gbdt 与 dnn 新 ctr 预估模型算法技术报告

陈世熹

一、简介

一直以来，百度凤巢 pc 的广告 ctr 预估模型都是离散 lr（逻辑回归）模型。离散 lr 模型具有特征可扩展性强、算法简单、效果稳定、模型易分析且可解释性强等优点，因此大量的特征和算法调研工作都投入到了离散 lr 之中。但是，离散 lr 也具有一些内在的缺点，包括

1、 在不改变特征的情况下，离散 lr 是一个线性模型，表达能力非常有限；
2、 目前离散特征数量巨大，已达到近千亿，通过增加特征组合的方式来提高模型表达能力的方式已经愈发困难。

为了提高 ctr 预估模型的表达能力，在最近一年，凤巢 pc 开始调研建立在连续值特征之上的模型。连续值模型的目标是把特征维度大幅降低，减小模型尺寸，并且使用非线性模型提高在长尾特征上的泛化能力。

预估模型的基本要素包含两个方面：一是我们对待估样本知道些什么，并且如何描述和表达我们的知识；二是我们所知道的东西如何影响预估结果。简单来说就是基础输入信息以及预估函数。很多时候，我们可能没法直接利用样本的输入信息，而需要对原始信息进行加工和改造，获得更加易于使用的信息。在离散 lr 中，我们首先通过 adfea 从原始视图日志中抽取离散特征，然后再在基于离散特征表示的数据上训练 lr 模型。而深度学习的思想其实就是对数据的特征表达进行多层的加工处理，得到抽象的特征表达。

目前，凤巢 pc 的连续值特征框架是建立在离散特征之上的，它是对数据的离散特征表达的进一步加工。具体而言，连续值特征具有跟离散特征相同的槽位结构，每个槽位的连续值特征对应相应槽位的离散特征的某些连续值统计量。目前最常用的统计量包括：

1、 离散特征在历史数据时间窗中的 pv（展现数）和 clk（点击数），会进行一些变换处理，如取 log 以及计算 ctr。如果考虑到不同展现位置（cmatch-rank）的天然点击率偏差并据此进行修正，则称为 coec 特征。
2、 从离散 lr 模型中获得的相应离散特征的 weight。

连续值特征的维度是固定的。如果一个样本在某个槽位上恰好命中了一个离散特征，那么连续值特征就是这个离散特征的相应统计量。如果没有命中，则取默认值，通常为 0。如果命中了多个，则取平均值或加权平均值。

在得到了样本的连续值特征表达之后，我们需要建立适用于连续值特征的预估模型。现有的模型包括 gbdt、dnn、clr（连续值 lr）、cbpr 和 ann 等，其中效果最好的是 gbdt、dnn 和 ann，而 ann 又是 dnn 的单隐层特例。这篇技术报告将会简单地介绍 gbdt 和 dnn 训练算法实现，由于本文并不是教程，并不会对具体的细节进行阐述。

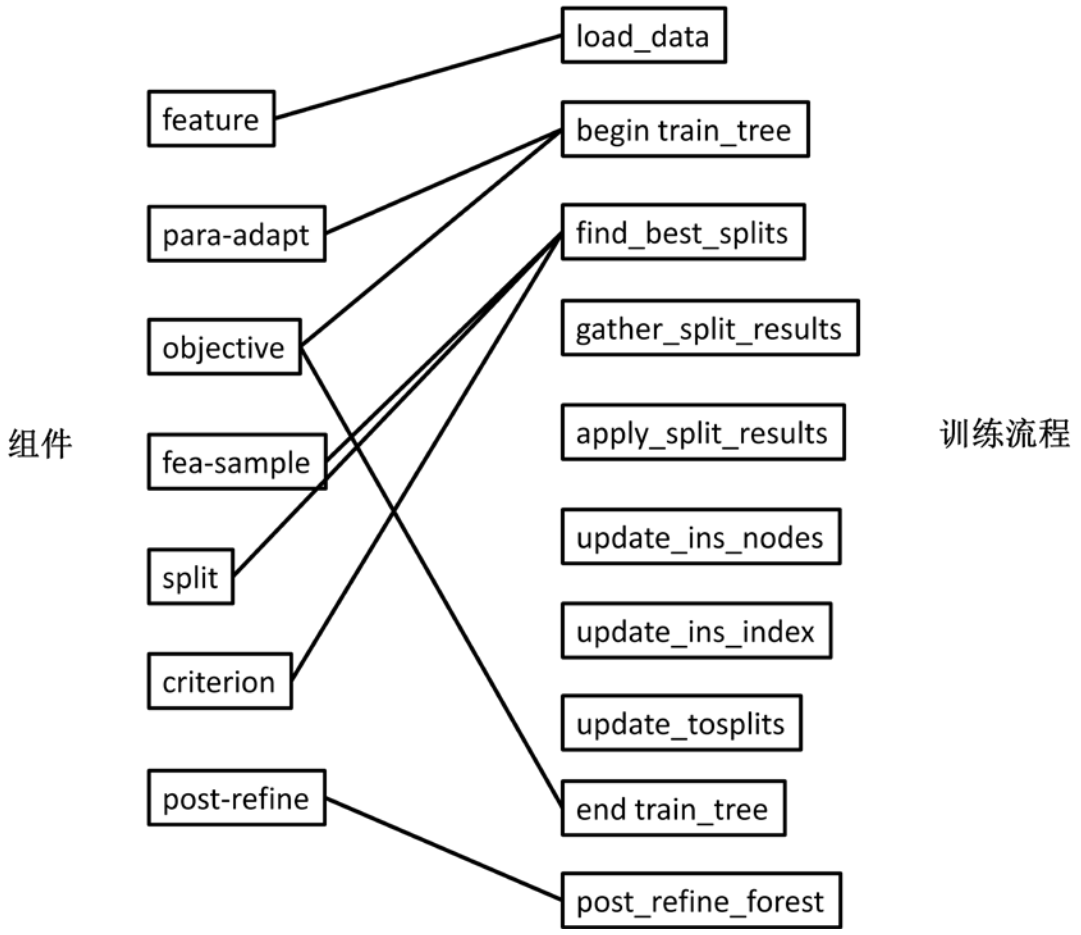
二、gbdt 训练算法实现

gbdt（Gradient Boost Decision Tree）是基于 gradient boost 和决策树的预估模型，它的基本原理是：

1、 对全局目标函数（通常是 MSE 或 logistic loss）通过梯度下降法进行迭代优化，计算和维护每个样本的局部梯度，每轮迭代求解一个子模型来拟合所有样本的局部梯度；
2、 子模型使用决策树模型。

由于单机版 gbdt 训练程序的计算效率和所支持的数据量问题，需要使用 mpi 实现并行化。Mpi gbdt 由以下两个部分组成：
1、 组件：包含许多不同部分的训练策略和参数调整策略的实现，不需要关心 mpi 并行化和多线程的细节，可以通过实现子类来进行扩展，支持灵活配置；
2、 训练框架：负责整体的并行化训练算法、多线程调度和数据管理等。通过调用组件的接口来完成学习策略。

下图简单地给出了组件和训练框架的若干部分



Mpi 训练按照每个 mpi 节点负责一个或若干个特征的方式来进行，因此属于特征级的并行。一般 gbdt 训练数据有 80 到 110 个特征，因此可以使用上百个 mpi 节点来进行并行化训练。但由于一个特征的计算不能跨节点进行，因此一个 mpi 节点必须能够在内存中存储一个特征所需的所有数据，这限制了所支持的训练数据规模。在 64GB 内存的机器上，所支持的最大数据量约为 12 亿条样本左右。为了支持更大规模的训练数据，mpi gbdt 实现了采样策略，每棵决策树的训练数据从所有训练数据中进行采样，从而实现了对上百亿训练数据的支持。

训练框架部分所有耗时的阶段都进行了多线程优化，组件的代码实现是不需要关心与 mpi

和多线程有关的细节的。这使组件的代码实现十分简单，便于调研和扩展。

三、dnn 训练算法实现

dnn（deep neural network）是另一种基于连续值特征的非线性模型，它通过多层的神经网络来拟合目标函数。本章将简单介绍 homura 训练程序。

homura 是基于随机梯度下降方法（SGD）的 dnn 训练程序，它每次获取一个 minibatch 的训练数据，在这一小部分的训练数据上计算参数的梯度并更新参数。在训练开始前，它会对训练数据的顺序进行随机扰乱，并且进行 normalization。在训练中，它使用 mpi 进行并行化计算，每个 mpi 节点内使用 openBLAS 或 intel MKL 进行矩阵运算。

Homura 的实现是基于 Layer 的，大部分的流程都被封装到一个 Layer 的子类中。Layer 与 Layer 之间通过 Output 实例来联系，Output 表示计算流程中的输入输出或中间数据，例如每一层的参数矩阵，神经元的输出向量等。这使实现得程序具有比较强的可扩展性。下面列出了 homura 最重要的一些 Layer 和相关类：

TextInputLayer	文本输入层
TextPartitioningLayer	文本随机划分层
TextShufflingLayer	文本随机顺序扰乱层
DataParsingLayer	负责解释文本格式的输入数据
NormalizingLayer	负责输入数据的 normalization。
MatrixInputLayer	二进制数据输入层
MatrixOutputLayer	二进制数据输出层
LossLayer	负责计算损失函数。
LossFunction	负责损失函数的具体计算实现，具有子类 LogisticLossFunction。
ActivationLayer	负责计算激活函数
ActivationFunction	负责激活函数的具体计算实现，具有子类 SigmoidActivationFunction、TanhActivationFunction 、 RectifiedLinearActivationFunction 、 SoftPlusActivationFunction。
SGDLayer	负责 dnn 参数的更新，需要处理 mpi 并行化。
SGDUpdateRule	负责 SGD 参数更新公式的实现，具有子类 AdaGradSGDUpdateRule、AdaDeltaSGDUpdateRule 等。
LinearLayer	负责矩阵加法的层。
MultiplicationLayer	负责矩阵乘法的层。

3.1、分布式计算

Dnn 分布式计算需要考虑具体的分布式策略，主要包括数据分布式和模型分布式。在数据分布式中，在每次 minibatch 的计算中每个节点会负责一部分样本的计算，然后把梯度汇总起来进行参数的更新（模型参数的更新也是分布式的），然后再把更新后的参数广播到所有节点上。而在模型分布式中，每个节点会负责模型某部分的计算任务，但会涉及到所有的训练数据，具体的模型和计算任务的划分方式也不固定。具体选择哪种方式需要考虑若干因素：1、 训练数据是稀疏输入还是稠密输入，模型有多大：如果是非常稀疏的数据，可能单个节

点内存无法存储整个模型，必须使用分布式模型存储，此时模型分布式计算可能比较适合；但如果特征的频率分布极其 skew，模型分布式计算反而有可能导致计算任务分配不均衡，则需要模型分布式存储的数据分布式计算。

2、 使用 sgd 优化算法还是 batch 优化算法：如果是 batch 优化，数据分布式的计算开销和通讯开销通常更低一些，而且计算任务的分配更容易保持均衡。

3、 分布式平台是 hadoop、mpi 还是 gpu 集群，通讯开销有多大：如果通讯开销很大，就不得不考虑异步 sgd 并且尽量减少通讯频率，此时数据分布式更为适合。

一般来说，数据分布式计算具有以下特点：

1、 每次 minibatch 的计算只需要两次通讯来分发梯度和参数，通讯频率较低；

2、 通讯数据量主要跟模型大小有关，因此在 minibatch 足够大或模型足够小时可以忽略通讯数据量；

3、 分发梯度和参数的部分可以集中处理和优化，每一层的具体计算可以完全不涉及分布式的细节，实现较为简单，可扩展性强；

4、 适合使用异步 sgd 来避免同步开销。

而模型分布式计算则具有以下特点：

1、 每一层在前馈和向后传播的阶段都各需要一次通讯，通讯频率比较高；

2、 Minibatch 越小则通讯数据量越少，因此在模型足够大时可以忽略通讯数据量；

3、 每一层的计算都需要进行分布式通讯，对不同的连接层实现需要分别实现分布式的处理，较为复杂，可扩展性差；

4、 不太适合使用异步 sgd。

Homura 使用了数据分布式的同步 sgd 算法实现，以达到较好的可扩展性并降低网络通讯开销。

下面是对 homura 计算性能的一些实验结果。我们使用连续值数据（共 220 维）进行训练，网络结构为 220:1024:256:128:128:128:1，每个节点的 minibatch 大小为 1000，其在不同 mpi 节点下的计算速度如下（由于 mpi 集群网络状况不稳定，统计时间会有一些波动）：

szwg mpi 节点数	每秒处理的平均训练样本数	每天可处理的训练样本数
1	15166	13.1 亿
10	72377	62.5 亿
20	250965	216.8 亿
50	533312	460.8 亿
100	953146	823.5 亿

下表是单个 mpi 节点在不同 minibatch 大小时的计算速度：

minibatch 大小	每秒处理的平均训练样本数
100	2934
200	5203
500	8146
1000	15166
2000	16282
5000	13530

大致上，基于 cpu 的 homura 在单个 mpi 节点上的数据处理速度约为使用单个 gpu 卡的 paddle（双 buffer 优化版本）的 10%左右。由于同步的开销，大约需要 65 个 mpi 节点可以达到使用 4 个 gpu 卡的 paddle（双 buffer 优化版本）的处理速度。后续会考虑采用异步 sgd 来进行优化。

3.2、参数更新算法和激活函数

Homura 中实现了若干参数自适应更新算法，其中效果最好的是 AdaGrad 和 AdaDelta。这些算法都基于参数历史梯度的大小来自动调整参数的学习率。

AdaGrad 的学习率公式为：

$$\eta_t = \frac{\eta_0 \cdot \sqrt{i}}{\sqrt{i + \sum_{s=1}^{t} g_s^2}}$$

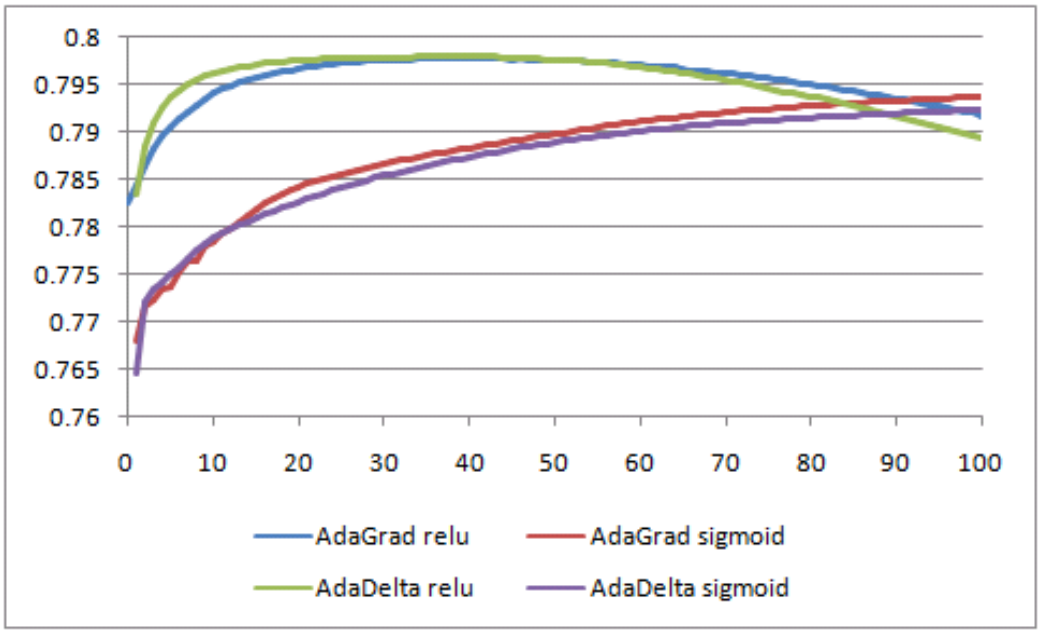
而 AdaDelta 的学习率公式为：

$$\begin{aligned}
\overline{g_i}(t+1) &= (1-\tau_i^{-1}) \cdot \overline{g_i}(t) + \tau_i^{-1} \cdot \nabla_{\theta_i(t)} \\
\overline{v_i}(t+1) &= (1-\tau_i^{-1}) \cdot \overline{v_i}(t) + \tau_i^{-1} \cdot (\nabla_{\theta_i(t)})^2 \\
\overline{h_i}(t+1) &= (1-\tau_i^{-1}) \cdot \overline{h_i}(t) + \tau_i^{-1} \cdot h_i^{(t)} \\
\eta_i^* &\leftarrow \frac{(\overline{g_i})^2}{\overline{h_i} \cdot \overline{v_i}} \\
\tau_i(t+1) &= \left(1 - \frac{\overline{g_i}(t)^2}{\overline{v_i}(t)}\right) \cdot \tau_i(t) + 1
\end{aligned}$$

对于激活函数，homura 实现了 sigmoid、tanh、relu 和 softplus 等。其中 sigmoid 和 tanh 是传统的激活函数，而 relu 和 softplus 是最近深度学习研究中兴起的激活函数。Relu 激活函数可以在无需逐层预训练的情况达到可跟 sigmoid 相同的训练效果，且学习速度比较高，因此是最近 dnn 学习算法中的首选激活函数。下表给出了各种不同激活函数的计算公式：

sigmoid	1/(1+exp(-x))
tanh	2/(1+exp(-2*x))-1
relu	max(0,x)
softplus	log(1+exp(x))

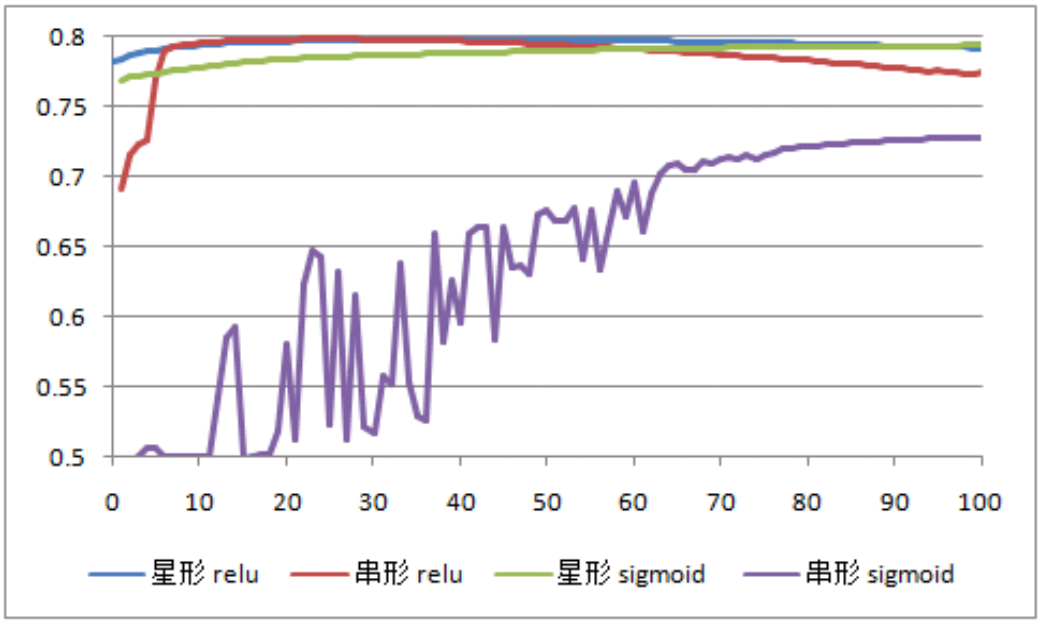
我们对不同的参数更新算法和激活函数进行了调研，下图是在凤巢 pc 无偏数据的连续值特征上的 dnn 训练调研结果（使用星形连接结构）



其中横坐标表示迭代轮数（整个训练数据集的扫描次数），纵坐标为 auc 指标。可以看出 relu 相比 sigmoid 激活函数具有更快的学习速度，而 AdaDelta 算法也稍优于 AdaGrad 算法。在实践中，AdaGrad 要达到较好的学习速度往往可能需要调整学习率参数，否则有时会达到很差的训练效果，而 AdaDelta 的表现则比较稳定，不太需要调整学习率参数。

3.3、星形连接

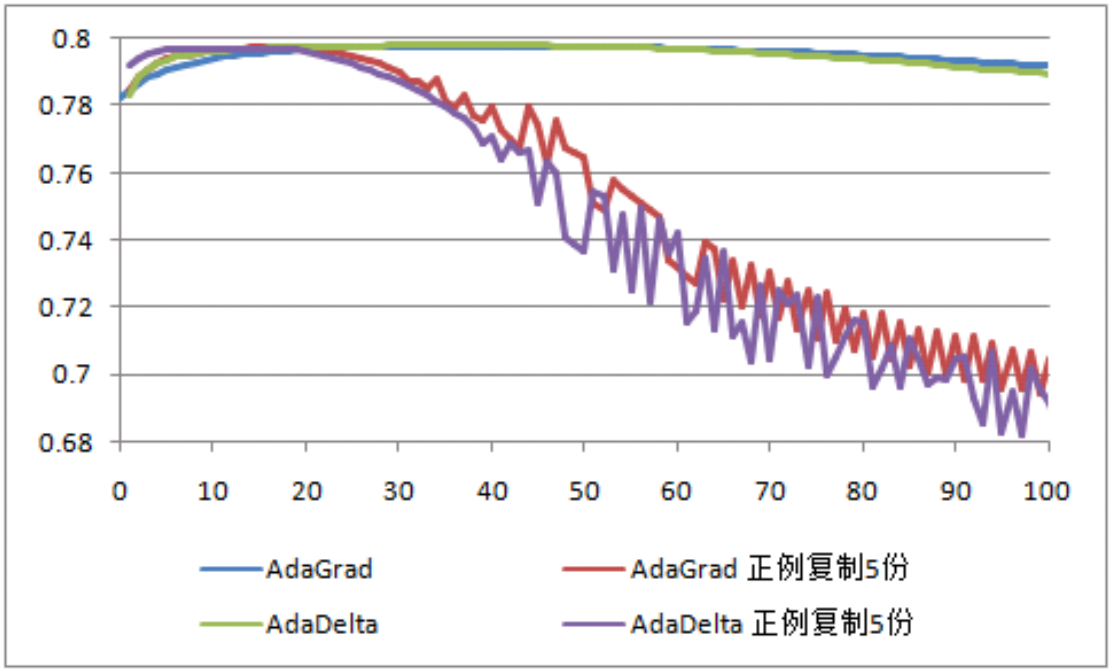
传统的 dnn 连接方式是串联的，也就是说，每一层都只跟它的上层和下层连接。在 homura 中，由于没有实现逐层预训练的策略，通常会使用更加适合无需预训练的 relu 激活函数来进行训练，但还可以采样一种特殊的连接方式，称为星形连接。在星形连接中，在串形连接的基础上，它还把输出层更前面的所有层相连，使得低层可以直接从输出层获得梯度传播，从而在没有预训练的情况下也可以较好地训练低层网络。下图给出了星形连接和串形连接在连续值特征数据上的调研结果（使用 AdaGrad）。



可以看出，星形连接比串形连接具有更快的学习速度，它对 sigmoid 激活函数的效果最为明显。

3.4、样本复制、采样策略

在 homura 训练时，我们通常还会采样一种样本复制、采样的策略，那就是把正例的样本数复制为 5 倍，或者把负例采样成 20%。这种策略可以在正例数据量远少于负例的情况下使得在每个 minibatch 中正例和负例的数量更为均衡，显著降低梯度的方差。注意这跟简单地把正例的权重乘 5 或把负例的权重乘 0.2 是不一样的，因为这样每个 minibatch 内的正负例比例还是不均衡，并不能够降低梯度的方差。下图给出了正例复制 5 倍的调研结果：



可以看出把正例复制 5 倍的策略基本上能够把学习速度提高 5 倍。后面的 auc 下降主要是因为过拟合，但 auc 达到的最高点是一致的。

3.5、网络大小和层数

下表中给出了在 ann 数据上（包含了 coec 连续值特征以及 lr weight 等其它特征）使用不同网络结构的训练结果

网络结构	AUC
799:1024:256:128:128:128:1	0.796375
799:1024:1	0.795545
799:512:1	0.795566
799:256:1	0.795105
799:128:1	0.794480
799:64:1	0.793148

可以看出 dnn 网络的规模对于连续值特征数据并没有太大影响，实践中可能使用小的 dnn 网络更好，可以显著减少线下训练以及线上预估的计算开销。

四、总结和展望

Gbdt 和 dnn 等非线性模型相比线性 lr 模型可以显著地提高模型的表达能力，提高 ctr 预估质量。然而，目前所用的连续值特征只是对离散特征的简单统计量，转换成连续值特征表达的数据已经完全丢失了其原有的物理信息。例如，对于具有相似的 lr weight、pv 和 clk 的两个离散特征，必然具有相似的连续值表达，它们对连续值特征模型预估函数的作用也是相似的，连续值特征模型将无法从这两个离散特征的物理意义的不同把它们区分开来。因此，目前的连续值特征模型更像是一种负责多信号后处理的 ensemble 方法，并不能够提高特征的内在表达能力。

由于目前连续值特征丢失了离散特征原有的物理含义，无法从连续值特征上学习出有用的抽象特征，在连续值特征上进行深度学习可能是没有太大意义的。事实上，调研显示 5 个隐层的神经网络和单隐层的神经网络在预估效果上并没有太大差别，差距仅在 1 个千分点以内，正是验证了这一点。目前建立在连续值特征上的非线性模型并没有进行真正的深度学习，而仅仅是利用了全局统计量之间的相关性更好地拟合了目标函数。

为了提高连续值特征的表达能力，为训练样本提供更加丰富的物理信息，我们可以尝试对每个离散特征学习出一个低维向量表达（或高维稀疏向量表达），用一个向量来刻画每个离散特征的内在性质以及它与其它离散特征的相互作用。譬如，对于用户 id 类或广告 id 类的离散特征，我们用一个向量来表达用户的兴趣或广告的类型。建立在这样的向量表达之上的 dnn 模型能够利用更多有意义的信息来提高模型的表达能力。这种方式跟深度学习所倡导的从基础特征逐步抽取高阶抽象特征的思想一致。

这样的向量表达可以从两个方面来提高 ctr 预估效果：一是它使得相似的离散特征具有相似的向量表达，利用对象之间的相似性来提高泛化能力；二是它让不同的离散特征可以通过非线性的方式来互相作用，提高模型的表达能力。离散 lr 则一般通过增加基础特征的方式来实现基于相似性的泛化能力，通过增加组合特征的方式来实现不同基础特征之间的非线性相互作用。但是，通过增加离散特征的方式来实现这两个方面的改进存在一些缺点：

1、 对象（如用户、广告、查询）之间的相似性必须通过获取明确的离散特征信息（如地域、分词、关键词）来实现，而不能从数据中自动学习到。另外，这种方式通常达到的是对离散特征进行硬分类的效果，而不是模糊聚类，不能很好地捕捉和利用不同类别之间的相关性。
2、 组合特征会使离散特征数量指数式的膨胀，导致计算开销大、容易过拟合、泛化能力极差等问题，增加组合特征后的 lr 模型的训练样本复杂度也是指数增长的，因此不可能增加太多的组合特征。对于多值特征（一个样本会同时命中多个离散特征的槽位），组合特征还会使样本的命中特征数指数增长。

而基于向量表达的途径则可以填补离散 lr 的这些缺点。

要构造出离散特征的向量表达并不容易，可行的方案包括离散 dnn（有监督 bp、auto-encoder、rbm？）、基于矩阵分解的协同过滤、分解机器等。希望在未来可以对此进行探索。