNN中有两大核心思想：网络局部连接，卷积核参数共享，对应GCN如何实现？

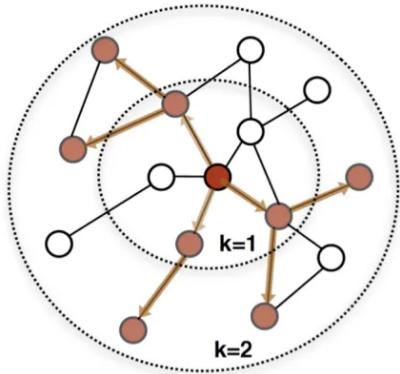
- 网络局部连接 -- 由拉普拉斯矩阵第 (2) 点特性实现
- 卷积核参数共享 -- 由W实现

## GraphSage

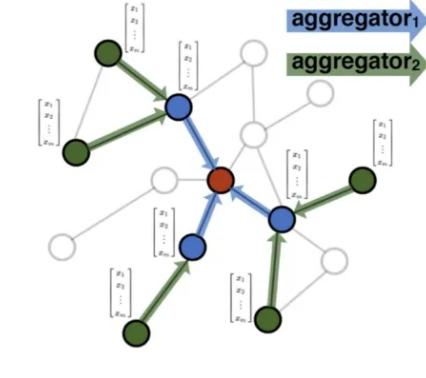
GraphSAGE的核心：GraphSAGE不是试图学习一个图上所有node的embedding，而是学习一组 aggregator聚合函数，为每个node产生embedding的映射

GraphSAGE提出了一个解决方案。它的流程大致分为3步：

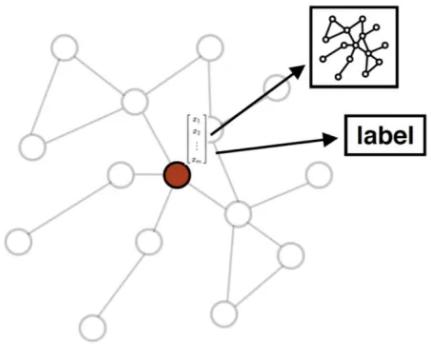
1. 对邻居进行随机采样，每一跳抽样的邻居数不多于  $S_k$  个，如第一跳采集了3个邻居，第二跳采集了5个邻居；
2. 生成目标节点的embedding：先聚合二跳邻居的特征，生成一跳邻居的embedding，再聚合一跳的embedding，生成目标节点的embedding；
3. 将目标节点的embedding输入全连接网络得到目标节点的预测值。



1. Sample neighborhood



2. Aggregate feature information from neighbors



3. Predict graph context and label using aggregated information

GraphSAGE的思想就是不断的聚合邻居信息，然后进行迭代更新。随着迭代次数的增加，每个节点的聚合的信息几乎都是全局的，这点和CNN中感受野的思想类似。

【注意】采样时从内到外（节点阶数从K->0），而聚合是从外到内的（节点阶数从0->K）。

### 为什么GraphSage能对新增节点进行预测？

1. 采用每一层固定个数的邻居节点采样，无论是否新增节点，每一层对应的参数矩阵的维度都是不变的，即参数矩阵维度与实际节点数无关。【第k层参数矩阵维度 = 第k层节点数\*(第k-1层特征维度 || 第k层特征维度)】
2. 每一层共享同一个参数矩阵，学习一个节点的信息是怎么通过其邻居节点的特征聚合而来的。因此，对一个新加入的节点a，只需要知道其自身特征和相邻节点，就可以得到其向量表示。不必重新训练得到其他所有节点的向量表示。

- 聚合函数的选择：

GraphSAGE也对聚合函数的性质进行了分析，并提出了聚合函数需要满足下面的条件：

- 聚合函数需要对聚合节点的数量进行自适应，也就是说无论聚合的节点的数量是多少，进行聚合操作后得到的特征向量的维度必须是相同的。
- 聚合函数需要具有排列不变性，也就是说得到的特征向量的顺序应该与数据的输入顺序无关，即满足  $\text{Aggregate}(v_1, v_2) = \text{Aggregate}(v_2, v_1)$ 。
- 聚合函数必须是可导的。

论文中给出了几个聚合函数：

- 平均聚合：先对邻居的embedding按维度进行平局，再对得到的平均值进行非线性变换：

$$\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W} \cdot \text{MEAN}(\mathbf{h}_v^{k-1} \cup \mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v))) \quad (1)$$

- 池化聚合：先对上一层每个节点的embedding进行非线性变换，然后再对得到的结果进行平均或者最大池化：

$$\text{AGGREGATE}_k^{\text{pool}} = \max(\sigma(\mathbf{W}_{\text{pool}} \mathbf{h}_{u_i}^k + b), \forall u_i \in \mathcal{N}(v)) \quad (2)$$

- LSTM：LSTM并不满足排列不变性，并且计算量非常大，但是由于LSTM拥有非常大的容量，因此有时候也被考虑用作聚合函数。

从上面的采样和聚合过程我们可以看出，GraphSAGE算法的计算过程仅和当前节点的\$k\$阶邻居节点相关，而不需要考虑图的全局信息，这也使得GraphSAGE具有了归纳学习的能力。

参考：<https://www.cnblogs.com/ChenKe-cheng/p/11872916.html>

代码详解：<https://zhuanlan.zhihu.com/p/336195862>

## 异质图神经网络

HAN、HetGNN、HeGNN

HGT

[https://blog.csdn.net/qq\\_44015059/article/details/109099051](https://blog.csdn.net/qq_44015059/article/details/109099051)

现有的GNN方法的不足：（以HAN, GTNs, HetGNN等方法为例）

- (1) 大多数方法需要为异质图设计元路径 (GTNs除外)；
- (2) 现有的方法要么假定不同类别的节点/边共享相同的特征和表示空间，要么就是单独为某一类型的节点或边设计不同的不可共享的参数。这样的话不能充分捕获异质图的属性信息；
- (3) 大多数方法都没有考虑异质图的动态时间特征；

(4) 不能建模大规模 (Web-scale) 异质图。

创新点：

(1) 为了解决图的异质性问题，引入节点类型和边类型有依赖的注意力机制。使用元关系参数化权重矩阵，用于计算每条边的注意力系数。

因此，不同类型的节点和边就可以维护其特定的表示空间。同时，通过不同的边相连的节点仍然可以交互、传递、聚合信息，并且不受它们之间分布差异的限制。HGT可以通过跨层的信息传递来合并不同类型的高阶邻居的信息，这可以看作是“软”元路径。

(2) 为了处理动态图，在HGT中引入相对时间编码 (RTE) 技术。

并不是将输入图按不同的时间戳分片处理，而是将在不同时间出现的所有边看成一个整体。设计RTE策略，建模任意时间长度的结构性的时间依赖关系，甚至包含不可见的和未来的时间戳。通过端到端的训练，RTE使得HGT自动学习到异质图的时序依赖关系以及异质图随时间的演化。

(3) 为了处理Web-scale的图数据，作者设计了异构小批图采样算法

直接对异构图使用同构图类型数据的采样方法，容易得到对于不同节点类型非常不平衡的子图，因为每种类型的度分布和节点总数可能会有很大差异。

优点：

- 保持每种类型的节点和边的数量相近
- 保持采样子图的密集性最小化信息损失，降低样本方差。

具体实现：

1. 为每个节点类型  $T$  保留一个单独的节点预算  $B[T]$ ，保存节点采样概率。：已知节点  $t$  已经采样，将其所有的直接邻居加到相应的预算中，并对这些邻居加上  $t$  的归一化度，用于计算采样概率。这种归一化相当于将每个采样节点的随机游走概率累加到其邻域，避免采样被高度节点所控制。直观地看，该值越高，则候选节点与当前采样节点的关联程度越高，因此应该给予被采样的概率越高。

2. 更新预算后，计算抽样概率  $\text{prob}^{(l-1)}[\tau][s] \leftarrow \frac{\hat{B}[\tau][s]^2}{\|\hat{B}[\tau]\|_2^2}$  (节点  $s$  的累计归一化度的平方/ $s$  对应类别的概率) -- 利用这种抽样概率可以降低抽样方差。然后，使用计算出的概率对  $n$  个类型的节点进行抽样，将它们添加到输出节点集，更新其邻域到预算，并将其从预算中移除。重复这个过程  $L$  次，得到一个从初始节点开始的深度为  $L$  的采样子图。最后，重构了采样节点间的邻接矩阵。

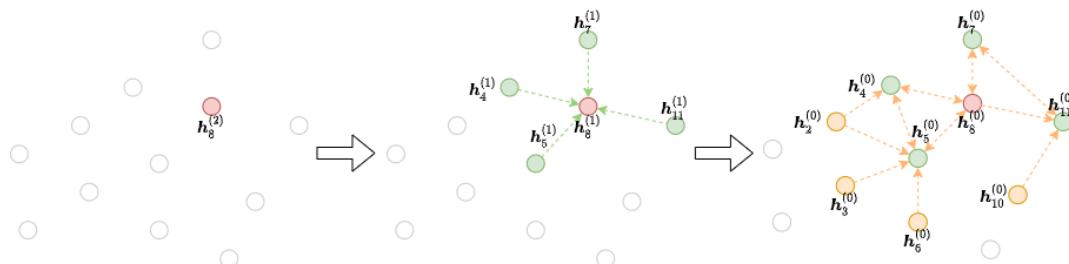
## 图谱构成

	节点数		边数	用户平均度数	APP平均度数	
用户	97,085,571		400,170,863	4	-	
APP	151,782		1,373,801,466	14	-	
			转账关系	101,503,378	10	-
			用户-APP关系	4,921,086,886	51	32,422
				=边数/用户节点		=边数/APP节点

有转账关系的用户约1kw

## 在大图上进行随机小批次训练的方法 —— 邻居采样方法

每次梯度下降，选择一个小批次的图节点，其最终表示将在神经网络的第  $L$  层进行计算，然后在网络的第  $L-1$  层选择该批次节点的全部或部分邻居节点。重复这个过程，直到到达输入层。这个迭代过程会构建计算的依赖关系图，从输出开始，一直到输入。



该方法能节省在大图上训练图神经网络的开销和计算资源。

## 树模型

### GBDT

决策树分为两大类，分类树和回归树。

分类树用于分类标签值，如晴天/阴天/雾/雨、用户性别、网页是否是垃圾页面；

回归树用于预测实数值，如明天的温度、用户的年龄、网页的相关程度；

两者的区别：

- 分类树的结果不能进行加减运算，晴天+晴天没有实际意义；
- 回归树的结果是预测一个数值，可以进行加减运算，例如 20岁-3岁=23岁。
- GBDT的核心在于累加所有树的结果作为最终结果，因此GBDT 中的决策树是CART回归树，预测结果是一个数值。在点击率预测方面常用 GBDT，例如用户点击某个内容的概率。

## 思考

## 1) GBDT常用损失函数有哪些?

- 回归问题常用损失 (MAE、MSE、RMSE) 、还有组合损失如Huber Loss (MAE和MSE结合)  
-- 组合损失的好处 (1) 在0附近可导 (2) loss较大时为MAE, 较小时为MSE, 降低梯度能够逐步靠近最优点

- 分类问题损失: 交叉熵、指数损失函数 $\exp(-y \cdot f(x))$   $y$ 为label{-1, 1};  $f(x)$ 为预测值

## 2) GBDT 分类任务和回归任务的区别

做分类任务和回归任务相似, 所用的损失函数不同

## 3) 为什么GBDT不适合使用高维稀疏特征?

- 高维稀疏特征导致子树数量多, 每次分裂只能筛选少量样本, 训练阶段为串行结构速度较慢, 使得树模型训练变得低效;
- 高维稀疏特征容易造成过拟合, 模型泛化性弱

## 4) GBDT的优缺点

优点

- 预测阶段, 因为每棵树的结构都已确定, 可并行化计算, 计算速度快。
- 适用稠密数据, 泛化能力和表达能力都不错, 数据科学竞赛榜首常见模型。
- 可解释性不错, 鲁棒性亦可, 能够自动发现特征间的高阶关系, 不需要对数据做过多预处理。

缺点

- GBDT 在高维稀疏的数据集上, 效率较差, 且效果表现不如 SVM 或神经网络。
- 适合数值型特征, 在 NLP 或文本特征上表现弱。
- 训练过程无法并行, 工程加速只能体现在单颗树构建过程中。

## 5) 为什么GBDT使用负梯度来近似拟合残差?

- 负梯度是更加广义上的拟合项, **扩展到更复杂的损失函数中**。直接拟合残差只是损失函数为平方损失的特殊情况。
- 损失函数除了loss还有正则项, 正则中有参数和变量, 很多情况下只拟合残差loss变小但是正则变大, 损失函数不一定就小, 这时候就要用梯度了, 梯度的本质也是一种方向导数, 综合了各个方向(参数)的变化, 选择了一个总是最优(下降最快)的方向;

## 6) Shrinkage

#### 四、Shrinkage

Shrinkage（缩减）的思想认为，每次走一小步逐渐逼近结果的效果，要比每次迈一大步很快逼近结果的方式更容易避免过拟合。即它不完全信任每一个棵残差树，它认为每棵树只学到了真理的一小部分，累加的时候只累加一小部分，通过多学几棵树弥补不足。用方程来看更清晰，即

没用Shrinkage时： ( $y_i$ 表示第*i*棵树上 $y$ 的预测值，  $y(1 \sim i)$ 表示前*i*棵树 $y$ 的综合预测值)

$$y(i+1) = \text{残差}(y_1 \sim y_i), \text{ 其中: 残差}(y_1 \sim y_i) = y_{\text{真实值}} - y(1 \sim i)$$

$$y(1 \sim i) = \text{SUM}(y_1, \dots, y_i)$$

Shrinkage不改变第一个方程，只把第二个方程改为：

$$y(1 \sim i) = y(1 \sim i-1) + \text{step} * y_i$$

即Shrinkage仍然以残差作为学习目标，但对于残差学习出来的结果，只累加一小部分 (step\*残差) 逐步逼近目标，step一般都比较小，如0.01~0.001（注意该step非gradient的step），导致各个树的残差是渐变的而不是陡变的。直觉上这也很好理解，不像直接用残差一步修复误差，而是只修复一点点，其实就是把大步切成了很多小步。本质上，Shrinkage为每棵树设置了一个**weight**，累加时要乘以这个**weight**，但和**Gradient**并没有关系。这个**weight**就是step。就像Adaboost一样，Shrinkage能减少过拟合发生也是经验证明的，目前还没有看到从理论的证明。

### 7) 梯度提升 vs 梯度下降

下面我们来对比一下「梯度提升」与「梯度下降」。这两种迭代优化算法，都是在每1轮迭代中，利用损失函数负梯度方向的信息，更新当前模型，只不过：

- 梯度下降中，模型是以参数化形式表示，从而模型的更新等价于参数的更新。

$$w_t = w_{t-1} - \rho_t \nabla_w L|_{w=w_{t-1}}$$

$$L = \sum_i l(y_i, f_w(w_i))$$

- 梯度提升中，模型并不需要进行参数化表示，而是直接定义在函数空间中，从而大大扩展了可以使用的模型种类。

$$F = F_{t-1} - \rho_t \nabla_F L|_{F=F_{t-1}}$$

$$L = \sum_i l(y_i, F(x_i))$$

每次迭代训练一个弱学习器来拟合残差（损失函数负梯度值），最后将多个弱学习器相加得到最终的模型输出。

梯度提升和梯度下降的关系：

梯度提升的目的也是为了梯度下降，但是梯度下降是通过直接求导的方式就可以对参数进行更新求解，而梯度提升是通过累计弱学习器的形式来进行梯度下降。

### 8) GBDT和随机森林的区别

相同点

- 都是集成模型，由多棵树组构成，最终的结果都是由多棵树一起决定。
- 训练完成后可以给出特征重要性指标

不同点

- 弱学习器：组成RF的树可以是分类树，也可以是回归树；而GBDT只能由回归树组成。

- **训练过程**: 随机森林的树可以并行生成，而 GBDT 只能串行生成。
- **输出结果**: 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。
- **异常值**: 随机森林对异常值不敏感，而 GBDT 对异常值比较敏感。
- **学习误差**: 随机森林降低模型的方差，而 GBDT 是降低模型的偏差。

<https://www.showmeai.tech/article-detail/193>

<https://fengxc.me/GBDT%E8%AF%A6%E8%A7%A3.html>

## XGBoost

XGBoost为什么只用到二阶泰勒展开，不用三阶甚至四阶展开？

$$g_i = \partial_{\hat{y}_i}^{\wedge(t-1)} l(\hat{y}_i, y_i) \quad (10)$$

$$h_i = \partial_{\hat{y}_i}^2 l(\hat{y}_i, y_i) \quad (11)$$

如果损失函数定义为平方损失函数的话，即

$$l(\hat{y}_i, y_i) = \frac{1}{2} (\hat{y}_i - y_i)^2 \quad (12)$$

则：

$$\begin{aligned} g_i &= \hat{y}_i - y_i \\ h_i &= 1 \end{aligned} \quad (13)$$

这也是为什么只用到二阶泰勒展开，因为三阶导已经为0。

参考-全：<https://www.showmeai.tech/article-detail/194>

目标函数推导：<https://blog.csdn.net/htbeker/article/details/91517805>

## XGBoost核心

### 1. 目标函数加入正则化项

- 在求解过程中考虑了树结构的复杂度，有效地防止过拟合

### 2. 目标函数使用二阶泰勒展开式展开

- 1) 目标函数公式化简后只与一阶导  $G_i$ 、二阶导  $H_i$  以及树的叶子节点数  $T$  有关，与损失函数的具体形式无关，因此XGBoost支持自定义损失函数；
- 2) 在  $F_{m-1}(\mathbf{x})$  确定了的情况下，对每个样本点  $i$  都可以轻易计算出一个  $g_i$  和  $h_i$ ，每次迭代中都可以并行计算；
- 3) 每次迭代中，学习器以目标函数在当前模型值的负梯度值作为残差近似值进行拟合，以二阶泰勒展开式展开后，负梯度值同时包含了一阶导和二阶导的信息，能够利用更多有效信息，同时加快收敛速度；
- 4) 在节点值求解中也是包含了一阶导和二阶导的信息，同样能够利用更多有效信息，并加快收敛速度；
- 5) 从 1) 得，目标函数只与树结构有关，可用于衡量树结构的好坏，因此用于作为特征最佳切分点的衡量指标，并且可采用并行列块存储的方式实现并行计算（具体原因在7.讲到）；

### 3. 节点分裂准则

一般集成决策树如GBDT子树的基本处理思路和 CART 一样，都采用贪心准则计算，对特征值排序后遍历划分点，将其中最优的分裂收益作为该特征的分裂收益，选取具有最优分裂收益的特征作为当前节点的划分特征，按其最优划分点进行二叉划分，得到左右子树。

基于性能的考量，XGBoost 还对贪心准则做了一个近似版本，简单说，处理方式是「将加权的特征值分位数作为划分候选点」。这样将划分候选点集合由全样本间的遍历缩减到了几个分位数之间的遍历。

特征分位数的选取有 global 和 local 两种可选策略：

- global 在全体样本上的特征值中选取，在根节点分裂之前进行一次即可；
- local 则是在待分裂节点包含的样本特征值上选取，每个节点分裂前都要进行。

特征分位数的计算方法：

在近似算法取分位数时，实际上 XGBoost 会取以二阶导  $h_i$ 为权重的分位数，如下图表示的三分位。

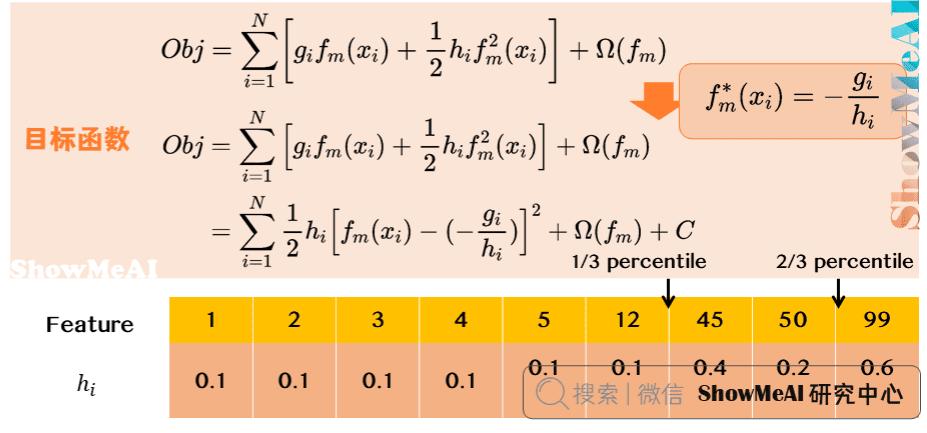


## XGBoost最全解析

### 核心原理归纳解析

#### 节点分裂准则-加权分位数

<http://www.showmeai.tech/>



查看目标函数  $Obj = \sum_{i=1}^N \left[ g_i f_m(x_i) + \frac{1}{2} h_i f_m^2(x_i) \right] + \Omega(f_m)$ , 令偏导为0易得  $f_m^*(x_i) = -\frac{g_i}{h_i}$ , 此

目标函数可理解为以  $h_i$  为权重,  $-\frac{g_i}{h_i}$  为标签的二次损失函数:

$$\begin{aligned} Obj &= \sum_{i=1}^N \left[ g_i f_m(x_i) + \frac{1}{2} h_i f_m^2(x_i) \right] + \Omega(f_m) \\ &= \sum_{i=1}^N \frac{1}{2} h_i \left[ f_m(x_i) - \left( -\frac{g_i}{h_i} \right) \right]^2 + \Omega(f_m) + C \end{aligned}$$

## 4. 处理特征缺失和特征稀疏的问题

XGBoost 也能对缺失值处理, 也对特征稀疏问题 (特征中出现大量的 0 或one-hot encoding结果) 做了一些优化。XGBoost 用「稀疏感知」策略来同时处理这两个问题:

- 简单说, 它的做法是将缺失值和稀疏 0 值等同视作缺失值, 将其「绑定」在一起, 分裂节点的遍历会跳过缺失值的整体, 在此基础上再对非缺失值进行切分遍历。。这样大大提高了运算效率。
- 0 值在XGB中被处理为数值意义上的 0 还是 NA, 需要结合具体平台的设置, 预处理区分开作为数值的 0 (不应该被处理为 NA) 和作为稀疏值的 0 (应该被处理为 NA) 。

## 5. 列采样

- 和随机森林的列采样做法一致, 每次节点分裂并不是用全部特征作为候选集, 而是一个子集。
- 这么做能更好地控制过拟合, 还能减少计算开销。

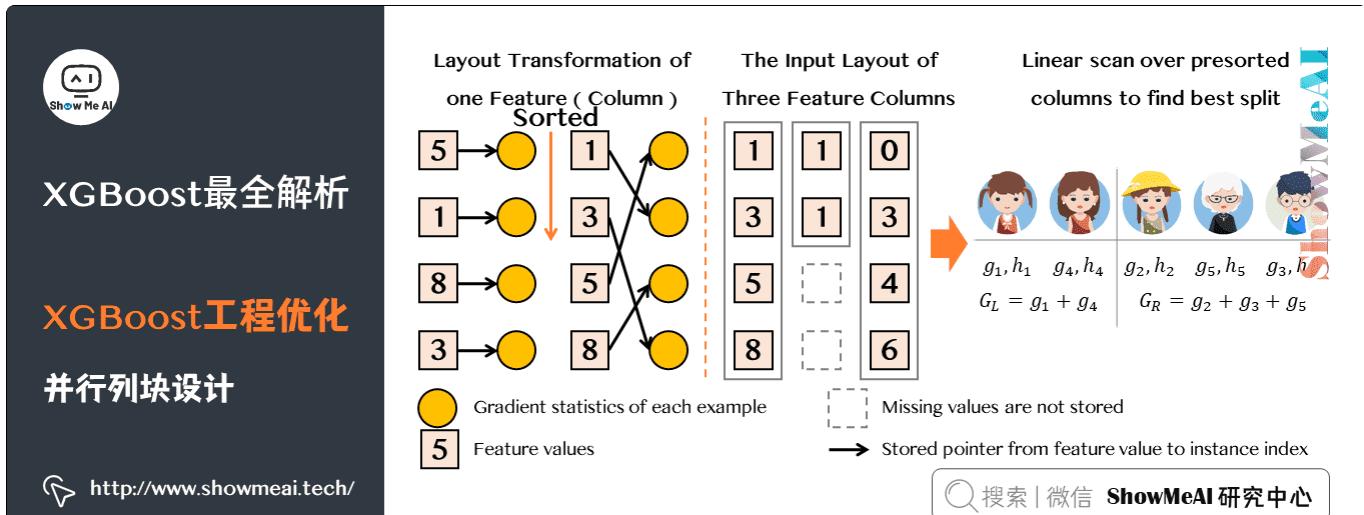
## 6. 学习率

学习率也叫步长、shrinkage, 具体的操作是在每个子模型前 (即每个叶节点的回归值上) 乘上该系数, 不让单颗树太激进地拟合, 留有一定空间, 使迭代更稳定。XGBoost默认设定为 0.3。

这一点GBDT中也有使用

## 7. XGBoost工程优化 —— 并行列块设计

XGBoost 将每一列特征提前进行排序，以块（Block）的形式储存在缓存中，并以索引将特征值和梯度统计量对应起来，每次节点分裂时会重复调用排好序的块。而且不同特征会分布在独立的块中，因此可以进行分布式或多线程的计算。



## XGBoost 防止过拟合的方法

- 在目标函数中加入正则化项
- colsample\_bytree: 每棵树每一级的分裂中，对列数的采样比例
- colsample\_bytree: 和RF一样的列采样，每棵树训练所用到的特征的采样比例
- 学习率 (shrinkage) : 表示每棵树在整个训练中的比重
- subsample: 对样本进行不放回采样，引入样本随机性
- early stop: 当模型在验证集上表现在连续若干次迭代中都不再提升时，终止训练过程

## XGBoost vs GBDT

- GBDT 是机器学习算法，XGBoost 在算法基础上还有一些工程实现方面的优化。
- GBDT 使用的是损失函数一阶导数，相当于函数空间中的梯度下降；XGBoost 还使用了损失函数二阶导数，相当于函数空间中的牛顿法。
- 正则化：XGBoost 显式地加入了正则项来控制模型的复杂度，能有效防止过拟合。
- 列采样：XGBoost 采用了随机森林中的做法，每次节点分裂前进行列随机采样。
- 缺失值：XGBoost 运用稀疏感知策略处理缺失值，GBDT无缺失值处理策略。

- 并行高效：XGBoost 的列块设计能有效支持并行运算，效率更优。

## 随机森林

随机森林是一个用随机方式建立的，包含多个决策树的集成分类器。其输出的类别由各个树投票而定（如果是回归树则取平均）。假设样本总数为n，每个样本的特征数为a，则随机森林的生成过程如下：

1. 从原始样本中采用有放回抽样的方法选取n个样本；
2. 对n个样本选取a个特征中的随机k个，用建立决策树的方法获得最佳分割点；
3. 重复m次，获得m个决策树；
4. 对输入样例进行预测时，每个子树都产生一个结果，采用多数投票机制输出。

随机森林的随机性主要体现在两个方面：

1. 数据集的随机选取：从原始的数据集中采取有放回的抽样（bagging），构造子数据集，子数据集的数据量是和原始数据集相同的。不同子数据集的元素可以重复，同一个子数据集中的元素也可以重复。
2. 待选特征的随机选取：与数据集的随机选取类似，随机森林中的子树的每一个分裂过程并未用到所有的待选特征，而是从所有的待选特征中随机选取一定的特征，之后再在随机选取的特征中选取最优的特征。

以上两个随机性能够使得随机森林中的决策树都能够彼此不同，提升系统的多样性，从而提升分类性能。

随机森林的优点：

1. 可并行计算，训练速度快，泛化能力强，因为训练时树与树之间是相互独立的；
2. 相比单一决策树，能学习到特征之间的相互影响，且不容易过拟合；
3. 能处理高维数据（即特征很多），并且不用做特征选择，因为特征子集是随机选取的；
4. 对于不平衡的数据集，可以平衡误差；
5. 相比SVM，不是很怕特征缺失，因为待选特征也是随机选取；
6. 训练完成后可以给出哪些特征比较重要。