推荐系统脉络 [1]

leolinuxer

July 22, 2020

Contents

I 推荐系统的概念 **2**

1 前深度学习时代 CTR 预估模型的演化之路 **3**

1.1 LR——CTR 模型的核心和基础 4

1.1.1 逻辑回归的数学基础 4

1.1.2 人类的直觉和可解释性 4

1.1.3 工程化的需要 5

1.2 POLY2——特征交叉的开始 5

1.2.1 FM——隐向量特征交叉 5

1.2.2 FFM——引入特征域概念 6

1.3 CTR 模型特征交叉方向的演化 7

1.4 GBDT+LR——特征工程模型化的开端 8

1.5 FTRL——天下武功，唯快不破 9

1.6 LS-PLM——阿里曾经的主流 CTR 模型 10

2 深度学习推荐模型的演化趋势 **12**

3 推荐系统的系统架构 **13**

II 推荐系统的工业化应用 **14**

4 Facebook 的经典 CTR 预估模型 **14**

4.1 用户场景 14

4.2 模型结构 14

4.3 模型的实效性问题和更新策略 15

4.4 facebook 的实时数据流架构 16
4.5 降采样和模型校正 17

5 Facebook 的 DLRM 模型 18
5.1 模型架构 19
5.2 Facebook 的模型并行训练方法 20
5.3 DLRM 模型的效果 21

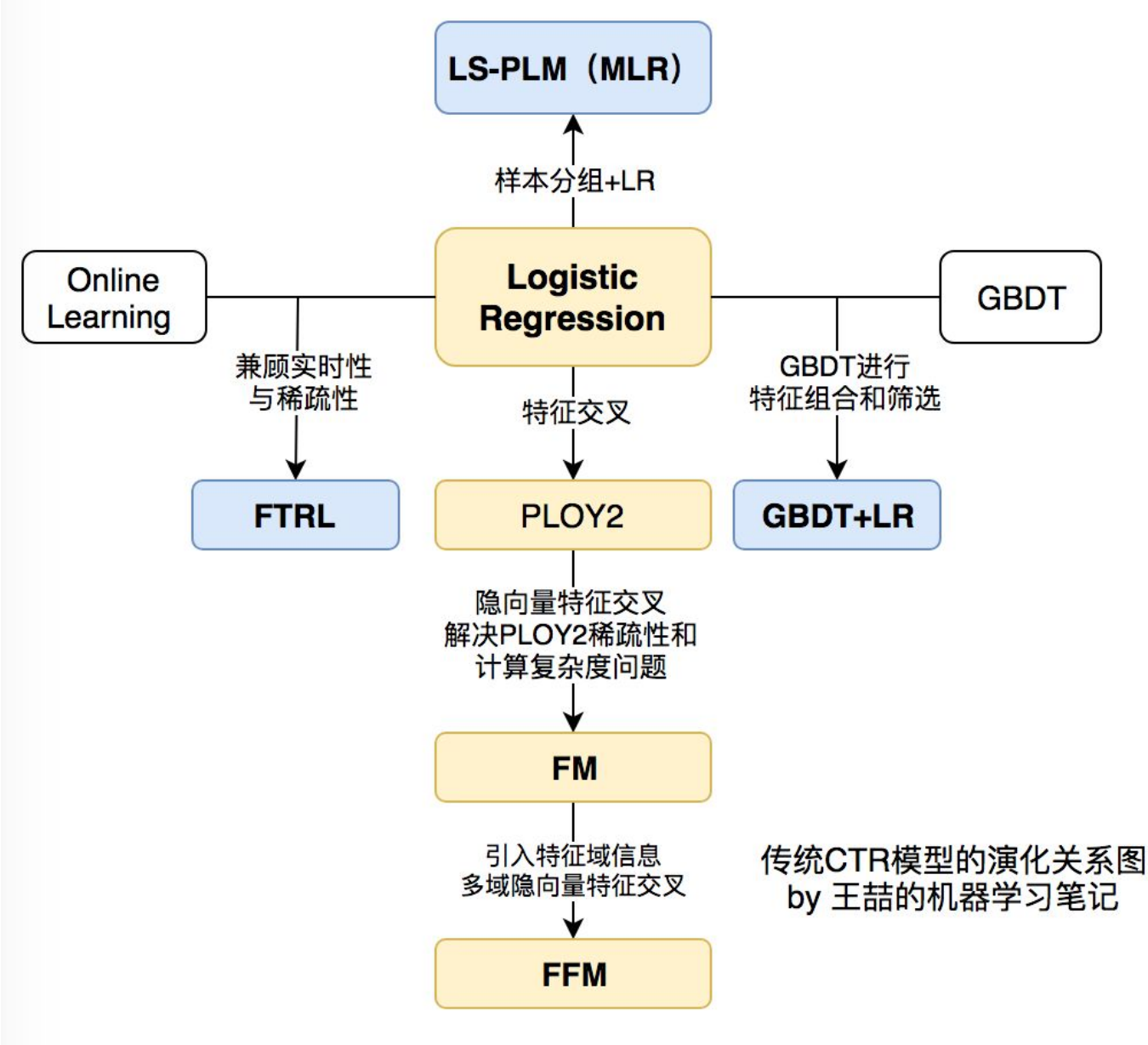
6 Youtube 的深度学习推荐系统 23
6.1 算法基本架构 23
6.2 召回层 24
6.3 排序层 25
6.4 工程实践 26
6.4.1 softmax 的训练 26
6.4.2 最近邻搜索 26
6.4.3 新视频的偏好特征 26
6.4.4 采样训练样本 27
6.4.5 历史的时序特征 27
6.4.6 生成测试集 27
6.4.7 优化目标 28
6.4.8 video embedding 28
6.4.9 特征的特殊处理 28
6.4.10 输出层 28
6.5 详细解读输出层采用 weighted logistic regression 29

7 阿里深度兴趣网络（DIN） 30
7.1 用户场景 30
7.2 注意力机制 31
7.3 论文作者的评论 33

Part I

推荐系统的概念

1 前深度学习时代 CTR 预估模型的演化之路



看到上面的关系图，有经验的同学可能已经对各模型的细节和特点如数家珍了。中间位置的 LR 模型向四个方向的延伸分别代表了传统 CTR 模型演化的四个方向。

- 向下为了解决**特征交叉**的问题，演化出 PLOY2，FM，FFM 等模型；
- 向右为了使用模型化、自动化的手段解决之前**特征工程**的难题，Facebook 将 LR 与 GBDT 进行结合，提出了 GBDT+LR 组合模型；
- 向左 Google 从 **online learning** 的角度解决模型时效性的问题，提出了 FTRL；

- 向上阿里基于**样本分组**的思路增加模型的非线性，提出了 LS-PLM（MLR）模型。

1.1 LR——CTR 模型的核心和基础

位于正中央的是当之无愧的 Logistic Regression。仍记得 2012 年我刚进入计算广告这个行业的时候，各大中小公司的主流 CTR 模型无一例外全都是 LR 模型。LR 模型的流行是有三方面原因的，一是数学形式和含义上的支撑；二是人类的直觉和可解释性的原因；三是工程化的需要。

1.1.1 逻辑回归的数学基础

逻辑回归作为广义线性模型的一种，**它的假设是因变量 y 服从伯努利分布**。那么在点击率预估这个问题上，“点击” 这个事件是否发生就是模型的因变量 y。而用户是否点击广告这个问题是一个经典的掷偏心硬币问题，因此 CTR 模型的因变量显然应该服从伯努利分布。所以采用 LR 作为 CTR 模型是符合 “点击” 这一事件的物理意义的。

与之相比较，线性回归（Linear Regression）作为广义线性模型的另一个特例，其**假设是因变量 y 服从高斯分布**，这明显不是点击这类二分类问题的数学假设。

在了解 LR 的数学理论基础后，其数学形式就不再是空中楼阁了，具体的形式如下：

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

其中 x 是输入向量，θ 是我们要学习的参数向量。结合 CTR 模型的问题来说，x 就是输入的特征向量，$h(x)$ 就是我们最终希望得到的点击率。

1.1.2 人类的直觉和可解释性

直观来讲，LR 模型目标函数的形式就是各特征的加权和，再施以 sigmoid 函数。忽略其数学基础（虽然这是其模型成立的本质支撑），仅靠人类的直觉认知也可以一定程度上得出使用 LR 作为 CTR 模型的合理性。

使用各特征的加权和是为了综合不同特征对 CTR 的影响，而由于不同特征的重要程度不一样，所以为不同特征指定不同的权重来代表不同特征的重要程度。最后要套上 sigmoid 函数，正是希望其值能够映射到 0-1 之间，使其符合 CTR 的物理意义。

LR 如此符合人类的直觉认知显然有其他的好处，就是模型具有极强的可解释性，算法工程师们可以轻易的解释哪些特征比较重要，在 CTR 模型的预测有偏差的时候，也可以轻易找到哪些因素影响了最后的结果，如果你有跟运营、产品一起工作的经验的话，更会知道可解释性强是一个模型多么优秀的 “品质”。

1.1.3 工程化的需要

在互联网公司每天动辄 TB 级别的数据面前，模型的训练开销就异常重要了。在 GPU 尚未流行开来的 2012 年之前，LR 模型也凭借其易于并行化、模型简单、训练开销小等特点占据着工程领域的主流。囿于工程团队的限制，即使其他复杂模型的效果有所提升，在没有明显 beat LR 之前，公司也不会贸然加大计算资源的投入升级 CTR 模型，这是 LR 持续流行的另一重要原因。

1.2 POLY2——特征交叉的开始

但 LR 的表达能力毕竟是非常初级的。由于 LR 仅使用单一特征，无法利用高维信息，在“辛普森悖论”现象的存在下，只用单一特征进行判断，甚至会得出错误的结论。

针对这个问题，当时的算法工程师们经常采用手动组合特征，再通过各种分析手段筛选特征的方法。但这个方法无疑是残忍的，完全不符合“懒惰是程序员的美德”这一金科玉律。更遗憾的是，人类的经验往往有局限性，程序员的时间和精力也无法支撑其找到最优的特征组合。因此采用 PLOY2 模型进行特征的“暴力”组合成为了可行的选择。

$$\phi_{\text{Poly2}}(\mathbf{w}, \mathbf{x}) = \sum_{j_1=1}^{n} \sum_{j_2=j_1+1}^{n} w_h(j_1, j_2) x_{j_1} x_{j_2}$$

在上面 POLY2 二阶部分的目标函数中（上式省略一阶部分和 sigmoid 函数的部分），我们可以看到 POLY2 对所有特征进行了两两交叉，并对所有的特征组合赋予了权重 $w_h(j_1, j_2)$。POLY2 通过暴力组合特征的方式一定程度上解决了特征组合的问题。并且由于本质上仍是线性模型，其训练方法与 LR 并无区别，便于工程上的兼容。

但 POLY2 这一模型同时存在着两个巨大的缺陷：(1) 由于在处理互联网数据时，经常采用 one-hot 的方法处理 id 类数据，致使特征向量极度稀疏，POLY2 进行无选择的特征交叉使原本就非常稀疏的特征向量更加稀疏，使得大部分交叉特征的权重缺乏有效的数据进行训练，无法收敛；(2) 权重参数的数量由 n 直接上升到 n^2，极大增加了训练复杂度。

1.2.1 FM——隐向量特征交叉

为了解决 POLY2 模型的缺陷，2010 年德国康斯坦茨大学的 Steffen Rendle 提出了 FM（Factorization Machine）。

$$\phi_{\text{FM}}(\mathbf{w}, \mathbf{x}) = \sum_{j_1=1}^{n} \sum_{j_2=j_1+1}^{n} (\mathbf{w}_{j_1} \cdot \mathbf{w}_{j_2}) x_{j_1} x_{j_2}$$

从 FM 的目标函数的二阶部分中我们可以看到，相比 POLY2，主要区别是用两个向量的内积 $\mathbf{w}_{j_1} \cdot \mathbf{w}_{j_2}$ 取代了单一的权重 $w_h(j_1, j_2)$。具体来说，FM 为每个特征学习了一个隐权重向量（latent vector），在特征交叉时，使用两个特征隐向量的内积作为交叉特征的权重。

通过引入特征隐向量的方式，直接把原先 n^2 级别的权重数量减低到了 $n * k$（k 为隐向量维度，

$n >> k$)。在训练过程中，又可以通过转换目标函数形式的方法，使 FM 的训练复杂度进一步降低到 $n * k$ 级别。相比 POLY2 极大降低训练开销。

隐向量的引入还使得 FM 比 POLY2 能够更好的解决数据稀疏性的问题。举例来说，我们有两个特征，分别是 channel 和 brand，一个训练样本的 feature 组合是 (ESPN, Adidas)，在 POLY2 中，只有当 ESPN 和 Adidas 同时出现在一个训练样本中时，模型才能学到这个组合特征对应的权重。而在 FM 中，ESPN 的隐向量也可以通过 (ESPN, Gucci) 这个样本学到，Adidas 的隐向量也可以通过 (NBC, Adidas) 学到，这大大降低了模型对于数据稀疏性的要求。甚至对于一个从未出现过的特征组合 (NBC, Gucci)，由于模型之前已经分别学习过 NBC 和 Gucci 的隐向量，FM 也具备了计算该特征组合权重的能力，这是 POLY2 无法实现的。也许 FM 相比 POLY2 丢失了某些信息的记忆能力，但是泛化能力大大提高，这对于互联网的数据特点是非常重要的。

工程方面，FM 同样可以用梯度下降进行学习的特点使其不失实时性和灵活性。相比之后深度学习模型复杂的网络结构，FM 比较容易实现的 inference 过程也使其没有 serving 的难题。因此 FM 在 2012-2014 年前后逐渐成为业界 CTR 模型的重要选择。

1.2.2 FFM——引入特征域概念

2015 年，基于 FM 提出的 FFM（Field-aware Factorization Machine ，简称 FFM）在多项 CTR 预估大赛中一举夺魁，并随后被 Criteo、美团等公司深度应用在 CTR 预估，推荐系统领域。相比 FM 模型，FFM 模型主要引入了 Field-aware 这一概念，使模型的表达能力更强。

$$\phi_{\text{FFM}}(\mathbf{w}, \mathbf{x}) = \sum_{j_1=1}^{n} \sum_{j_2=j_1+1}^{n} (\mathbf{w}_{j_1,f_2} \cdot \mathbf{w}_{j_2,f_1}) x_{j_1} x_{j_2}$$

上式是 FFM 的目标函数的二阶部分。其与 FM 目标函数的区别就在于隐向量由原来的 w_{j_1} 变成了 w_{j_1,f_2} ，这就意味着每个特征对应的不是一个隐向量，而是对应着不同域的一组隐向量，当 w_{j_1} 特征与 w_{j_2} 特征进行交叉时，x_{j_1} 特征会从一组隐向量中挑出与特征 2_{j_2} 的域 f_2 对应的隐向量 w_{j_1,f_2} 进行交叉。同理特征 x_{j_2} 也会用与 x_{j_1} 的域 f_1 对应的隐向量进行交叉。

这里再次强调一下，上面所说的“域”就代表着特征域，域内的特征一般会采用 one-hot 编码形成 one-hot 特征向量。

FFM 模型学习每个特征在 f 个域上的 k 维隐向量，交叉特征的权重由特征在对方特征域上的隐向量内积得到，权重数量共 $n * k * f$ 个。在训练方面，由于 FFM 的二次项并不能够像 FM 那样简化，因此其复杂度为 kn^2。

相比 FM，FFM 由于引入了 field 这一概念，为模型引入了更多有价值信息，使模型表达能力更强，但与此同时，FFM 的计算复杂度上升到 kn^2，远远大于 FM 的 $k * n$。

> 理解：比较 Poly2、FM、FFM
>
> Poly2：对所有特征进行两两交叉，为每种特征组合都学习一个权重；

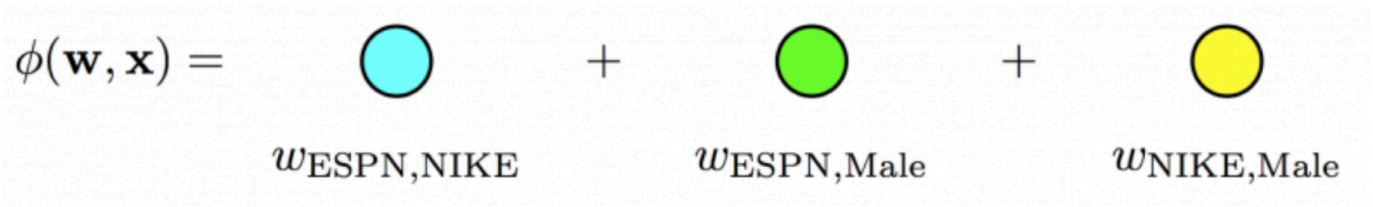
FM：为每个特征学习一个隐权重向量。在特征交叉时，使用两个特征隐向量的内积作为交叉特征的权重。

FFM：为每个特征学习一个隐权重“矩阵”，矩阵的每一列对应一个域，该列向量为对应的隐权重向量。在特征交叉时，使用两个特征对应域的隐向量的内积作为交叉特征的权重。

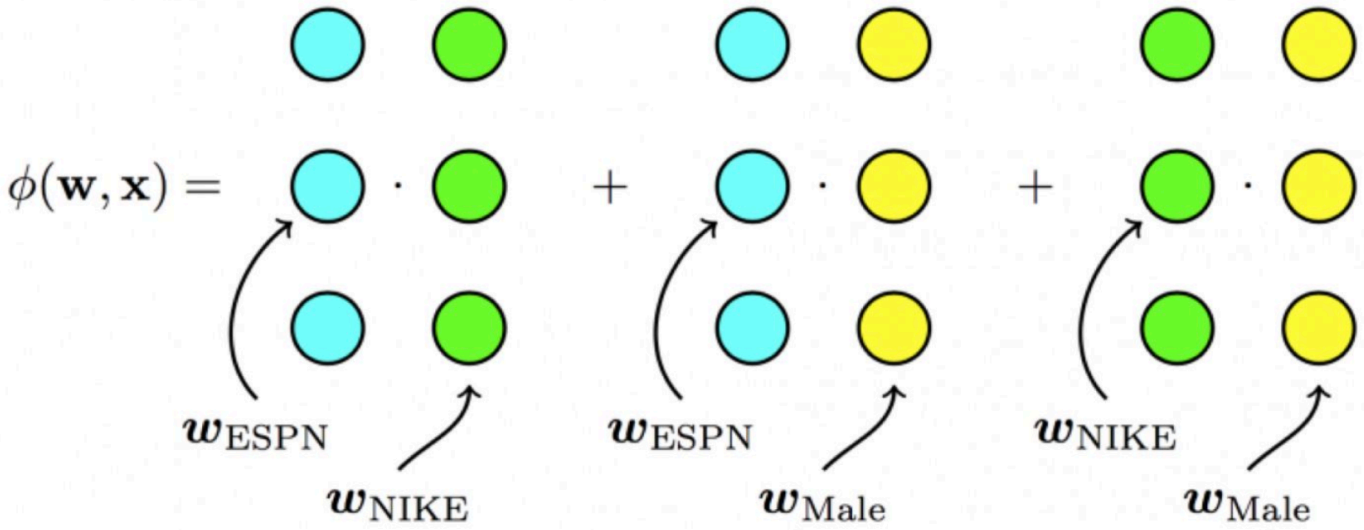
1.3 CTR 模型特征交叉方向的演化

以上模型实际上是 CTR 模型朝着特征交叉的方向演化的过程，我们再用图示方法回顾一下从 POLY2 到 FM，再到 FFM 进行特征交叉方法的不同。

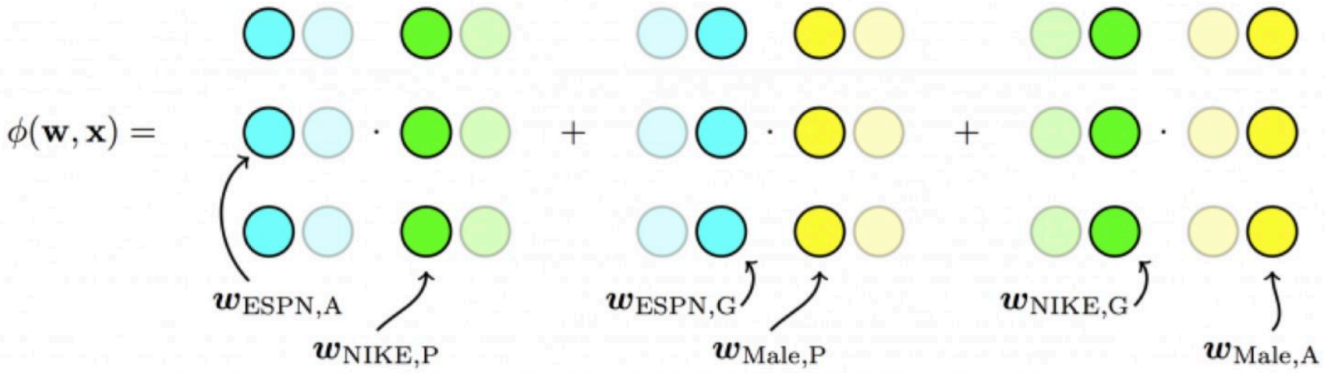
POLY2 模型直接学习每个交叉特征的权重，权重数量共 n^2 个。



FM 模型学习每个特征的 k 维隐向量，交叉特征由相应特征隐向量的内积得到，权重数量共 $n*k$ 个。



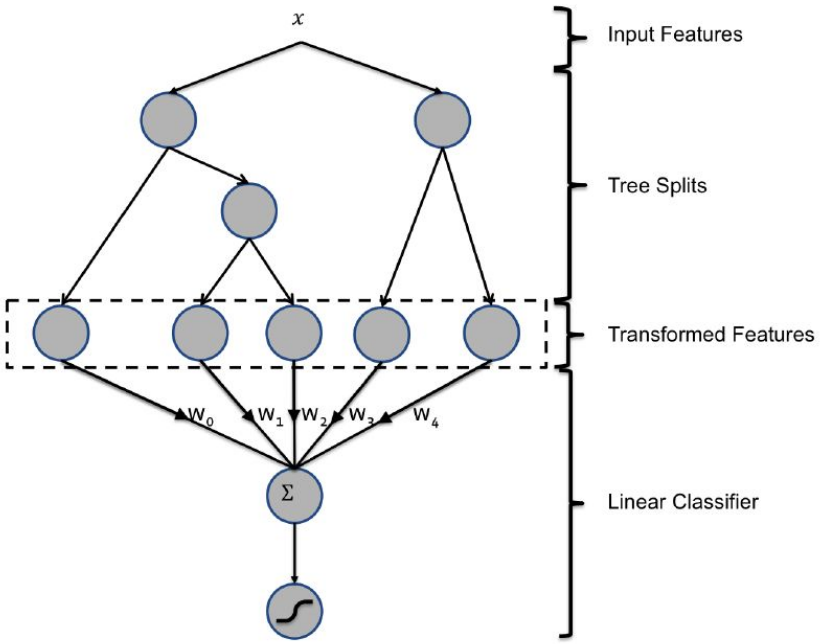
FFM 模型引入了特征域这一概念，在做特征交叉时，每个特征选择与对方域对应的隐向量做内积运算得到交叉特征的权重。参数数量共 $n*k*f$ 个。



1.4 GBDT+LR——特征工程模型化的开端

FFM 模型采用引入特征域的方式增强了模型的表达能力，但无论如何，FFM 只能够做二阶的特征交叉，如果要继续提高特征交叉的维度，不可避免的会发生组合爆炸和计算复杂度过高的情况。那么有没有其他的方法可以有效的处理高维特征组合和筛选的问题？2014 年，Facebook 提出了基于 GBDT+LR 组合模型的解决方案。

简而言之，Facebook 提出了一种利用 GBDT 自动进行特征筛选和组合，进而生成新的离散特征向量，再把该特征向量当作 LR 模型输入，预估 CTR 的模型结构。



大家知道，GBDT 是由多棵回归树组成的树林，后一棵树利用前面树林的结果与真实结果的残差做为拟合目标。每棵树生成的过程是一棵标准的回归树生成过程，因此每个节点的分裂是一个自然的特征选择的过程，而多层节点的结构自然进行了有效的特征组合，也就非常高效的解决了过去非常棘手的特征选择和特征组合的问题。

利用训练集训练好 GBDT 模型之后，就可以利用该模型完成从原始特征向量到新的离散型特征向量的转化。具体过程是这样的，一个训练样本在输入 GBDT 的某一子树后，会根据每个节点的规则最终落

入某一叶子节点，那么我们把该叶子节点置为 1，其他叶子节点置为 0，所有叶子节点组成的向量即形成了该棵树的特征向量，把 GBDT 所有子树的特征向量连接起来，即形成了后续 LR 输入的特征向量。

由于决策树的结构特点，事实上，**决策树的深度就决定了特征交叉的维度**。如果决策树的深度为 4，通过三次节点分裂，最终的叶节点实际上是进行了 3 阶特征组合后的结果，如此强的特征组合能力显然是 FM 系的模型不具备的。但由于 GBDT 容易产生过拟合，以及 GBDT 这种特征转换方式实际上丢失了大量特征的数值信息，因此我们不能简单说 GBDT 由于特征交叉的能力更强，效果就比 FFM 好，在模型的选择和调试上，永远都是多种因素综合作用的结果。

GBDT+LR 比 FM 重要的意义在于，**它大大推进了特征工程模型化这一重要趋势**，某种意义上来说，之后深度学习的各类网络结构，以及 embedding 技术的应用，都是这一趋势的延续。

1.5 FTRL——天下武功，唯快不破

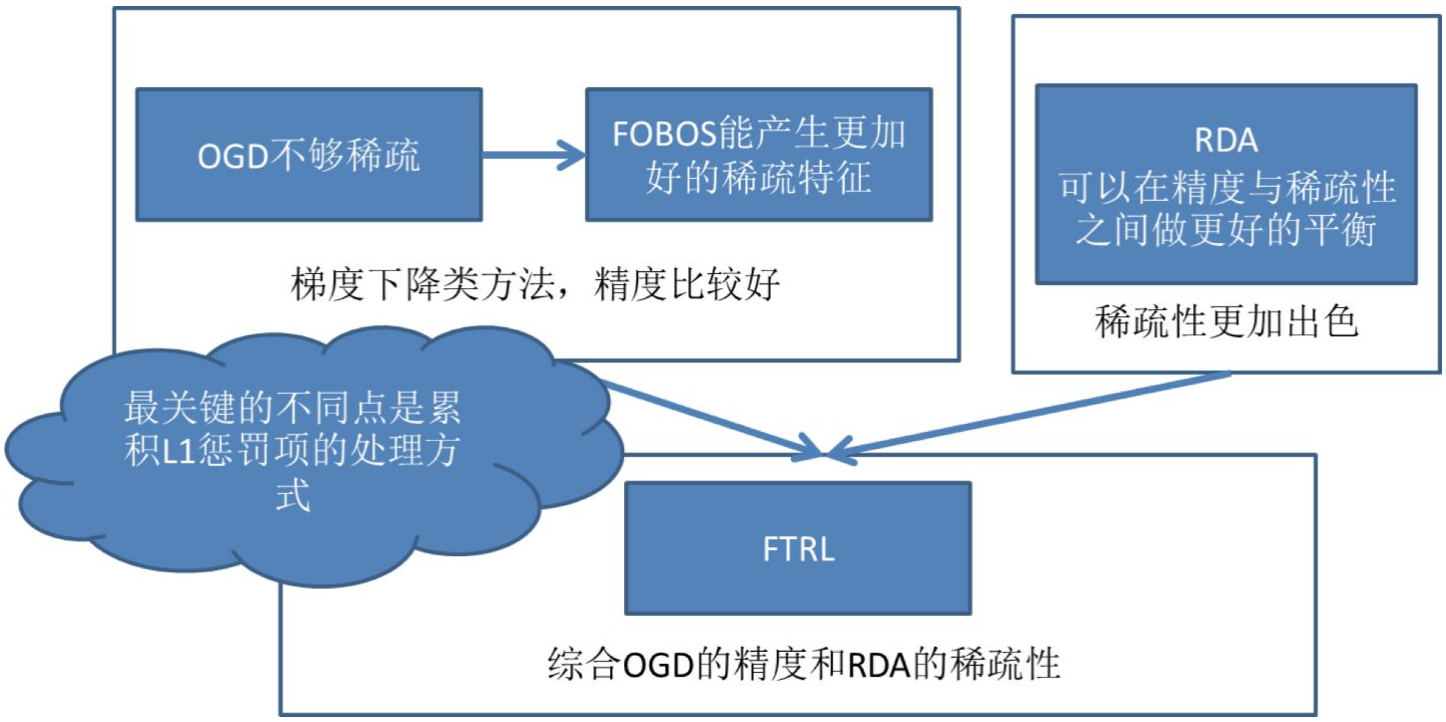
FTRL 的全称是 Follow-the-regularized-Leader，是一种在线实时训练模型的方法，Google 在 2010 年提出了 FTRL 的思路，2013 年实现了 FTRL 的工程化，之后快速成为 online learning 的主流方法。与模型演化图中的其他模型不同，FTRL 本质上是模型的训练方法。虽然 Google 的工程化方案是针对 LR 模型的，但理论上 FTRL 可以应用在 FM，NN 等任何通过梯度下降训练的模型上。

为了更清楚的认识 FTRL，这里对梯度下降方法做一个简要的介绍，从训练样本的规模角度来说，梯度下降可以分为：batch，mini-batch，SGD（随机梯度下降）三种，batch 方法每次都使用全量训练样本计算本次迭代的梯度方向，mini-batch 使用一小部分样本进行迭代，而 SGD 每次只利用一个样本计算梯度。对于 online learning 来说，为了进行实时得将最新产生的样本反馈到模型中，SGD 无疑是最合适的训练方式。

但 **SGD 对于互利网广告和推荐的场景来说，有比较大的缺陷，就是难以产生稀疏解**。为什么稀疏解对于 CTR 模型如此重要呢?

之前我们已经多次强调，由于 one hot 等 id 类特征处理方法导致广告和推荐场景下的样本特征向量极度稀疏，维度极高，动辄达到百万、千万量级。为了不割裂特征选择和模型训练两个步骤，如果能够在保证精度的前提下尽可能多的让模型的参数权重为 0，那么我们就可以自动过滤掉这些权重为 0 的特征，生成一个“轻量级”的模型。“轻量级”的模型不仅会使样本部署的成本大大降低，而且可以极大降低模型 inference 的计算延迟。这就是模型稀疏性的重要之处。

而 SGD 由于每次迭代只选取一个样本，梯度下降的方向虽然总体朝向全局最优解，但微观上的运动的过程呈现布朗运动的形式，这就导致 SGD 会使几乎所有特征的权重非零。即使加入 L1 正则化项，由于 CPU 浮点运算的结果很难精确的得到 0 的结果，也不会完全解决 SGD 稀疏性差的问题。就是在这样的前提下，**FTRL 几乎完美地解决了模型精度和模型稀疏性兼顾的训练问题**。



但 FTRL 的提出也并不是一蹴而就的。如上图所示，FTRL 的提出经历了下面几个关键的过程：

- **从最近简单的 SGD 到 OGD（online gradient descent）**，OGD 通过引入 L1 正则化简单解决稀疏性问题；
- **从 OGD 到截断梯度法**，通过暴力截断小数值梯度的方法保证模型的稀疏性，但损失了梯度下降的效率和精度；
- **FOBOS（Forward-Backward Splitting）**，google 和伯克利对 OGD 做进一步改进，09 年提出了保证精度并兼顾稀疏性的 FOBOS 方法；
- **RDA**：微软抛弃了梯度下降这条路，独辟蹊径提出了正则对偶平均来进行 online learning 的方法，其特点是稀疏性极佳，但损失了部分精度。
- Google **综合 FOBOS 在精度上的优势和 RDA 在稀疏性上的优势**，将二者的形式进行了进一步统一，提出并应用 FTRL，使 FOBOS 和 RDA 均成为了 FTRL 在特定条件下的特殊形式。

1.6 LS-PLM——阿里曾经的主流 CTR 模型

下面我们从样本 pattern 本身来入手，介绍阿里的的 LS-PLM（Large Scale Piece-wise Linear Model），它的另一个更广为人知的名字是 MLR（Mixed Logistic Regression）。MLR 模型虽然在 2017 年才公之于众，但其早在 2012 年就是阿里主流的 CTR 模型，并且在深度学习模型提出之前长时间应用于阿里的各类广告场景。

本质上，MLR 可以看做是对 LR 的自然推广，它在 LR 的基础上采用分而治之的思路，先对样本进行分片，再在样本分片中应用 LR 进行 CTR 预估。在 LR 的基础上加入聚类的思想，其动机其实来源于对计算广告领域样本特点的观察。

举例来说，如果 CTR 模型要预估的是女性受众点击女装广告的 CTR，显然我们并不希望把男性用

户点击数码类产品的样本数据也考虑进来，因为这样的样本不仅对于女性购买女装这样的广告场景毫无相关性，甚至会在模型训练过程中扰乱相关特征的权重。为了让 CTR 模型对不同用户群体，不用用户场景更有针对性，其实理想的方法是先对全量样本进行聚类，再对每个分类施以 LR 模型进行 CTR 预估。MLR 的实现思路就是由该动机产生的。

$$f(x) = \sum_{i=1}^{m} \pi_i(x) \cdot \eta_i(x) = \sum_{i=1}^{m} \frac{e^{\mu_i \cdot x}}{\sum_{j=1}^{m} e^{\mu_j \cdot x}} \cdot \frac{1}{1 + e^{-w_i \cdot x}}$$

MLR 目标函数的数学形式如上式，首先用聚类函数 π 对样本进行分类（这里的 π 采用了 softmax 函数，对样本进行多分类），再用 LR 模型计算样本在分片中具体的 CTR，然后将二者进行相乘后加和。

其中超参数分片数 m 可以较好地平衡模型的拟合与推广能力。当 $m = 1$ 时 MLR 就退化为普通的 LR，m 越大模型的拟合能力越强，但是模型参数规模随 m 线性增长，相应所需的训练样本也随之增长。在实践中，阿里给出了 m 的经验值为 12。

MLR 算法适合于工业级的广告、推荐等大规模稀疏数据场景问题。主要是由于表达能力强、稀疏性高等两个优势

(1). **端到端的非线性学习**：从模型端自动挖掘数据中蕴藏的非线性模式，省去了大量的人工特征设计，这使得 MLR 算法可以端到端地完成训练，在不同场景中的迁移和应用非常轻松。

(2). **稀疏性**：MLR 在建模时引入了 L1 和 L2,1 范数，可以使得最终训练出来的模型具有较高的稀疏度，模型的学习和在线预测性能更好。

如果我们用深度学习的眼光来看待 MLR 这个模型，其在结构上已经很接近由输入层、单隐层、输出层组成的神经网络。所以某种意义上说，MLR 也在用自己的方式逐渐逼近深度学习的大门了。

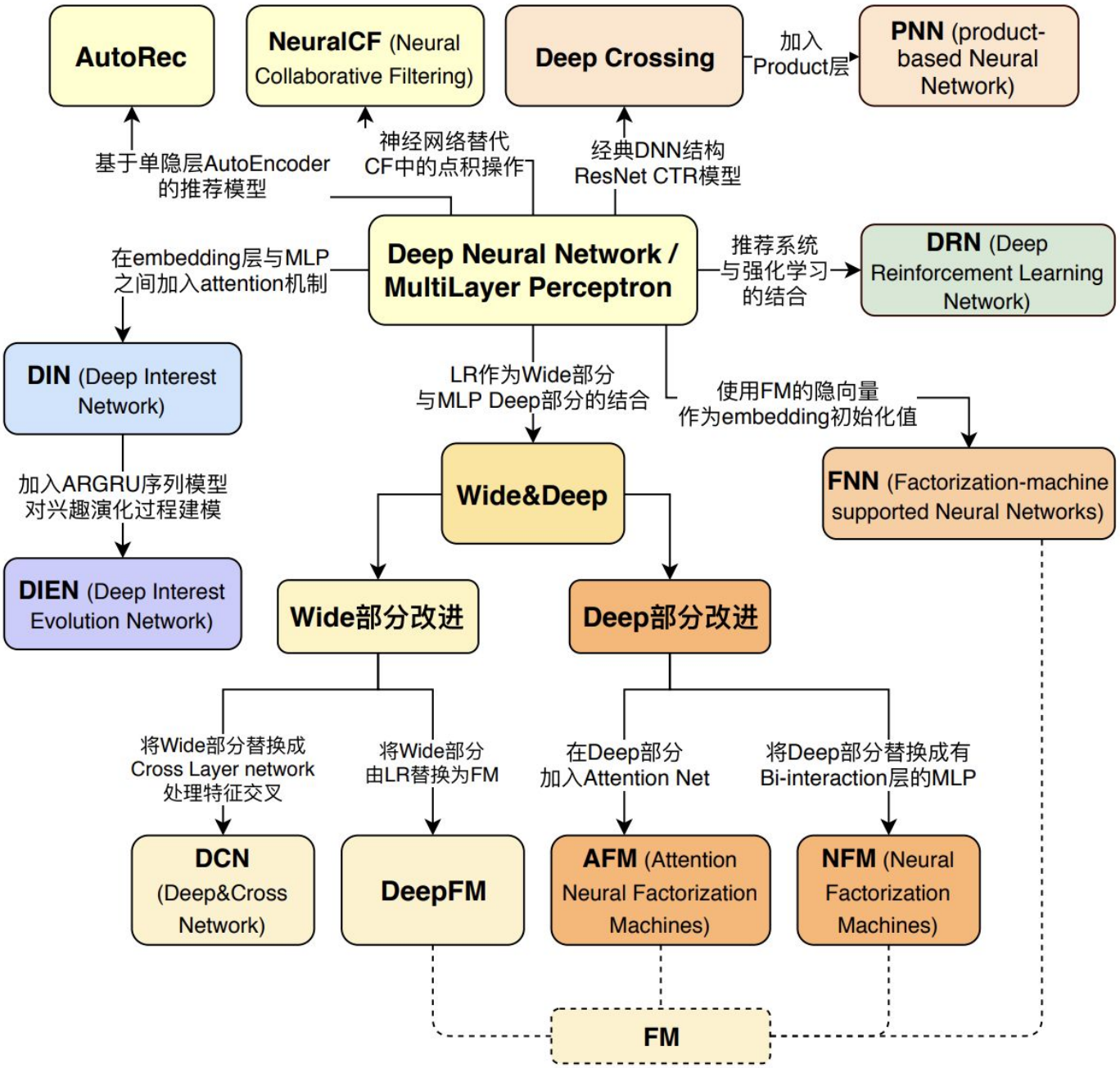
1.7 深度学习 CTR 模型的前夜

2010 年 FM 被提出，特征交叉的概念被引入 CTR 模型；2012 年 MLR 在阿里大规模应用，其结构十分接近三层神经网络；2014 年 Facebook 用 GBDT 处理特征，揭开了特征工程模型化的篇章。这些概念都将在深度学习 CTR 模型中继续应用，持续发光。

另一边，Alex Krizhevsky 2012 年提出了引爆整个深度学习浪潮的 AlexNet，深度学习的大幕正式拉开，其应用逐渐从图像扩展到语音，再到 NLP 领域，推荐和广告也必然会紧随其后，投入深度学习的大潮之中。

2016 年，随着 FNN，Deep&Wide，Deep crossing 等一大批优秀的 CTR 模型框架的提出，深度学习 CTR 模型逐渐席卷了推荐和广告领域，成为新一代 CTR 模型当之无愧的主流。

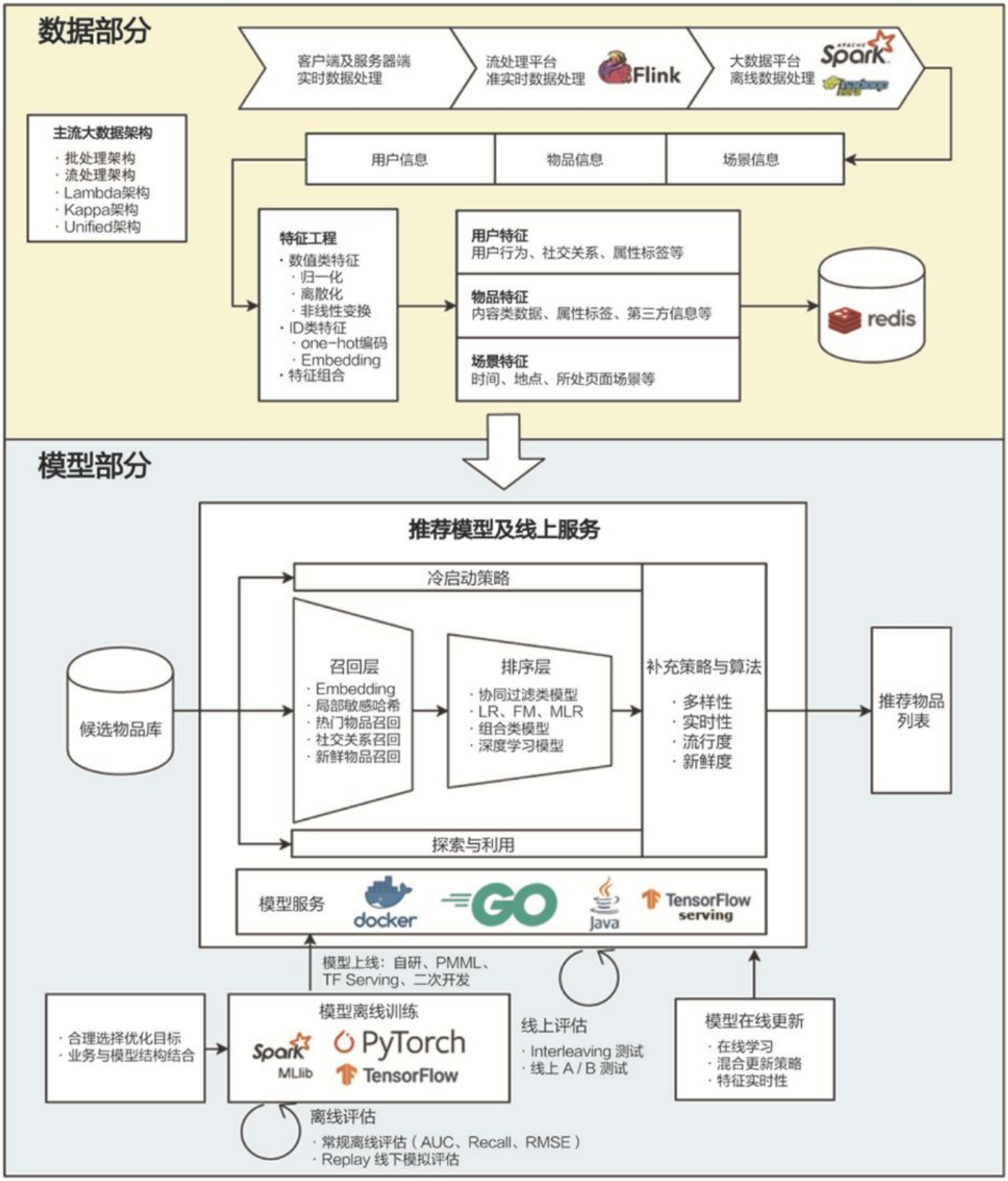
2 深度学习推荐模型的演化趋势



上图中涉及到的主要知识点如下:

- Product 层
- AutoEncoder
- Attention 机制
- ARGRU
- Cross Layer Network
- Bi-interaction
- ……

3　推荐系统的系统架构


数据部分
客户端及服务器端
实时数据处理
流处理平台
准实时数据处理
Flink
大数据平台
离线数据处理
Spark
主流大数据架构
· 批处理架构
· 流处理架构
· Lambda架构
· Kappa架构
· Unified架构
用户信息
物品信息
场景信息
特征工程
· 数值类特征
· 归一化
· 离散化
· 非线性变换
· ID类特征
· one-hot编码
· Embedding
· 特征组合
用户特征
用户行为、社交关系、属性标签等
物品特征
内容类数据、属性标签、第三方信息等
场景特征
时间、地点、所处页面场景等
redis
模型部分
推荐模型及线上服务
冷启动策略
候选物品库
召回层
· Embedding
· 局部敏感哈希
· 热门物品召回
· 社交关系召回
· 新鲜物品召回
排序层
· 协同过滤类模型
· LR、FM、MLR
· 组合类模型
· 深度学习模型
补充策略与算法
· 多样性
· 实时性
· 流行度
· 新鲜度
推荐物品列表
探索与利用
模型服务
docker
GO
Java
TensorFlow serving
模型上线：自研、PMML、
TF Serving、二次开发
· 合理选择优化目标
· 业务与模型结构结合
模型离线训练
Spark MLlib
PyTorch
TensorFlow
线上评估
· Interleaving 测试
· 线上 A / B 测试
模型在线更新
· 在线学习
· 混合更新策略
· 特征实时性
离线评估
· 常规离线评估（AUC、Recall、RMSE）
· Replay 线下模拟评估

Part II

推荐系统的工业化应用

4 Facebook 的经典 CTR 预估模型

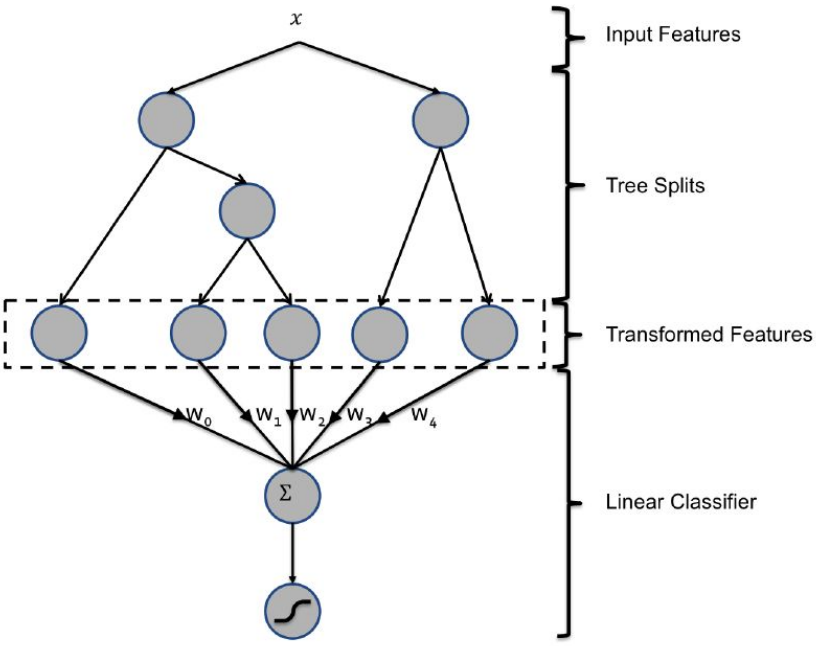
论文：Practical Lessons from Predicting Clicks on Ads at Facebook

4.1 用户场景

论文中的用户场景是一个标准的点击率预估的场景，需要强调的只有一点，因为我们需要利用 CTR 计算精准的出价、ROI 等重要的后续预估值，因此 CTR 模型的预估值需要是一个**具有物理意义的精准的 CTR，而不是仅仅输出广告排序的高低关系**。所以文中不仅把 CTR calibration 作为重要的评价指标，更是在最后介绍了模型校正的相关方法。

4.2 模型结构

文章提出了一种利用 GBDT 自动进行特征筛选和组合，进而生成新的 feature vector，再把该 feature vector 当作 logistic regression 的模型输入，预测 CTR 的模型结构。



这里需要强调的是，**用 GBDT 构建特征工程，和利用 LR 预测 CTR 两步是独立训练的**。所以自然不存在如何将 LR 的梯度回传到 GBDT 这类复杂的问题，而利用 LR 预测 CTR 的过程是显然的，在此不再赘述，我们着重讲一讲如何利用 GBDT 构建新的特征向量。

大家知道，GBDT 是由多棵回归树组成的树林，后一棵树利用前面树林的结果与真实结果的残差做为拟合目标。每棵树生成的过程是一棵标准的回归树生成过程，因此每个节点的分裂是一个自然的特征选择的过程，而多层节点的结构自然进行了有效的特征组合，也就非常高效的解决了过去非常棘手的特征选择和特征组合的问题。

我们利用训练集训练好 GBDT 模型，之后就可以利用该模型构建特征工程。具体过程是这样的，一个样本在输入 GBDT 的某一子树后，会根据每个节点的规则最终落入某一叶子节点，那么我们把该叶子节点置为 1，其他叶子节点置为 0，所有叶子节点组成的向量即形成了该棵树的特征向量，把 GBDT 所有子树的特征向量 concatenate 起来，即形成了后续 LR 输入的特征向量。

举例来说，比如 GBDT 由三颗子树构成，每个子树有 4 个叶子节点，一个训练样本进来后，先后落到了“子树 1”的第 3 个叶节点中，那么特征向量就是 [0,0,1,0]，“子树 2”的第 1 个叶节点，特征向量为 [1,0,0,0]，“子树 3”的第 4 个叶节点，特征向量为 [0,0,0,1]，最后 concatenate 所有特征向量，形成的最终的特征向量为 [0,0,1,0,1,0,0,0,0,0,0,1]，我们再把该向量作为 LR 的输入，预测 CTR。

引入了 GBDT+LR 的模型后，相比单纯的 LR 和 GBDT，提升效果是非常显著的。从论文中可以看到，混合模型比单纯的 LR 或 Trees 模型在 loss 上减少了 3

该模型的优势我们上面已经提到，即可以自动进行特征组合和特征筛选，但在实践过程中，模型的缺陷也比较明显，相比 FTRL，FM，NN 等能够通过梯度下降训练的模型来说，**GBDT 缺乏 online learning 的能力，因此我们往往只能相隔一天甚至几天才能够 update GBDT 模型，势必影响模型的实效性**，那么 Facebook 是如何解决模型更新的问题的呢?

4.3 模型的实效性问题和更新策略

虽然我们的直觉是模型的训练时间和 serving 时间之间的间隔越短，模型的效果越好，但为了证明这一点，facebook 的工程师还是做了一组实效性的实验，在结束模型的训练之后，观察了其后 6 天的模型 loss（这里采用 normalized entropy 作为 loss）

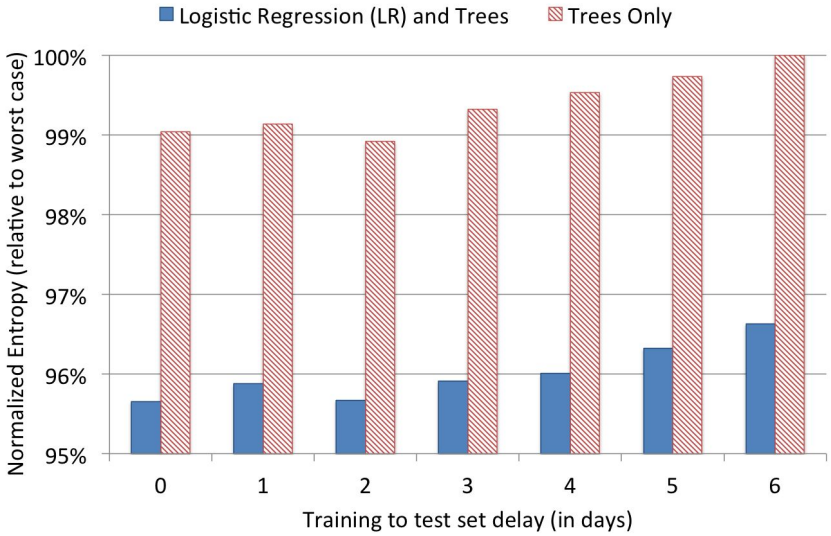


Figure 1: 模型更新延迟与 loss 的关系

可以看出，模型的 loss 在第 0 天之后有所上升，特别是第 2 天过后显著上升。因此 daily update 的模型相比 weekly update 的模型效果肯定是有大幅提升的。

但囿于 facebook 巨大的数据量以及 GBDT 较难实施并行化的原因，GBDT 的更新时间往往超过 24 小时，所以为了兼顾 data freshness 和客观的工程要求，facebook 采取了下面的模型更新方法：

The boosted decision trees can be trained daily or every couple of days, but the linear classifier can be trained in near real-time by using some flavor of online learning.

就是说 GBDT 的部分几天更新一次，而 LR 的部分进行准实时的更新，这无疑是很好的工程实践经验。时至今日，我们已经开始使用大量不同的 embedding 方法进行特征编码，facebook 当时的做法也对我们现在的工程实践有重要的参考价值。因为大量深度学习 embedding 方法的更新计算开销也非常大，但对实效性要求并不高，我们也完全可以低频更新 embedding，高频或实时更新基于 embedding 特征的 LR，NN 等预测模型。

4.4 facebook 的实时数据流架构

为了实现模型的准实时训练，facebook 专门介绍了其基于 Scribe 的数据流架构，文中称其为 online data joiner。

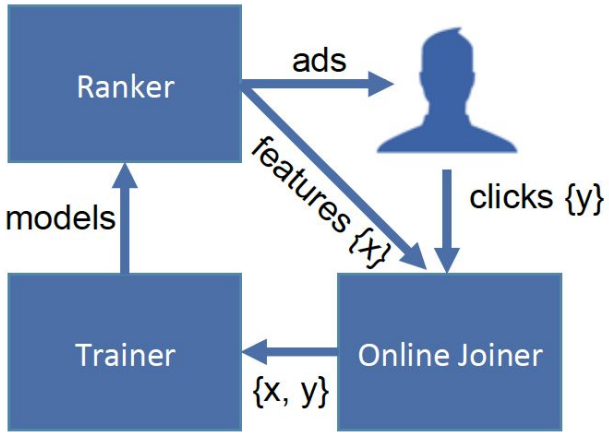


Figure 4: Online Learning Data/Model Flows.

该模块最重要的作用是准实时的把来自不同数据流的数据整合起来形成 sample features，并最终与 click 数据进行 join，形成完整的 labeled sample。在整个过程中，我认为最应该注意的有三点：

(1). **waiting window 的设定**：waiting window 指的是在 impression 发生后，我们要等待多久才能够判定一个 impression 是否有 click。如果 waiting window 过大，数据实时性受影响，如果 waiting window 过小，会有一部分 click 来不及 join 到 impression，导致样本 CTR 与真实值不符。这是一个工程调优的问题，需要有针对性的找到跟实际业务匹配的合适的 waiting window。除此之外，无论怎样我们都会漏掉一部分 click，这就要求我们阶段性的全量 retrain 我们的模型，避免 online learning 误差的积累。

(2). **分布式的架构与全局统一的 action id**：为了实现分布式架构下 impression 记录和 click 记录的 join，facebook 除了为每个 action 建立全局统一的 request id 外，还建立了 HashQueue 缓存 impressions。

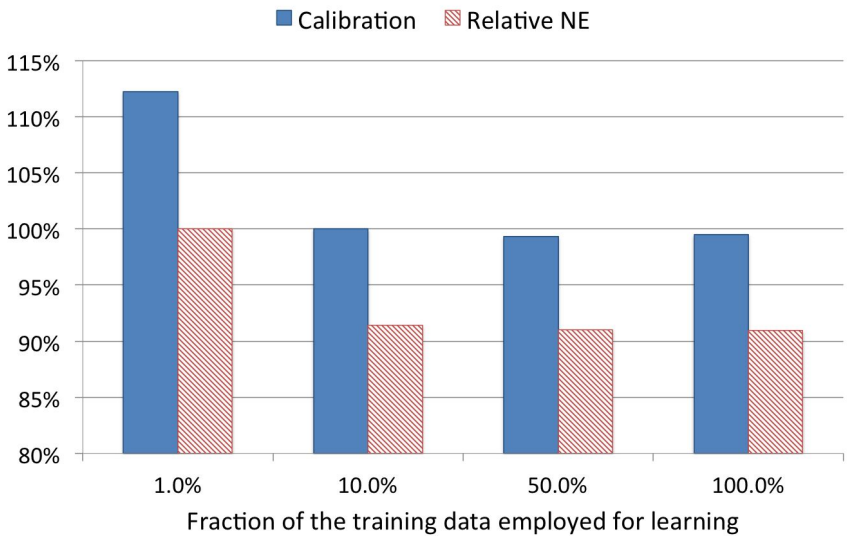
hashQueue 缓存的 impression，如果在窗口过期时还没有匹配到 click 就会当作 negative sample，这里说的窗口与上面提到的 waiting window 相匹配。facebook 使用 scribe 实现了这一过程，更多公司使用 Kafka 完成大数据缓存和流处理。

(3). **数据流保护机制**：facebook 专门提到了 online data joiner 的保护机制，因为一旦 data joiner 失效，比如 click stream 无法 join impression stream，那么所有的样本都会成为负样本，由于模型实时进行 online learning 和 serving，模型准确度将立刻受到错误样本数据的影响，进而直接影响广告投放和公司利润，后果是非常严重的。为此，facebook 专门设立了异常检测机制，一旦发现实时样本数据流的分布发生变化，将立即切断 online learning 的过程，防止预测模型受到影响。

4.5 降采样和模型校正

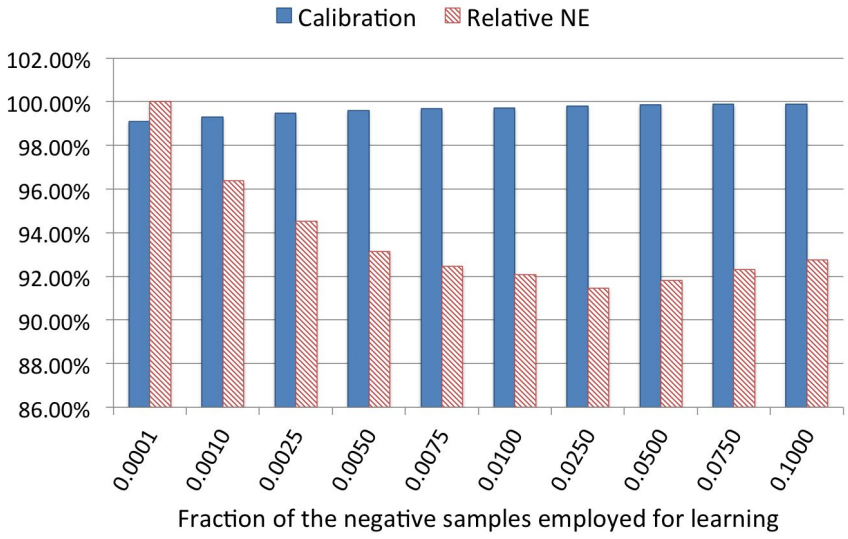
对于巨型互联网公司来说，为了控制数据规模，降低训练开销，降采样几乎是通用的手段，facebook 实践了两种降采样的方法，uniform subsampling 和 negative down sampling

uniform subsampling 是对所有样本进行无差别的随机抽样，为选取最优的采样频率，facebook 试验了 0.001，0.01，0.1，0.5 和 1 五个采样频率，loss 的比较如下：



可以看到当采样率是 10

另一种方法 negative down sampling 保留全量正样本，对负样本进行降采样。除了提高训练效率外，负采样还直接解决了正负样本不均衡的问题，facebook 经验性的选择了从 0.0001 到 0.1 的一组负采样频率，试验效果如下：



大家可以看到，当负采样频率在 0.025 时，loss 不仅优于更低的采样频率训练出来的模型，居然也优于负采样频率在 0.1 时训练出的模型，虽然原文没有作出进一步的解释，但推测最可能的原因是解决了数据不均衡问题带来的效果提升。

负采样带来的问题是 CTR 预估值的漂移，比如真实 CTR 是 0.1%，进行 0.01 的负采样之后，CTR 将会攀升到 10% 左右。而为了进行准确的竞价以及 ROI 预估等，CTR 预估模型是要提供准确的有物理意义的 CTR 值的，因此在进行负采样后需要进行 CTR 的校正，使 CTR 模型的预估值的期望回到 0.1%。校正的公式如下：

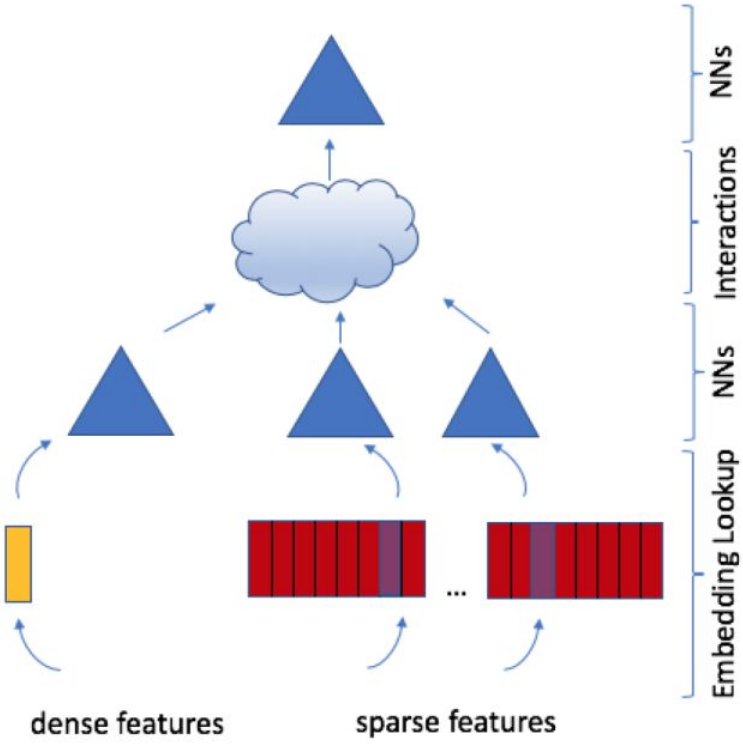
$$q = \frac{p}{p + (1-p)/w}$$

其中 q 是校正后的 CTR，p 是模型的预估 CTR，w 是负采样频率。大家可以利用简单的转换关系就可以得出上述公式，有兴趣的同学可以手动推导一下。

5 Facebook 的 DLRM 模型

论文：Deep Learning Recommendation Model for Personalization and Recommendation Systems

5.1 模型架构

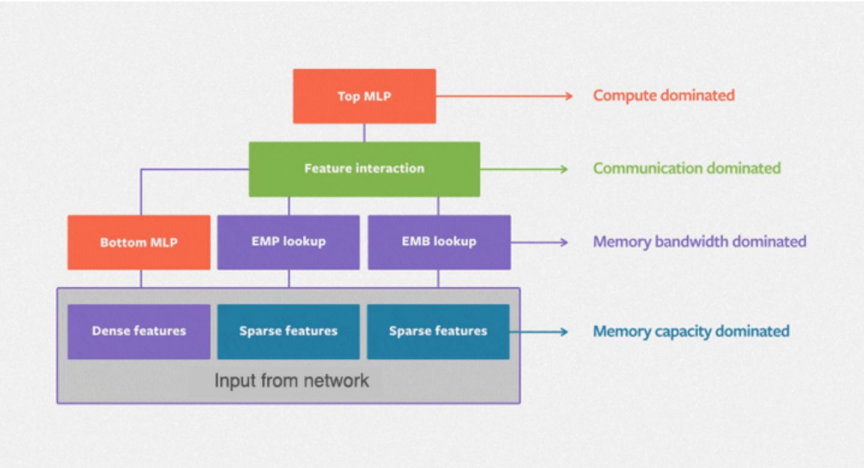


从下到上来解释一下 Facebook DLRM 这个模型的模型结构：

- **特征工程**：所有特征被分为两类，一类是将类别、id 类特征用 one hot 编码生成的稀疏特征（sparse features），一类是数值型连续特征（dense features）
- **Embedding 层**：每个类别类特征转换成 one hot vector 后，用 embedding 层转换成维度为 n 的 embedding。也就是说，每种稀疏特征转换成一个 embedding 向量。而年龄、收入等连续型特征将被 concat 成一个特征向量后，输入图中黄色的 MLP（Multi Layer Perceptron）中，被转化成同样维度为 n 的向量。至此，无论是类别类稀疏特征，还是连续型特征组成的特征向量在经过 Embedding 层后，都被转换成了 n 维的 embedding 向量。
- **NNs 层**：Embedding 再往上是由三角形代表的 MLP 神经网络层，也就是说得到 n 维的 embedding 向量后，每类 embedding 还有可能进一步通过 MLP 做转换，原文中这么说的 “optionally passing them through MLPs”，就是说选择性的通过 MLP 做进一步转换，看来那三个三角形其实是根据调参情况可有可无的。
- **interactions 层**：这一层其实这篇文章相对还算创新的一点，它是怎么做的呢？原文这么说的：This is done by taking the dot product between all pairs of embedding vectors and processed dense features. These dot products are concatenated with the original processed dense features and post-processed with another MLP (the top or output MLP)。也就是说会将之前的 embedding 两两做点积，做完之后再跟之前 dense features 对应的 embedding concat 起来，喂给后续的 MLP。所以这一步其实是希望特征之间做充分的交叉，组合之后，再进入上层 MLP 做最终的目标拟合。这一点其实 follow 了 FM 的特征交叉概念。

- 最上层那个三角不用多说，是另一个 FC NN，并最终用 sigmoid 函数给出最终的点击率预估。

整个模型看下来，没有特别 fancy 的结构，也没有加入 sequential model，RF 等模型的思路，可以说是一个工业界的标准深度学习模型。facebook 的博客上还给出了稍微详细一点的模型结构图，大家可以参考。



5.2 Facebook 的模型并行训练方法

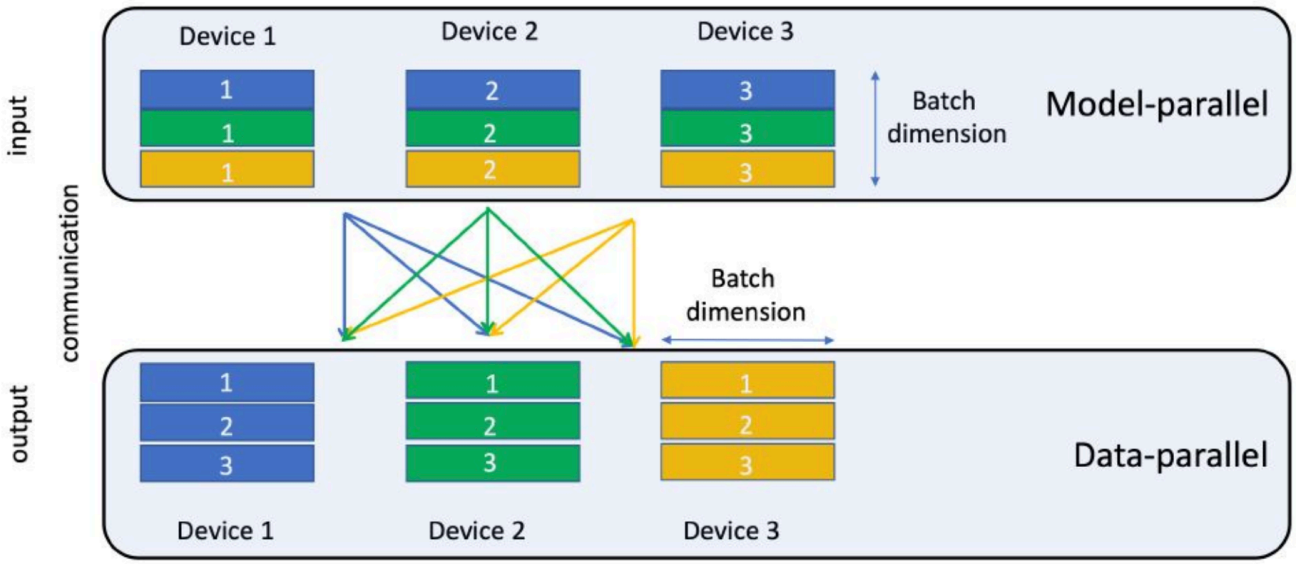
作为一篇业界的论文，模型的实际训练方法，训练平台往往是让人受益最多的。而对于 Facebook 的数据量来说，单节点的模型训练必然无法快速完成任务，那么模型的并行训练就肯定是少不了的解决方法。我们先看原文是如何解释 DLRM 这个模型的并行训练过程的：

Our parallelized DLRM will use a combination of model parallelism for the embeddings and data parallelism for the MLPs to mitigate the memory bottleneck produced by the embeddings while parallelizing the forward and backward propagations over the MLPs.

简单来说，DLRM 融合使用了模型并行和数据并行的方法，对于 Embedding 部分采用了模型并行，对于 MLP 部分采用了数据并行。Embedding 部分采用模型并行的原因是减轻大量 Embedding 参数带来的内存瓶颈问题。MLP 部分采用数据并行是可以并行进行前向和反向传播。

其实原文中并没有给出非常准确的并行训练方法，这里凭我自己的理解进一步解释一下 Embedding 做模型并行训练和上层 MLP 做数据并行训练的原理，对 pytorch 和 caffe2 熟悉的专家可以随时纠正我的解释：

- Embedding 做模型并行训练指的是在一个 device 或者说计算节点上，仅有一部分 Embedding 层参数，每个 device 进行并行 mini batch 梯度更新时，仅更新自己节点上的部分 Embedding 层参数。
- MLP 层和 interactions 进行数据并行训练指的是每个 device 上已经有了全部模型参数，每个 device 上利用部分数据计算 gradient，再利用 allreduce 的方法汇总所有梯度进行参数更新。



模型并行训练示意图

Figure 2: 模型并行训练示意图

5.3 DLRM 模型的效果

在性能的对比上，DLRM 选择了 Google 2017 年的 DCN（Deep cross network）作为 baseline。

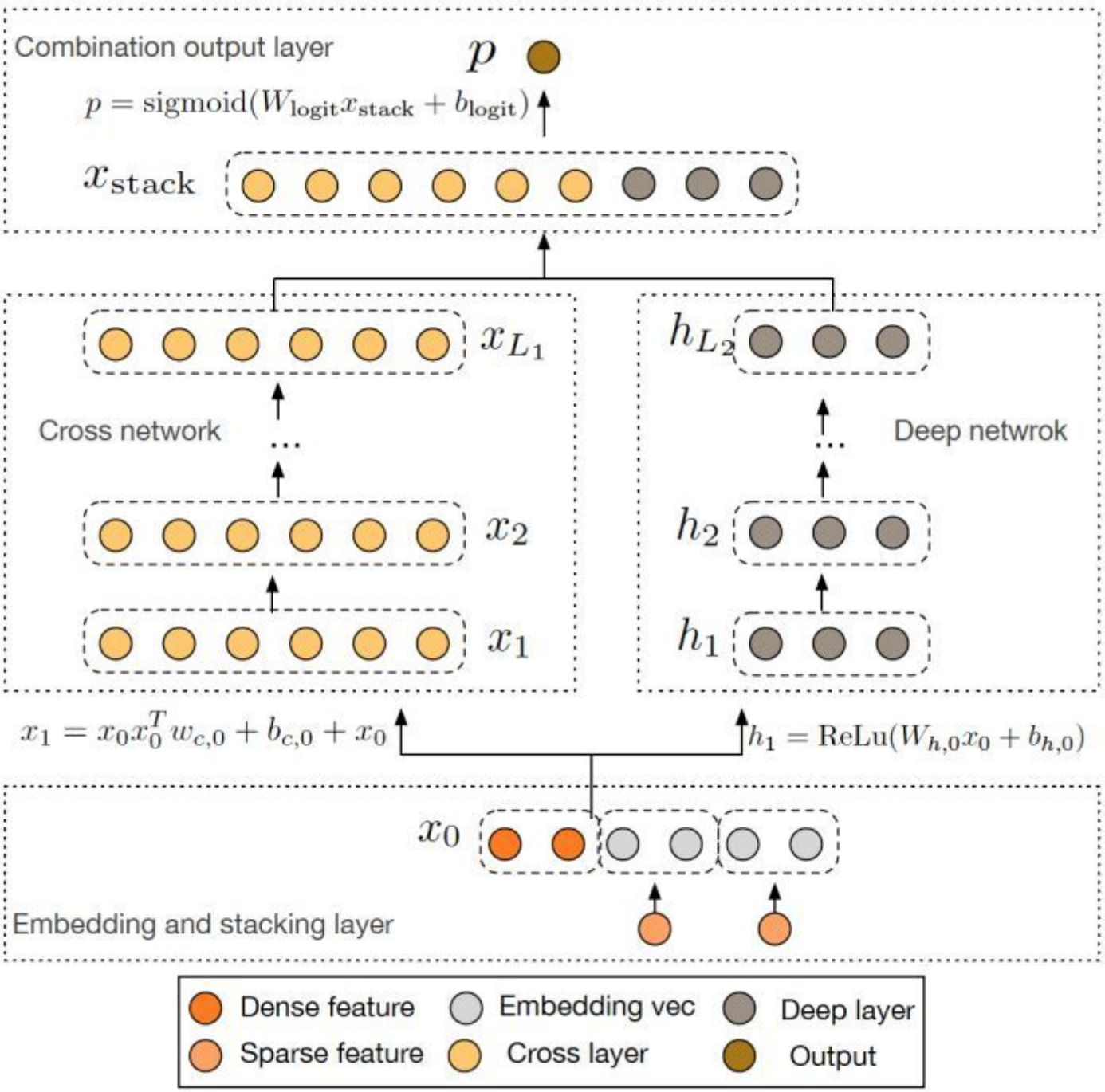
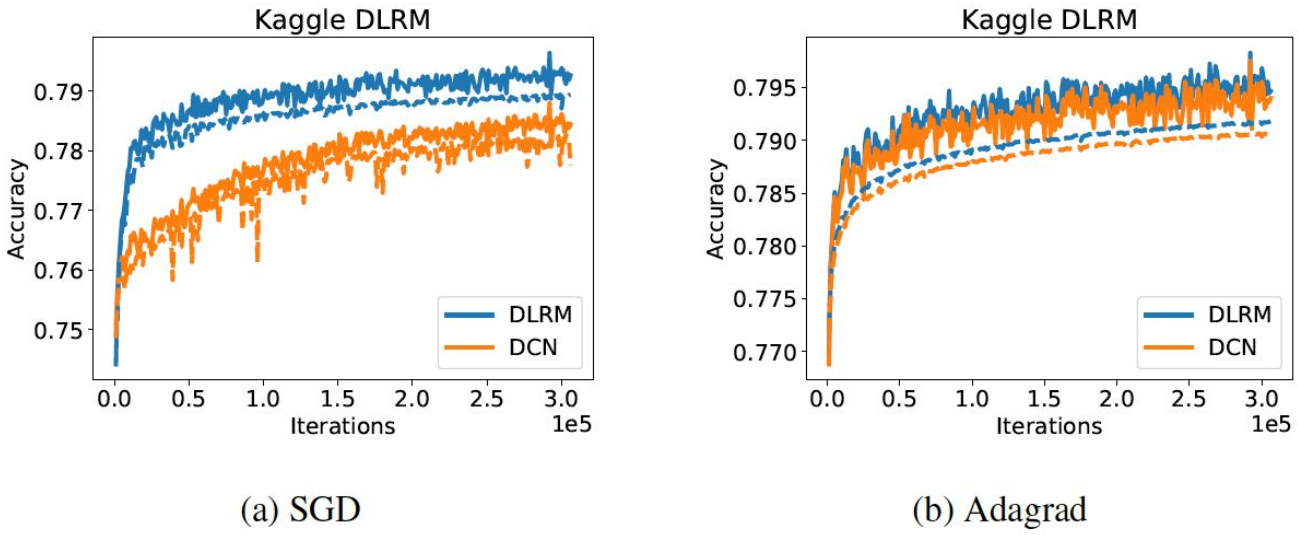


Figure 3: Google DCN 模型结构

DCN 可以看作是对 Wide&Deep 模型的进一步改进，主要的思路使用 Cross 网络替代了原来的 Wide 部分。其中设计 Cross 网络的基本动机是为了增加特征之间的交互力度，使用多层 cross layer 对输入向量进行特征交叉。单层 cross layer 的基本操作是将 cross layer 的输入向量 xl 与原始的输入向量 x0 进行交叉，并加入 bias 向量和原始 xl 输入向量。

可以看出 DLRM 和 DCN 的主要区别在于特征交叉方式的不同，DLRM 采用了不同特征域两两点积的交叉方式，DCN 采用了每个维度两两交叉的方式。利用 Criteo Ad Kaggle data 作为测试集，二者的性能对比如下：



(a) SGD (b) Adagrad

Figure 4: DLRM 与 DCN 性能对比

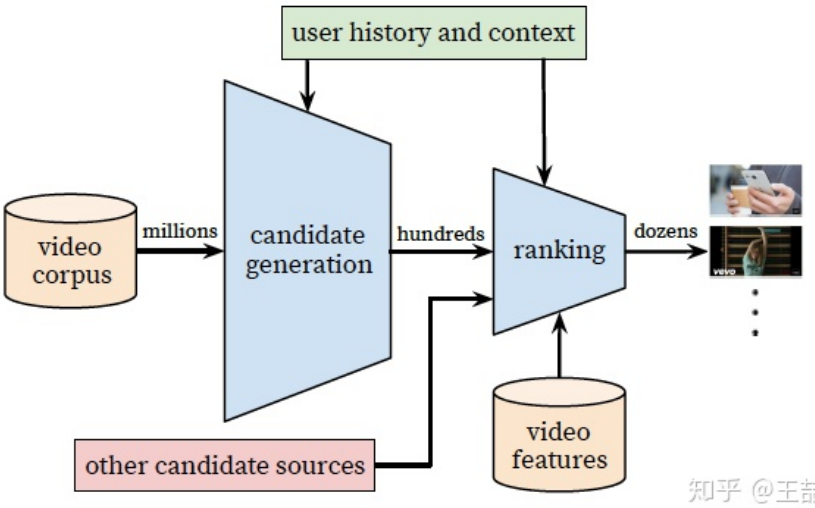
可以看到，DLRM 在 Accuracy 上稍胜一筹。当然模型的 performance 与数据集的选择，参数的调优都有很大关系。而且 DLRM 在 Adagrad 训练方式下的优势已经微乎其微，这里的性能评估大家仅做参考即可。

6 Youtube 的深度学习推荐系统

论文：Deep Neural Networks for YouTube Recommendations

6.1 算法基本架构

Youtube 的算法架构如下：



Youtube 的用户推荐场景自不必多说，作为全球最大的 UGC 的视频网站，需要在百万量级的视频

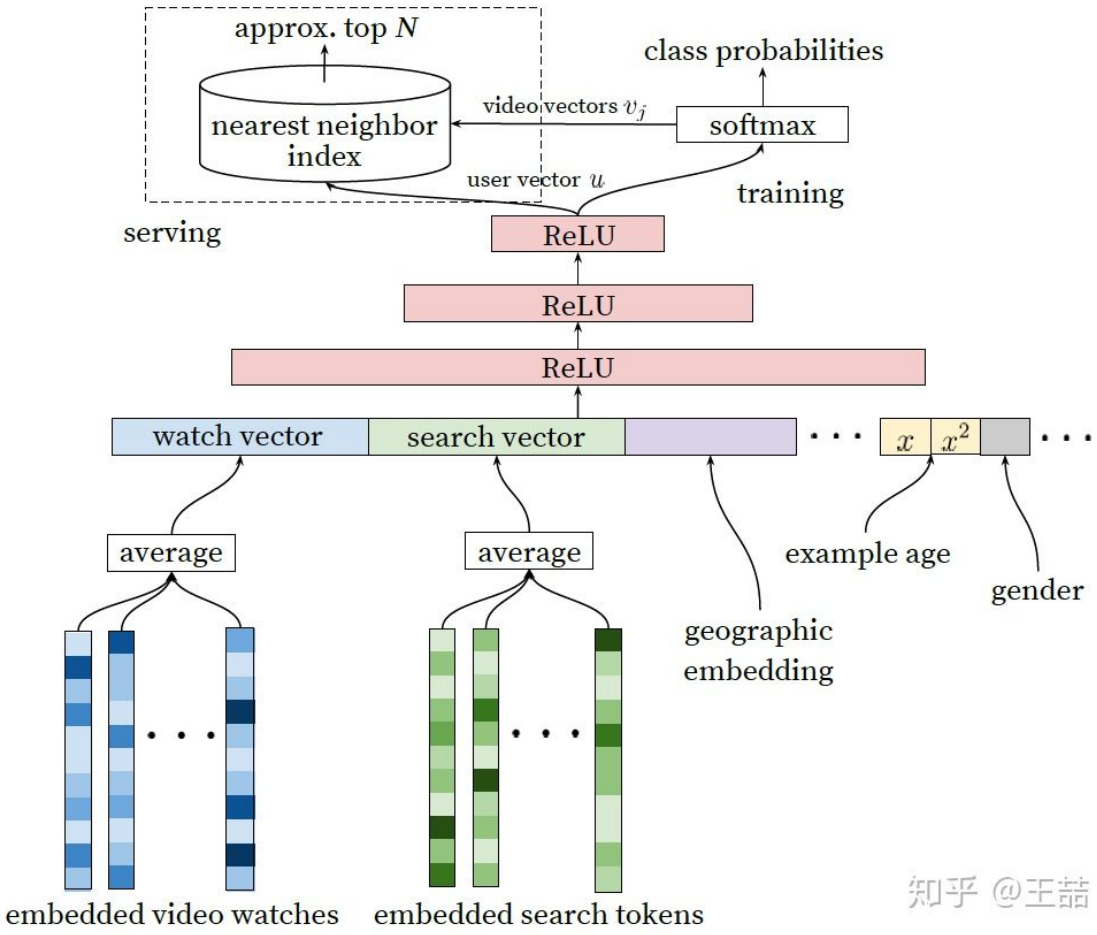
规模下进行个性化推荐。由于候选视频集合过大，考虑 online 系统延迟问题，不宜用复杂网络直接进行推荐，所以 Youtube 采取了两层深度网络完成整个推荐过程：

(1). 第一层是 Candidate Generation Model 完成候选视频的快速筛选，这一步候选视频集合由百万降低到了百的量级。

(2). 第二层是用 Ranking Model 完成几百个候选视频的精排

6.2 召回层

首先介绍 candidate generation 模型的架构：



我们自底而上看这个网络，最底层的输入是用户观看过的 video 的 embedding 向量，以及搜索词的 embedding 向量。至于这个 embedding 向量是怎么生成的，作者的原话是这样的

Inspired by continuous bag of words language models, we learn high dimensional embeddings for each video in a fixed vocabulary and feed these embeddings into a feedforward neural network

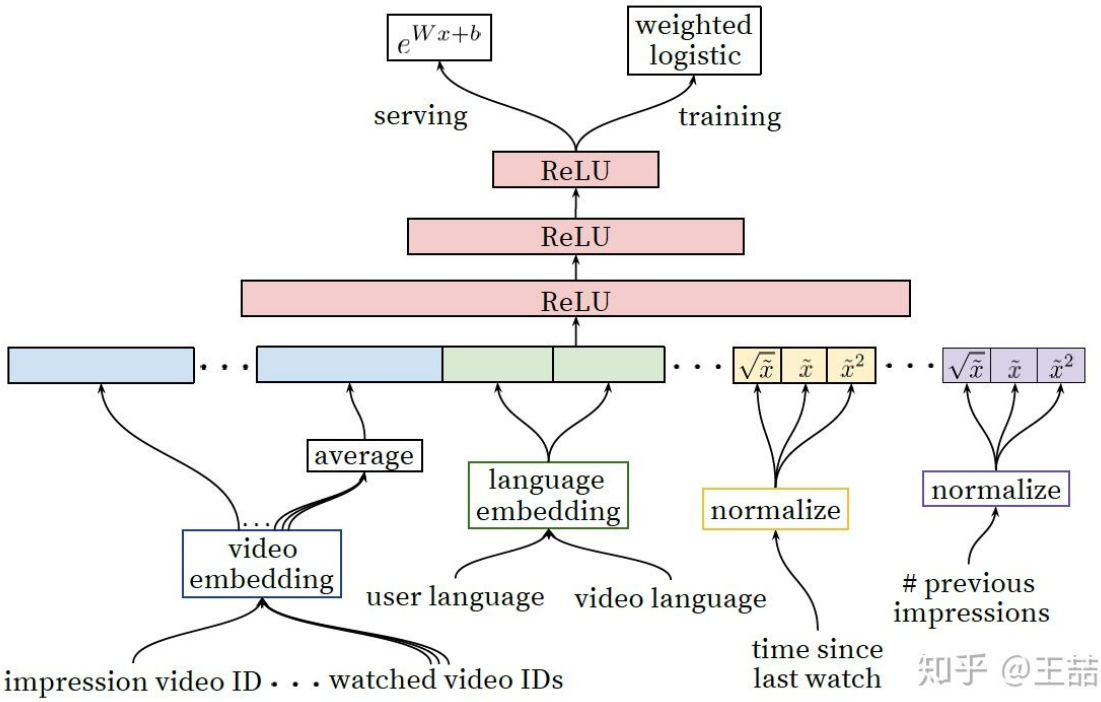
所以作者是先用 word2vec 方法对 video 和 search token 做了 embedding 之后再作为输入的，这也是做 embedding 的 “基本操作”，不用过多介绍；当然，除此之外另一种大家应该也比较熟悉，就是通过加一个 embedding 层跟上面的 DNN 一起训练，两种方法孰优孰劣，有什么适用场合，大家可以讨论一下。

特征向量里面还包括了用户的地理位置的 embedding，年龄，性别等。然后把所有这些特征 concatenate 起来，喂给上层的 ReLU 神经网络。

三层神经网络过后，我们看到了 softmax 函数。这里 Youtube 的同学们把这个问题看作为用户推荐 next watch 的问题，所以输出应该是一个在所有 candidate video 上的概率分布，自然是一个**多分类**问题。

6.3 排序层

既然得到了几百个候选集合，下一步就是利用 ranking 模型进行精排序，下面是 ranking 深度学习网络的架构图。



乍一看上面的 ranking model 似乎与 candidate generation 模型没有什么区别，模型架构还是深度学习的“基本操作”，唯一的区别就是特征工程，那么我们就讲讲特征工程。

事实上原文也明确说明了，引入另一套 DNN 作为 ranking model 的目的就是引入更多描述视频、用户以及二者之间关系的特征，达到对候选视频集合准确排序的目的。

During ranking, we have access to many more features describing the video and the user's relationship to the video because only a few hundred videos are being scored rather than the millions scored in candidate generation.

具体一点，从左至右的特征依次是

- impression video ID embedding: 当前要计算的 video 的 embedding
- watched video IDs average embedding: 用户观看过的最后 N 个视频 embedding 的 average pooling
- language embedding: 用户语言的 embedding 和当前视频语言的 embedding
- time since last watch: 自上次观看同 channel 视频的时间
- #previous impressions: 该视频已经被曝光给该用户的次数

上面五个特征中，我想重点谈谈第 4 个和第 5 个。因为这两个很好的引入了对用户行为的观察。

第 4 个特征背后的思想是：We observe that the most important signals are those that describe a user's previous interaction with the item itself and other similar items.

有一些引入 attention 的意思，这里是用了 time since last watch 这个特征来反应用户看同类视频的间隔时间。从用户的角度想一想，假如我们刚看过 “DOTA 经典回顾” 这个 channel 的视频，我们很大概率是会继续看这个 channel 的视频的，那么该特征就很好的捕捉到了这一用户行为。

第 5 个特征 #previous impressions 则一定程度上引入了 exploration 的思想，避免同一个视频持续对同一用户进行无效曝光。尽量增加用户没看过的新视频的曝光可能性。

6.4 工程实践

6.4.1 softmax 的训练

文中把推荐问题转换成多分类问题，在预测 next watch 的场景下，每一个备选 video 都会是一个分类，因此总共的分类有数百万之巨，这在使用 softmax 训练时无疑是低效的，这个问题 YouTube 是如何解决的?

这个问题原文的回答是这样的：We rely on a technique to sample negative classes from the background distribution (”candidate sampling”) and then correct for this sampling via importance weighting.

简单说就是进行了负采样（negative sampling）并用 importance weighting 的方法对采样进行 calibration。文中同样介绍了一种替代方法，hierarchical softmax，但并没有取得更好的效果。当然关于采样的具体技术细节以及优劣可能再开一篇文章都讲不完，感兴趣的同学可以参考 tensorflow 中的介绍（`https://www.tensorflow.org/extras/candidate_sampling.pdf`）以及 NLP 领域的经典论文 `http://www.aclweb.org/anthology/P15-1001`

6.4.2 最近邻搜索

在 candidate generation model 的 serving 过程中，YouTube 为什么不直接采用训练时的 model 进行预测，而是采用了一种最近邻搜索的方法?

这个问题的答案是一个经典的工程和学术做 trade-off 的结果，在 model serving 过程中对几百万个候选集逐一跑一遍模型的时间开销显然太大了，因此在通过 candidate generation model 得到 user 和 video 的 embedding 之后，通过最近邻搜索的方法的效率高很多。我们甚至不用把任何 model inference 的过程搬上服务器，只需要把 user embedding 和 video embedding 存到 redis 或者内存中就好了。

6.4.3 新视频的偏好特征

Youtube 的用户对新视频有偏好，那么在模型构建的过程中如何引入这个 feature?

为了拟合用户对 fresh content 的 bias，模型引入了 “Example Age” 这个 feature，文中其实并没有精确的定义什么是 example age。按照文章的说法猜测的话，会直接把 sample log 距离当前的时间作为

example age。比如 24 小时前的日志，这个 example age 就是 24。在做模型 serving 的时候，不管使用那个 video，会直接把这个 feature 设成 0。大家可以仔细想一下这个做法的细节和动机，非常有意思。

当然我最初的理解是训练中会把 Days since Upload 作为这个 example age，比如虽然是 24 小时前的 log，但是这个 video 已经上传了 90 小时了，那这个 feature value 就是 90。那么在做 inference 的时候，这个 feature 就不会是 0，而是当前时间每个 video 的上传时间了。

我不能 100% 确定文章中描述的是那种做法，大概率是第一种。还请大家踊跃讨论。

文章也验证了，example age 这个 feature 能够很好的把视频的 freshness 的程度对 popularity 的影响引入模型中。

6.4.4 采样训练样本

在对训练集的预处理过程中，YouTube 没有采用原始的用户日志，而是对每个用户提取等数量的训练样本，这是为什么?

原文的解答是这样的：Another key insight that improved live metrics was to generate a fixed number of training examples per user, effectively weighting our users equally in the loss function. This prevented a small cohort of highly active users from dominating the loss. 理由很简单，这是为了减少高度活跃用户对于 loss 的过度影响。

6.4.5 历史的时序特征

YouTube 为什么不采取类似 RNN 的 Sequence model，而是完全摒弃了用户观看历史的时序特征，把用户最近的浏览历史等同看待，这不会损失有效信息吗?

这个原因应该是 YouTube 工程师的“经验之谈”，如果过多考虑时序的影响，用户的推荐结果将过多受最近观看或搜索的一个视频的影响。YouTube 给出一个例子，如果用户刚搜索过“tayer swift”，你就把用户主页的推荐结果大部分变成 tayer swift 有关的视频，这其实是非常差的体验。为了综合考虑之前多次搜索和观看的信息，YouTube 丢掉了时序信息，讲用户近期的历史纪录等同看待。

但 RNN 到底适不适合 next watch 这一场景，其实还有待商榷，目前 youtube 已经上线了以 RNN 为基础的推荐模型，参考论文如下：`https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/46488.pdf`

6.4.6 生成测试集

在处理测试集的时候，YouTube 为什么不采用经典的随机留一法（random holdout），而是一定要把用户最近的一次观看行为作为测试集?

这个问题比较好回答，只留最后一次观看行为做测试集主要是为了避免引入 future information，产生与事实不符的数据穿越。

6.4.7 优化目标

在确定优化目标的时候，YouTube 为什么不采用经典的 CTR，或者播放率（Play Rate），而是采用了每次曝光预期播放时间（expected watch time per impression）作为优化目标?

这个问题从模型角度出发，是因为 watch time 更能反应用户的真实兴趣，从商业模型角度出发，因为 watch time 越长，YouTube 获得的广告收益越多。而且增加用户的 watch time 也更符合一个视频网站的长期利益和用户粘性。

这个问题看似很小，实则非常重要，objective 的设定应该是一个算法模型的根本性问题，而且是算法模型部门跟其他部门接口性的工作，从这个角度说，YouTube 的推荐模型符合其根本的商业模型，非常好的经验。

6.4.8 video embedding

在进行 video embedding 的时候，为什么要直接把大量长尾的 video 直接用 0 向量代替?

这又是一次工程和算法的 trade-off，把大量长尾的 video 截断掉，主要还是为了节省 online serving 中宝贵的内存资源。当然从模型角度讲，低频 video 的 embedding 的准确性不佳是另一个“截断掉也不那么可惜”的理由。

当然，之前很多同学在评论中也提到简单用 0 向量代替并不是一个非常好的选择，那么有什么其他方法，大家可以思考一下。

6.4.9 特征的特殊处理

针对某些特征，比如 #previous impressions，为什么要进行开方和平方处理后，当作三个特征输入模型?

这是很简单有效的工程经验，引入了特征的非线性。从 YouTube 这篇文章的效果反馈来看，提升了其模型的离线准确度。

6.4.10 输出层

为什么 ranking model 不采用经典的 logistic regression 当作输出层，而是采用了 weighted logistic regression?

因为在第 7 问中，我们已经知道模型采用了 expected watch time per impression 作为优化目标，所以如果简单使用 LR 就无法引入正样本的 watch time 信息。因此采用 weighted LR，将 watch time 作为正样本的 weight，在线上 serving 中使用 e(Wx+b) 做预测可以直接得到 expected watch time 的近似，完美。

6.5 详细解读输出层采用 weighted logistic regression

对于传统的深度学习架构，输出层往往采用 LR 或者 Softmax，在线上预测过程中，也是原封不动的照搬 LR 或者 softmax 的经典形式来计算点击率（广义地说，应该是正样本概率）。

而 YouTube 这一模型的神奇之处在于，输出层没有使用 LR，而是采用了 Weighted LR，模型 serving 没有采用 sigmoid 函数的形式，而是使用了 e^{Wx+b} 这一指数形式。按照原文说法，这样做预测的就是用户观看时长??

搞清楚这件事情并不是一件容易的事情，我们要从逻辑回归的本质意义上开始。

几乎所有算法工程师的第一堂课就是逻辑回归，也肯定知道逻辑回归的数学形式就是一个线性回归套 sigmoid 函数：

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

但为什么选择 sigmoid 函数？难道仅仅是 sigmoid 函数能把值域映射到 0-1 之间，符合概率的物理意义这么简单吗？

为解释这个问题，首先我们需要定义一个新的变量——Odds，中文可以叫发生比或者机会比。

$$Odds = \frac{p}{1-p}$$

假设一件事情发生的概率是 p，那么 **Odds 就是一件事情发生和不发生的比值**。

如果对 Odds 取自然对数，再让 ln(Odds) 等于一个线性回归函数，那么就得到了下面的等式。

$$logit(p) = \ln(\frac{p}{1-p}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

其中 $\ln(p/(1-p))$ 就是大名鼎鼎的 logit 函数，logistics regression 又名 logit regression，上面的式子就是逻辑回归的由来。我们再做进一步运算，就可以转变成我们熟悉的逻辑回归的形式：

$$\ln(\frac{p}{1-p}) = \theta^T x \Rightarrow \frac{p}{1-p} = e^{\theta^T x} \Rightarrow p = \frac{1}{1+e^{-\theta^T x}} \Rightarrow p = sigmoid(\theta^T x)$$

到这里大家应该已经完全明白了 LR 的推导过程了。

那么再对 $\ln(Odds) = \theta^T x$ 这个等式做一个小小的转换，两边取自然底数：

$$\ln(Odds) = \theta^T x \Rightarrow Odds = e^{\theta^T x} = YouTubeServingFunction$$

大家看到了吗，**Youtube 的 Serving 函数 e^{Wx+b} 计算的不是别的，正是 Odds**！

但我们还没有到达终点，因为 Youtube 要预测的明明是用户观看时长，怎么就成了 Odds 了？

这就要提到 YouTube 采用的独特的训练方式 Weighted LR，这里的 Weight，对于正样本 i 来说就是观看时长 T_i，对于负样本来说，则指定了单位权重 1。

Weighted LR 的特点是，正样本权重 w 的加入会让正样本发生的几率变成原来的 w 倍，也就是说样本 i 的 Odds 变成了下面的式子：

$$Odds(i) = \frac{w_i p}{1 - w_i p}$$

由于在视频推荐场景中，用户打开一个视频的概率 p 往往是一个很小的值，因此上式可以继续简化：

$$Odds(i) = \frac{w_i p}{1 - w_i p} \approx w_i p = T_i p = E(T_i)$$

而且由于 YouTube 采用了用户观看时长 T_i 作为权重，因此式子进一步等于 $T_i * p$，这里真相就大白了，由于 p 就是用户打开视频的概率，T_i 是观看时长，因此 $T_i * p$ 就是用户观看某视频的期望时长！

因此，YouTube 采用 e^{Wx+b} 这一指数形式预测的就是曝光这个视频时，用户观看这个视频的时长的期望！利用该指标排序后再进行推荐，是完全符合 YouTube 的推荐场景和以观看时长为优化目标的设定的。

再简要总结一下 YouTube Ranking Model 的 Serving 过程要点

(1) e^{Wx+b} 这一指数形式计算的是 Weighted LR 的 Odds；

(2) Weighted LR 使用用户观看时长作为权重，使得对应的 Odds 表示的就是用户观看时长的期望；

(3) Model Serving 过程中 e^{Wx+b} 计算的正是观看时长的期望。

7 阿里深度兴趣网络（DIN）

7.1 用户场景

用户场景很简单，就是在一个电商网站或 APP 中给用户推荐广告，当然对于阿里妈妈来说，广告也是商品，所以这篇文章的广告场景其实也是一个经典的推荐场景。

好，既然要推荐，我们当然需要利用用户的历史数据了，对于一个电商来说，历史数据当然就是点击，添加购物车，下单这些行为了。论文中给了一位用户的行为序列。

用户的行为序列

Figure 5: 用户的行为序列

显然是一个女生的行为历史啦，从最左边的手套，鞋子到右边的杯子，睡衣。要被推荐的候选商品是一件女式大衣。我们应该如何计算这件大衣的 CTR 呢?

如果按照之前的做法，我们会一碗水端平的考虑所有行为记录的影响，对应到模型中就是我们会用一个 average pooling 层把用户交互过的所有商品的 embedding vector 平均一下形成这个用户的 user vector，机灵一点的工程师最多加一个 time decay，让最近的行为产生的影响大一些，那就是在做 average pooling 的时候按时间调整一下权重。

但是我们仔细想一想我们自己的购买过程，其实每个用户的兴趣都是多样的，女生喜欢买衣服包包，也喜欢化妆品，甚至还为自己男朋友挑选过球衣球鞋，那么你在买大衣的时候，真的要把给男朋友买球鞋的偏好考虑进来么？具体到本文的例子中，在预测大衣的 CTR 这件事情上，用户浏览过杯子，跟用户浏览过另一件大衣这两个行为的重要程度是一样的吗?

这事不用问算法工程师，你就回家问问你老妈估计答案都是一定的，肯定是浏览过另一件大衣这件事的参考价值高啊。好了，就是这件你老妈都知道的事情，让阿里妈妈的算法工程师们加上了 attention 机制。

7.2 注意力机制

注意力机制顾名思义，就是模型在预测的时候，对用户不同行为的注意力是不一样的，“相关”的行为历史看重一些，“不相关”的历史甚至可以忽略。那么这样的思想反应到模型中也是直观的。

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

其中，V_u 是用户的 embedding 向量，V_a 是候选广告商品的 embedding 向量，V_i 是用户 u 的第 i 次

行为的 embedding 向量，因为这里用户的行为就是浏览商品或店铺，所以行为的 embedding 的向量就是那次浏览的商品或店铺的 embedding 向量。

因为加入了注意力机制，V_u 从过去 V_i 的**加和**变成了 V_u 的**加权和**，V_i 的权重 w_i 就由 V_i 与 V_a 的关系决定，也就是上式中的 $g(V_i, V_a)$。

那么 $g(V_i, V_a)$ 这个函数到底采用什么比较好呢？看完下面的架构图自然就清楚了。

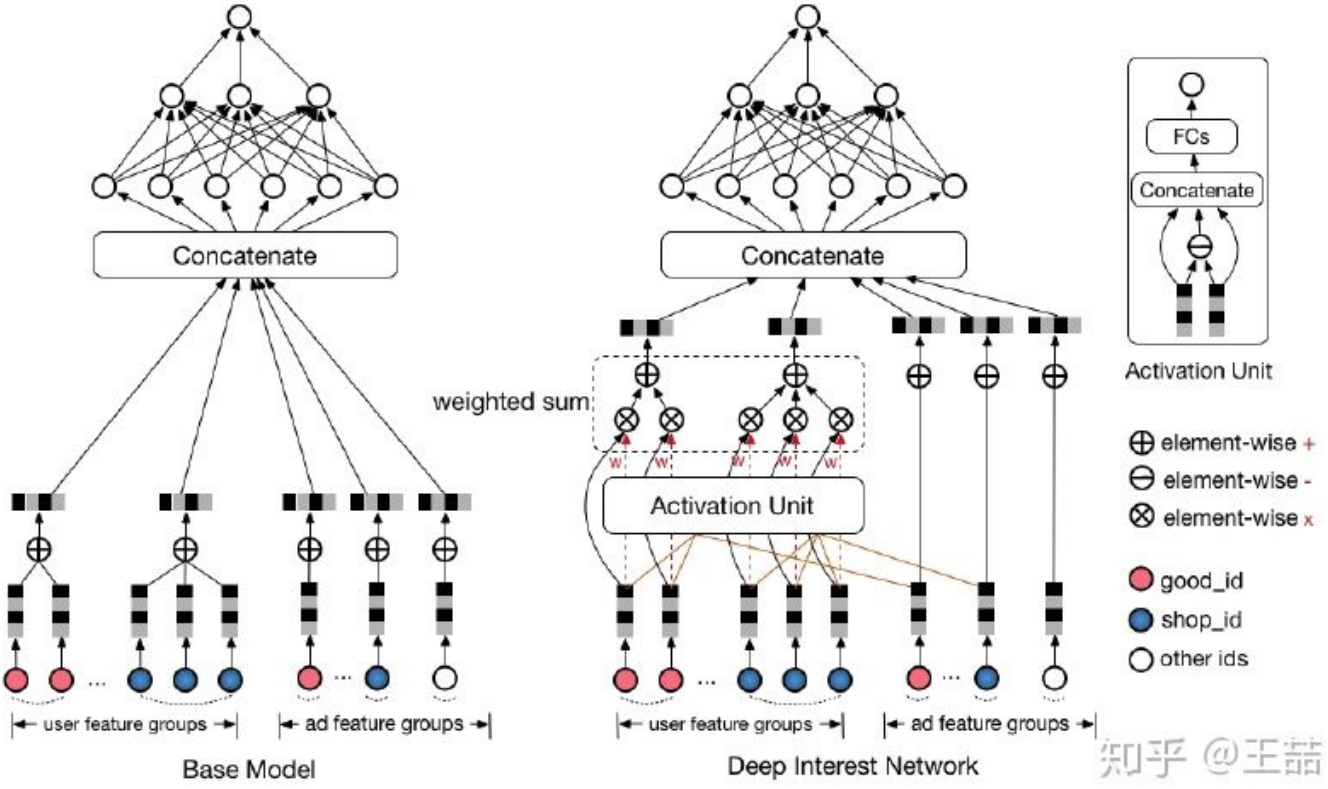


Figure 6: 用户的行为序列

相比原来这个标准的深度推荐网络（Base model），DIN 在生成用户 embedding vector 的时候加入了一个 activation unit 层，这一层产生了每个用户行为 V_i 的权重，下面我们仔细看一下这个权重是怎么生成的，也就是 $g(V_i, V_a)$ 是如何定义的。

传统的 Attention 机制中，给定两个 item embedding，比如 u 和 v，通常是直接做点积 uv 或者 uWv，其中 W 是一个 $|u| \times |v|$ 的权重矩阵，但这篇 paper 中阿里显然做了更进一步的改进，着重看上图右上角的 activation unit，**首先是把 u 和 v 以及 uv 的 element wise 差值向量合并起来作为输入，然后喂给全连接层，最后得出权重**，这样的方法显然损失的信息更少。但如果你自己想方便的引入 attention 机制的话，不妨先从点积的方法做起尝试一下，因为这样连训练都不用训练。

再稍微留意一下这个架构图中的红线，你会发现每个 ad 会有 good_id, shop_id 两层属性，shop_id 只跟用户历史中的 shop_id 序列发生作用，good_id 只跟用户的 good_id 序列发生作用，这样做的原因也是显而易见的。

7.3 论文作者的评论

其实最初的时候我们不是按照借鉴 attention 的套路来的，当时想法就是希望做个反向激活的网络结构，能自适应地挑跟 candidate 相关的用户行为来做用户兴趣的表征，后来写论文的时候仔细 survey 发现这个是 attention 的解法。再透露个细节，其实在 DL 模型之前，我们就研制了一种可以认为是 hard 模式的 attention 结构：那是特征组合的解法，我们用 ad 的 shop 属性去 hit 用户历史行为过的 shop list，如果命中了那么就说明历史用户有过直接行为，用行为 id 和频次来表示这个组合特征；如果没有命中特征就是空的，那显然 attention 结构就是一种自然的 soft 扩展，不过适用范围更强了

References

[1] 王🄵, 深度学习推荐系统.