表征学习

1. 华为 | 一种CTR预估中连续特征的Embedding学习框架

1.1 背景

当前大多数的研究主要集中在设计更复杂的网络架构来更好的捕获显式或隐式的特征交互。而另一个主要的部分，即Embedding模块同样十分重要，出于以下两个原因：

1）Embedding模块是FI模块的**上游**模块，直接影响FI模块的效果； 2）CTR模型中的大多数参数集中在Embedding模块(巨大的embedding table！)，对于模型效果有十分重要的影响。

但是，Embedding模块却很少有工作进行深入研究，特别是对于连续特征的embedding方面。现有的处理方式由于其硬离散化(hard discretization)的方式，通常suffer from low model capacity。而本文提出的AutoDis框架具有high model capacity, end-to-end training（而直接分桶的方式显然是不能端到端训练的，通常怎么分桶也是人"拍脑袋"决定的）, 以及unique representation（而直接分桶的方式很多不同的值会被分到同一个桶中，共享embedding 表达）.

- No Embedding：是指不对连续特征进行embedding操作，而直接使用原始的数值。如Google Play的Wide & Deep直接使用原始值作为输入;而在Youtube DNN 中，则是对原始值进行变换（如平方，开根号）后输入：

$$\mathbf{e}_{YouTube} = [\tilde{x}_1^2, \tilde{x}_1, \sqrt{\tilde{x}_1}, \tilde{x}_2^2, \tilde{x}_2, \sqrt{\tilde{x}_2}, \ldots, \tilde{x}_N^2, \tilde{x}_N, \sqrt{\tilde{x}_N}], \quad (3)$$

这类对连续特征不进行embedding的方法，由于模型**容量有限**（参数太少），通常难以有效捕获连续特征中信息。

- Field Embedding：是指同一个field无论取何值，都共享同一个embedding，随后将特征值与其对应的embedding相乘作为模型输入：

$$\mathbf{e}_{FE} = [x_1 \cdot \mathbf{e}_1, x_2 \cdot \mathbf{e}_2, \ldots, x_N \cdot \mathbf{e}_N], \quad (5)$$

其中，e1,e2,...en 是field embedding。

由于同一field的特征共享同一个embedding，并基于不同的取值对embedding进行缩放，这类方法的表达能力也是有限的。

- Discretization: 即将连续特征进行离散化，是工业界最常用的方法。这类方法通常是两阶段的，即首先将连续特征转换为对应的离散值，再通过look-up的方式转换为对应的embedding。embedding table为：$E_j \in R^{H_j \times d}$, 其中H_j为离散化分桶的桶数，d为embedding size。

【为什么要将连续特征离散化呢？】将连续特征进行离散化给模型引入了**非线性**(可以做特征交叉)，能够提升模型表达能力。同时，对于一个field，我们有$H_j \times d$个embedding参数，提升了模型的capacity。而对于离散化的方式，常用的有以下几种：

1） 等宽/等深分箱。对于等宽分箱，首先基于特征的最大值和最小值、以及要划分的桶的个数 H_j，来计算每个样本取值要放到哪个箱子里。对于等深分箱，则是基于数据中特征的**频次**进行分桶，每个桶内特征取值的个数是大致相同的。（注：等宽/等深分箱也是工业界最常使用的方法，但是其问题在于，我们必须知道数据的分布才能够进行分箱。对于等宽分箱还好，只需要知道数据的max/min，但是对于等深分箱，则需要统计数据的详细分布。然而，在线上数据是一个stream，是无法离线统计好全部数据分布的。）

2）LD (Logarithm Discretization)：对数离散化，其计算公式如下：

$$\widehat{x}_j = d_j^{LD}(x_j) = floor(log(x_j)^2). \tag{8}$$

3）TD (Tree-based Discretization)：基于树模型的离散化，如使用GBDT+LR来将连续特征分到不同的节点。这就完成了离散化。

离散化方法的缺点： 1）**TPP (Two-Phase Problem)**：将特征分桶的过程一般使用启发式的规则（如EDD、EFD）或者其他模型（如GBDT），无法与CTR模型进行一起优化，即**无法做到端到端训练**； 2）SBD (Similar value But Dis-similar embedding)：对于边界值，两个相近的取值由于被分到了不同的桶中，导致其embedding可能相差很远； 3）DBS (Dis-similar value But Same embedding)：对于同一个桶中的边界值，两边的取值可能相差很远，但由于在同一桶中，其对应的embedding是完全相同的。

1.2 AutoDis介绍

AutoDis的全称为Automatic end-to-end embedding learning framework for numerical features based on soft discretization. 用于连续特征的**端到端**离散化和embedding学习。

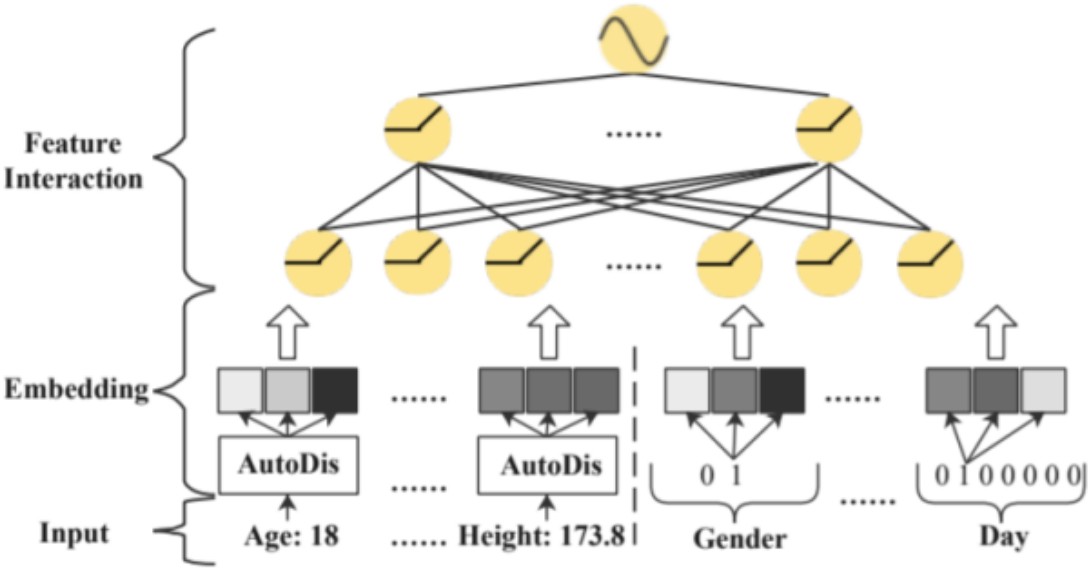


Figure 3: AutoDis works as a numerical feature embedding learning framework compatible to the existing deep CTR models.

1.3.1 Meta-Embeddings

为了提升model capacity，一种朴素的处理连续特征的方式是给每一个特征取值赋予一个独立的embedding。显然，这种方法参数量巨大（因为你可以有无穷个连续特征取值！），无法在实践中进行使用。另一方面，Field Embedding对同一field内的特征赋予都相同的embedding，降低了参数数量，但model capacity也受到了一定的限制。

为了平衡参数数量和模型容量，AutoDis设计了Meta-embedding模块: **对于第 j 个连续特征，对应 H_j 个Meta-Embedding**（可以看作是分 H_j 个**桶**，每一个桶对应一个embedding）。第j个特征的Meta-Embedding表示为：

$ME_j \in \mathbb{R}^{H_j \times d}$（$H_j$个桶，每个桶是d维的）

对于连续特征的一个具体取值，则是通过一定方式将这 H_j 个embedding进行**聚合**。相较于Field Embedding这种每个field只对应一个embedding的方法，AutoDis中每一个field对应 H_j 个embedding，提升了模型容量；同时，参数数量也可以通过 H_j 进行很好的控制。//。

1.3.2 Automatic Discretization

Automatic Discretization模块可以对连续特征进行自动的离散化，实现了离散化过程的端到端训练。具体来说，对于第 j 个连续特征的具体取值 x_j ，首先通过**两层神经网络**进行转换，得到 H_j 长度的向量（第一层是1维->10维，第二层是10维->10维）。下图的例子假设有41个特征，每个特征分配 $H_j = 10$ 个桶：

$$\mathbf{h}_j = \text{Leaky\_ReLU}(\mathbf{w}_j x_j), \tag{11}$$
$$\widetilde{\mathbf{x}}_j = \mathbf{W}_j \mathbf{h}_j + \alpha \mathbf{h}_j,$$

where $\mathbf{w}_j \in \mathbb{R}^{1 \times H_j}$ and $\mathbf{W}_j \in \mathbb{R}^{H_j \times H_j}$ are the learnable parameters of the automatic discretization network for the j-th numerical feature field, activation function is Leaky_ReLU [8] and α is the control factor of skip-connection.

最后得到的

$$\tilde{x}_j$$

需要经过带温度系数的softmax变成概率分布:

$$\hat{x}_j^h = \frac{e^{\frac{1}{\tau}\tilde{x}_j^h}}{\sum_{l=1}^{H_j} e^{\frac{1}{\tau}\tilde{x}_j^l}}, \tag{12}$$

传统的离散化方式是将特征取值分到某一个具体的桶中，即对每个桶的概率进行argmax，但这是一种无法进行梯度回传的方式，是硬离散化。而上式可以看作是一种软离散化（soft discretization）。对于**温度系数**τ，当其接近于0时，得到的分桶概率分布接近于one-hot，当其接近于无穷时，得到的分桶概率分布近似于均匀分布。这种方式也称为softargmax。

至此，我们得到了

$$H_j$$

个桶的embedding以及概率分布。

1.3.3 Aggregation Function

其实就是对

$$H_j$$

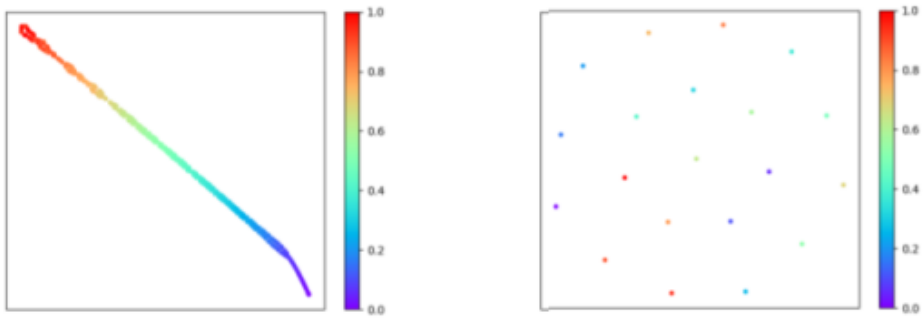
个桶的embedding进行加权求和。

模型的**训练**过程同一般的CTR过程相似，采用二分类的logloss指导模型训练，损失如下：

$$\mathcal{L} = -\frac{1}{Q}\sum_{i=1}^{Q} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i) + \lambda \|\Theta\|_2, \quad (17)$$

where y_i and $\hat{y}_i$ are the ground truth label and estimated value of the i-th instance, respectively. Q is the total number of training instances and λ is the L_2 regularization weight. $\Theta = \{\Theta_{Cat\_Emb}, \Theta_{AutoDis}, \Theta_{CTR}\}$ are the parameters of feature embeddings in categorical fields, parameters of meta-embeddings and automatic discretization (*i.e.*, Eq.(11-14)) in AutoDis, as well as deep CTR model parameters.

那么，AutoDis是否有效解决了SBD和DBS的问题呢？实验结果也印证了这一点：



(a) AutoDis embeddings　　(b) EDD embeddings

Figure 5: Visualization of the t-SNE transformed embeddings derived from AutoDis and EDD. The points in different colors reflect different feature values after normalization.

（右图：等深分箱，不同的取值都是分开的点，没有相似度的联系；左图：Autodis，相似的取值聚在一起，说明端到端的方法把握了数值的相似性）

为了使训练更稳定，我们需要先把特征做min-max归一化处理，即把min映射到0，把max映射到1. 所以，这样说来Autodis需要记录的数据分布就只是大致的min、max，比起等深分箱要记录的数据分布则少了很多。

实验效果:

- 准确率：等深分箱AUC为0.8138，Autodis的AUC为0.8152
- 内存消耗：等深分箱分H个桶，Autodis也分H个桶，只不过Autodis的一个样本可以"属于"不同的桶，所以参数量并没有怎么增加，只增加了"注意力分配"那一点点可以忽略不计的参数量。

综上，Autodis的优势主要在于一个样本可以"属于"不同的桶，实现了端到端的分桶优化，所以可以看到AUC有了微小的提升。然而，这样微小的提升本可以忽略不计，Autodis不可忽略的优势则在于，它只需要提前记录好min和max，而等深分箱则需要线下记录好整个数据的分布，来决定我们该如何分桶。倘若线上的数据分布发生了变化，那么我们线下确定的分桶方式就要失效了。

2. 谷歌 | 不需要embedding table的类别特征embedding方法

Learning to Embed Categorical Features without Embedding Tables for Recommendation

2.1 背景

在这篇文章中，我们主要研究很大的**vocabulary**中一个词语（特征）的embedding训练方法。我们知道，一个好的embedding对于模型来说是非常重要的。我们使用矩阵分解的方法分解user-item矩阵，得到比较好的初始化user/item ID embedding，可以用于CTR预估中ID类embedding的初始化；我们通过训练FM得到比较好的特征embedding，来把握content信息；在NLP中，我们通过训练word2vec得到词语的embedding。

对于类别型特征（用户ID/物品ID）标准的方式是用一个(巨大的) embedding table为每个类别特征分配一个embedding。然而这种方式有很大问题：

- 参数量巨大（Huge vocabulary size）：推荐系统通常包含几百万的用户ID/视频ID，如果每个特征都指定一个embedding会占据大量空间。
- 特征是动态的（Dynamic nature of input）：推荐系统中经常会出现**全新的**用户ID/视频ID，固定的embedding table不能解决**OOV**(out-of-vocabulary)问题（冷启动问题，遇到这种问题的话就只能通过图网络，把新的User/item根据content similarity和已有的user/item联系起来，然后通过图中embedding的聚合得到一个比较好的初始化embedding）
- 特征分布高度倾斜（Highly-skewed data distribution）：推荐数据中**低频**特征的训练实例数量较少，因此该特征的embedding在训练阶段就很少更新，对训练的质量有显著影响。

已有的类别特征embedding方法：

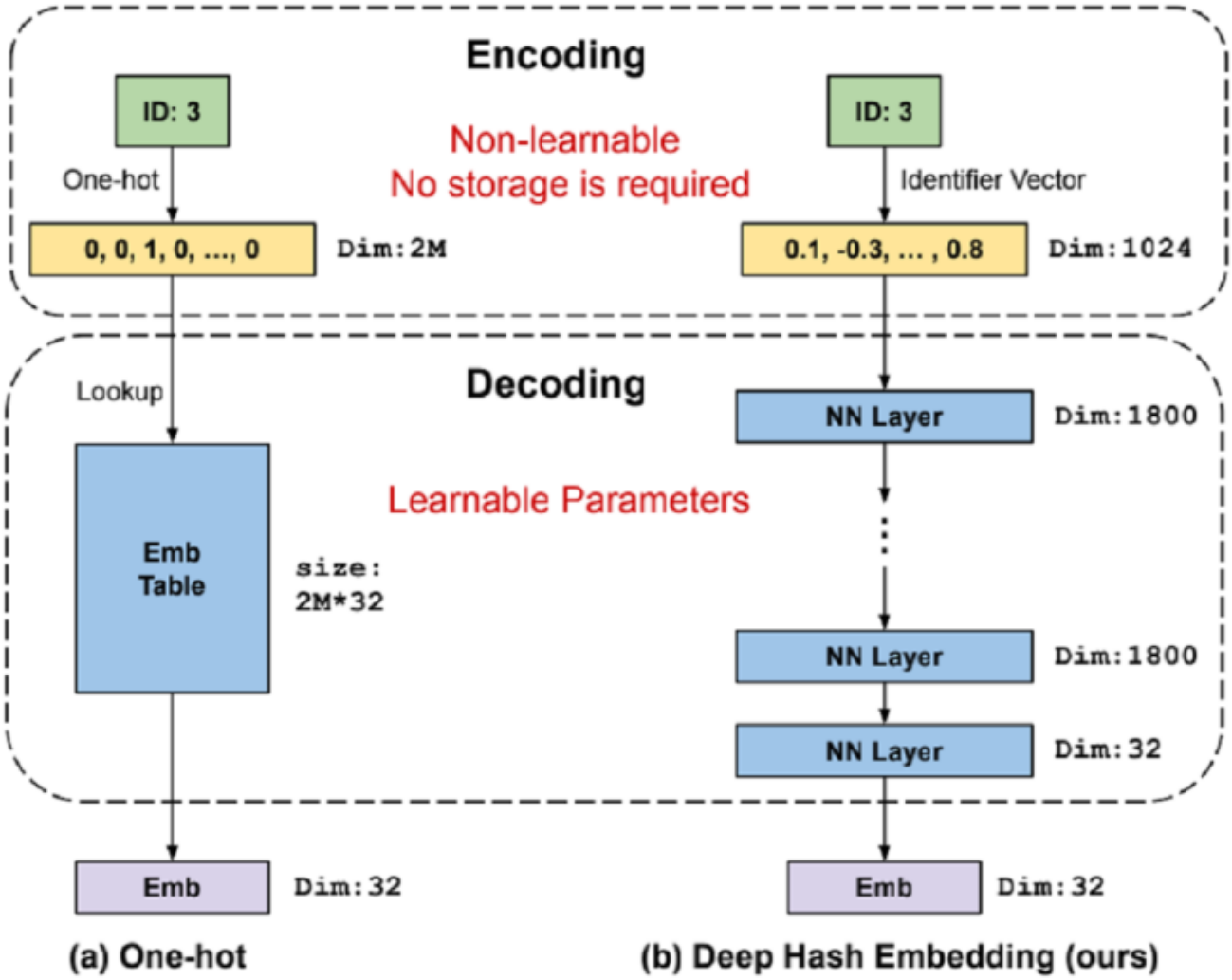
- **One-hot Full Embedding**：这种方式就是最常见的方法，做one-hot encoding，然后通过一个可学习的线性变换矩阵（说白了就是embedding table，可以看作一层神经网络，但没有bias项）得到对应的embedding表示：$\mathbf{e} = W^T \mathbf{b}$。缺点：embedding table随特征数量线性增长，很多时候embedding table占据了网络的大部分参数量，而实际的网络参数量相比而言很小（即内存问题）；无法处理新出现的特征（OOV），新特征来的时候就只能通过hashing的办法，哈希到公共溢出区的桶中，这样就不免会有哈希冲突的问题。
- **One-hot Hash Embedding**：为了解决One-hot Full Embedding中的内存消耗巨大的问题，可以使用**哈希**函数对类别特征进行映射分桶，将原始的 n维的 one-hot 特征编码映射为m维的 one-hot 特征编码(即m个桶)。这样，embedding table只用存储m项，大大降低了参数量。缺点：只要是哈希，就会有冲突！哈希冲突导致多个ID共用一个embedding, 这会伤害模型性能。

为了解决哈希冲突的问题，可以做如下改进：

取k个不同的哈希函数 $\{H^{(1)}, H^{(2)}, \dots H^{(k)}\}$, 按照上述方法生成**k**个one-hot编码：$\{\mathbf{b}^{(1)}, \mathbf{b}^{(2)}, \dots \mathbf{b}^{(k)}\}$(分到了k个桶), 每个one-hot编码都去查一下embedding table，并且把最后的结果concat到一起/或者做avg-pooling。

2.2 Deep Hash Embeddings

DHE将整个特征嵌入分为编码阶段(encoding)和解码阶段(decoding)。下图是One-hot Embedding与DHE的整体区别：



可以看到：

- DHE编码阶段通过多个(k=1024个)哈希函数将特征表示为**稠密的Identifier vector**, 解码阶段通过多层神经网络得到该特征的唯一表示。

2.2.1 Encoding阶段

【一个好的encoding应该有哪些特性】

- 唯一性（Uniqueness）：每个不同特征值的编码应该是唯一的。
- 同等相似性（ Equal Similarity）：只有唯一表示是不够的。例如二进制编码中：9表示为1001，8表示为1000：，7表示为0111。我们发现8的表示和9的表示更相似，这会引入bias，让编码器以为id = 8 与 id = 9比起id = 7 与 id = 9更相似，然而id类特征是没有顺序的，因此它们应该是**同等相似的。**
- 高维（High dimensionality）：我们希望这些编码便于后续解码函数区分不同的特征值。由于高维空间通常被认为是更可分的 (回忆一下SVM中的kernel方法...)，我们认为编码维度也应该相对较高。
- 高香农熵（High Shannon Entropy）：香农熵(以bit为单位)测量一个维度中所携带的信息。从信息论的角度来看，高香农熵的要求是为了防止冗余维数。例如，一个编码方案可能满足上述三个属性，但在某些维度上，所有特征值的编码值是相同的。所以我们希望通过最大化每个维度的熵来有效地利用所有维度。例如，

one-hot编码在每个维度上的熵都很低，因为对于大多数特征值来说，每个维度上的编码都是0。因此，one-hot编码需要非常高的维度(即)，而且效率非常低。高香农熵就是要让encoding更加高效。

DHE编码阶段(encoding)的设计：

提出的DHE运用 k 个哈希函数把每个类别特征映射为一个 k 维的稠密向量。

具体的，每个哈希函数 $H^{(i)}$ 都将一个正整数 $\mathbb{N}$ 映射到$1, 2, \ldots m$,本实验中取 $m = 1e6$ 。因此，k个哈希函数就把一个正整数 $\mathbb{N}$映射成了k维的向量 $E'(s) = [H^{(1)}(s), H^{(2)}(s), \ldots, H^{(k)}(s)]$, 向量中的每个元素都取自{1,2,...m}。实验中取k=1024.

然而，直接用上面得到的编码表示是不合适的，因此作者进行了两个变换操作来保证数值稳定性：

- 均匀分布（Uniform Distribution）：把 $E'(s)$中的每个值映射到[-1,1]之间
- 高斯分布（Gaussian Distribution）：把经过均匀分布后的向量转化为高斯分布 $N(0,1)$ 。

(作者说，这里是受到GAN网络的启发，用服从高斯分布的随机变量做GAN网络的输入。)

作者在文章中验证了这样设计的encoding满足上述的四个条件。

2.2.2 Decoding 阶段

Decoding阶段需要把Encoding阶段得到的 k 维向量映射为 d 维。如上面的图所示，作者用**多层神经网络**来实现。但是，由于参数量明显比embedding table降低很多，这种多层神经网络会导致**欠拟合**。因此，作者尝试了Mish激活函数 $f(x) = x \cdot tanh(ln(1 + e^x))$ 来代替ReLU激活函数，**引入更多的非线性**，从而提升表征能力。

作者还考虑了batch normalization (BN)等训练技巧。但是不能使用dropout，因为我们的问题是欠拟合而不是过拟合。

2.3 加入辅助信息以增强【泛化性】(解决OOV问题)

side information（内容信息）经常被用于推荐模型中，但是很少被用于生成类别型特征的embedding部分（其实在user-item图网络中，也利用了内容信息，这也算是side information在类别型特征embedding中的应用）。

- 记忆性(memorization): 例如one-hot编码的每个id embedding都是独立的，因此只有记忆性没有泛化性。ID 7和ID 8完全没有关系，不能够generalize。
- 泛化性(generalization): 本文提出的DHE方法，embedding network中的参数变化会影响所有特征的embedding结果，所以能够提供泛化能力。

另外，为了进一步增强泛化能力，对于物品ID/用户ID的特征embedding，可以考虑拼接上它们属性（年龄、品牌等）的表示，然后输入到DHE解码阶段来生成最终的特征嵌入。这样，哈希出来的unique identifier提供的是记忆能力、side information能够增强泛化能力。

结果：

- 准确率：DHE取得了和one-hot full embedding相近的性能
- 速度：DHE的速度比one-hot full embedding要慢，full embedding对1M查询（batchsize = 100）的速度为3.4s，而DHE的速度为27.2s。这是因为DHE需要计算哈希函数、之后还要通过神经网络计算稠密表征。但

是，欣慰的是，DHE通过GPU可以进行很大比例的加速。而full embedding这个"查embedding table"操作并不能够被GPU加速。所以，利用足够好的GPU，我们就可以减小DHE速度和full embedding的差异。

Table 8: Time (in seconds) of embedding generation for 1M queries with a batch size of 100.

	CPU	GPU	GPU Acceleration
Full Emb	3.4	3.4	-1%
Hash Emb [34]	8.4	6.1	-26%
DHE	76.1	27.2	-64%

- 内存：DHE的内存消耗仅是full embedding的1/4.这是因为原先要存储非常多的ID embedding，而现在只需要存储MLP的参数即可。

实际应用：

在精排阶段，对于那些种类特别多的ID 类特征，例如商品三级品类ID，我们可以用DHE的方法做embedding，只训练对应的MLP参数，不用存储巨大的embedding table，来降低内存消耗；对于那些种类不那么多的特征，例如城市、学历、年龄，我们直接用one-hot full embedding即可。其实，DHE可以看成是full embedding的一种近似模拟，如果内存不是瓶颈的话，还是用full embedding对AUC是最好的。

在召回阶段，也可以用DHE来得到user/item ID表征。传统的方法就是计算user-item的矩阵分解，或者利用MLP来做模拟的矩阵分解（Neural Collaborative Filtering），但不管怎样，我们最后都需要存储巨大的ID embedding table。而用DHE来生成user/item embedding时，可以先使用DHE生成user/item的embedding之后，再求二者点积，并以真实的点击/不点击标签作为label来训练，以此来更新MLP网络参数。这样，就减少了很多参数量，也能够得到比较好的user/item embedding。为了增强泛化能力，我们还可以在unique identifier后拼接side information，得到融合了side information的user/item embedding。