【浪潮之巅】4.Wide&Deep

原创 小陈 小陈的推荐算法学习笔记 5天前

0 一绪

前文PNN是通过定义多种特征向量交叉的操作，加强特征交叉的能力，这篇文章主要是介绍组合两种不同优点、优势互补的网络，提升模型综合能力的模型--Wide&Deep。

Wide&Deep是谷歌发表的实操性论文，里面有大量的工程经验和方法。网上关于这篇论文的解读很全面，也很精彩，正是因为这一点，小陈在上周一直摇摆不定，不知道从何开始解读这篇论文。

在犹豫了一周后，小陈在已有的数据基础上实现Wide&Deep 模型，但是因为已有线上running的模型效果更好，就没有做详细的实验。

1 Introduction

在推荐系统中，一个很大的挑战是如何同时让推荐结果满足扩展性和准确性（E&E问题）。

推荐的内容都是精准内容，用户兴趣收敛，无新鲜感，不利于长久的用户留存；推荐内容过于泛化，用户的精准兴趣无法得到满足，用户流失风险很大。相比较推荐的准确性，扩展性倾向于改善推荐系统的多样性。

谷歌将上述问题定义为Memorization（记忆）和Generalization（泛化）问题。

1.1 Memorization

Memorization表示从历史数据中发现item或者特征之间的相关性。在历用户史数据中，用户点击某个item是伴随一些特征的组合的。

比如说，预测玩游戏，特征组合[Type=strategy，isMoney=no，Hour=twenty]代表时间20，不需要花钱的策略游戏。这个特征组合出现过，并且有部分用户在这个特征组合下玩了游戏，那么在给另外一个用户预测时候，拥有这个特征组合的用户有很大概率会玩游戏，此时模型预测的得分也趋向于高得分。

虽然Memorization可以“记忆”特征组合，但是现在的特征维度是庞大的，按照维度n最大的组合可能性 2^n ，这种情况是不可能容忍的，并且特征组合是需要人工来做交叉的，这也是不现实的。

1.2 Generalization

Generalization表示相关性的传递，发现在历史数据中很少或者没有出现的新的特征组合，通过泛化出现过特征从而解释新出现特征的能力。

比如说，还是预测玩游戏，特征组合[Type=strategy，isMoney=no，Hour=eighteen]代表时间18，不需要花钱的策略游戏，这个特征组合在以往的历史数据中完全没有出现过，但是因为[Type=strategy，isMoney=no]组合在以前出现过，通过泛化这个出现过的组合，可以解释新的特征组合，用户在20点玩不花钱的策略游戏，那么会不会在其他时间点玩不花钱的策略游戏呢?

但是正是因为这种对历史上没有出现的特征组合有更好的泛化性，也带来了Generalization的弊端（过度泛化），会给用户推荐不是那么相关的物品，尤其是用户和物料item的交互矩阵较为稀疏时。比如说小陈就非常不喜欢智力类游戏，不会跟智力类游戏产生交互行为，但是模型的过度泛化，embedding方法可以得到小陈对所有游戏的非零预测，这个时候模型就对小陈过度解读了，推荐我不喜欢的游戏。

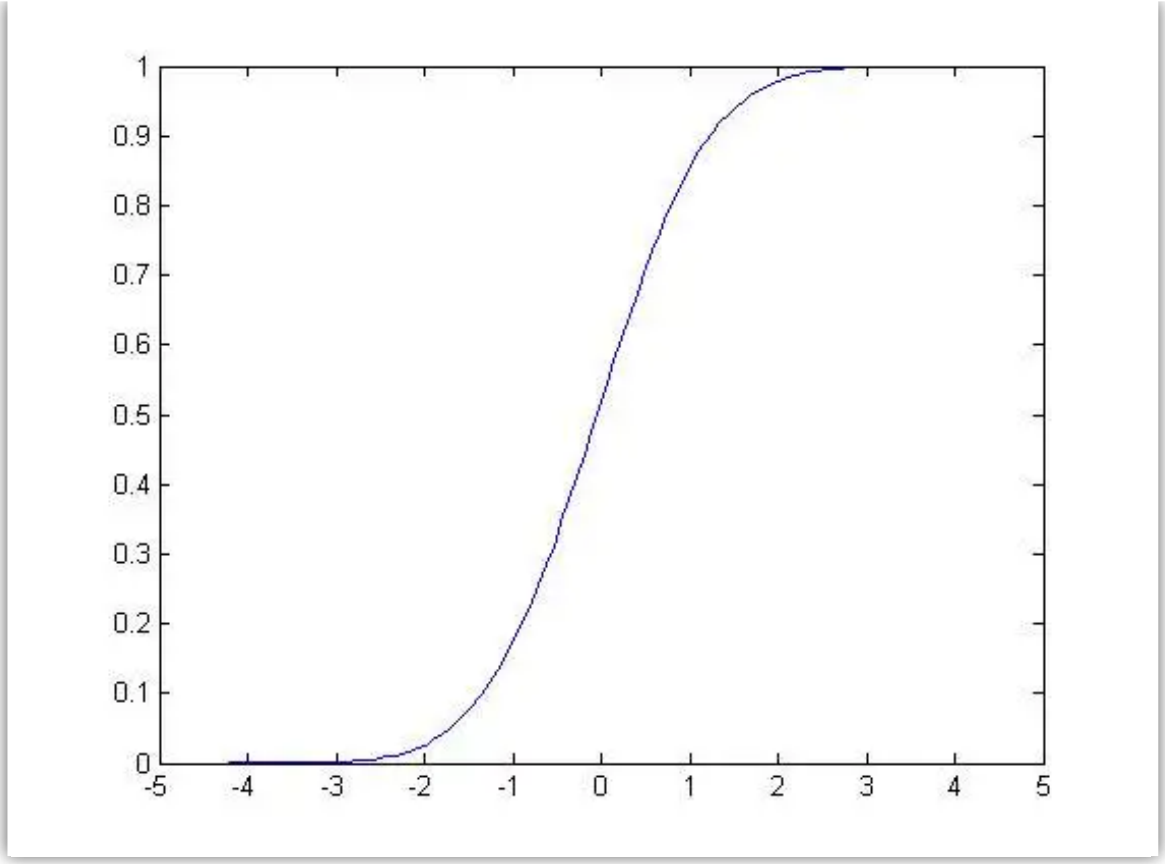
2 Feature Engineering

特征工程方面，虽然论文中篇幅不多，但是在实际使用中，这部分重之又重，论文中实际提到的方法主要有两种CDF归一化和低频特征过滤。

2.1 CDF归一化

CDF归一化针对数值型连续特征，刚接触CDF这个词，小陈是一脸懵逼，后来知道是累积分布归一化才松了口气，这道题小陈会啊!

不就是把所有数值排序好，横坐标是特征值X，纵坐标是在所有数值中，小于特征X的概率，类似于下面图中曲线。



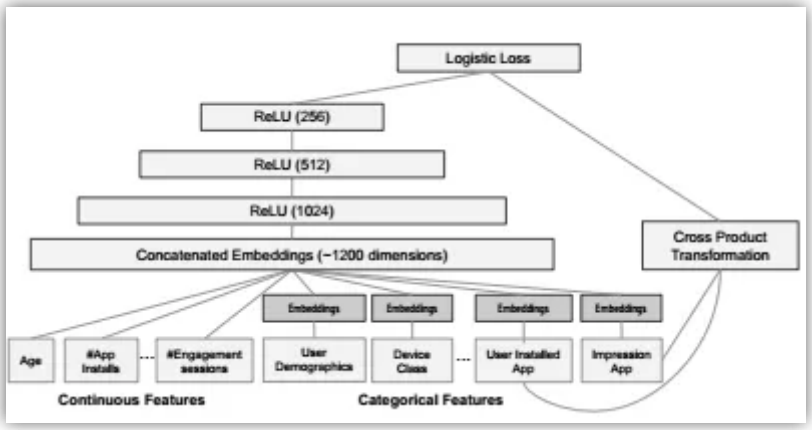
但是如果把整个特征分为n个桶，那么每个桶的边界如何确定呢？简单点说就是每个特征值怎么归到相应的桶里面。思考到这里，小陈又是一脸沉思了，道理我都知道，可是怎么做啊？

换个角度想想，这个CDF操作之后进行分桶，本质上就是进行分位数操作，取到分位数就行。而在Spark里面有分位数离散器QuantileDiscretizer，只需要确定分桶的数量就行，一键简单便捷。

论文中，分桶之后再做下最大值归一化到[0,1]区间。

但是，论文中这个操作小陈有点不明白，既然已经做了分桶的操作了，为什么不做one-hot处理，而是只是归一化呢？

其实答案已经在论文中给出了，下图中，对于连续的特征，并未做embedding处理，而是直接连入网络。那为什么这些连续特征不做embedding呢？



embedding也叫分布式表达，是利用其它维度的来表示某个特征，这里的embedding层size统一是32维。category feature全部embedding是因为这些特征蕴含用户和物料的大量信息，肯定是不止一维的，需要更多维来表达。但是数值特征，要么是基于统计的，要么就是年龄这类值代表具体含义的特征，已经是最细粒度的特征，可以认为它们是一维的embedding特征了，因此不需要做one-hot处理，并且one-hot处理之后变成category feature，还是需要做embedding表示，不仅徒增成本，而且还会在处理过程中损失信息。

解决了上面的问题，另外一个问题接踵而至，那为什么在CDF操作之后非要分桶归一化处理呢?

答案很简单，这样做就是为了使数值特征粗粒化，增加泛化能力。在deep部分，主要是负责Generalization，那么这样的特征的处理也是对于deep层的设计动机也是有帮助的。

2.2 Sparse低频特征过滤

对于category feature的特征处理，就更加考验实际的工程经验。

论文中是过滤掉category特征中出现频次低于某个阈值的部分。举个栗子，游戏类型一共有80种，在样本中，智力游戏这个特征出现了10次，策略游戏出现了1000次，棋牌游戏出现了100次。

我们设定，在所有样本中出现次数小于20次的特征全部要被干掉。那么智力游戏这个特征就不会被表达了，这些统一被过滤掉的特征会全部归为新的一类。就像Deep Crossing中对ID的处理一样，10000个之后的ID统一归为一类。

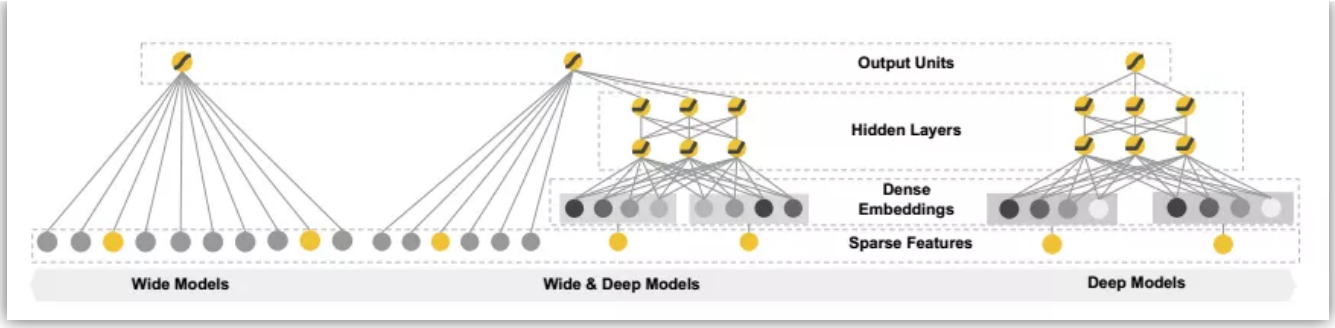
这个最小阈值该如何设定呢？论文中并没有给出。这个需要结合具体业务实际去试。

毫无疑问，对于deep部分，这种操作增加了泛化性。

对于wide层，这个操作增加了稀疏性，也是有利于模型。因为wide层输入是安装APP列表和预测APP的笛卡尔积，对于上百万APP，这种交叉的维度是巨大的，对应的参数量巨大，而通过把出现的很低频的APP直接过滤掉，直接减少整个参数空间量级，大大提高的模型的使用性。

3 Framework

从文章题目中顾名思义，Wide & Deep 是融合 Wide Models 和 Deep Models 得到，下图形象地展示出来。



3.1 Wide Part

wide部分是最常见的广义线性模型。

3.1.1 raw input

在实际的模型输入中，真正的输入只有两个特征，一个是用户安装过的APP列表，另一个是需要预测得分的APP 。实际上，APP都会转化为一个ID table，通过查找映射来达到对APP进行编码的目的。

在小陈的实际使用中，这种APP特征因为维度巨大，使用显性的map映射不太现实，这时候就需要通过hash算法。因此对应的安装列表就会变成Muti-hot的稀疏表示，预测的APP就会变成one-hot表示。

这部分又会出现一个问题，为什么谷歌团队在wide部分，输入的数据特征只有两个？下面解释这个问题。

3.1.2 cross-product transformation

wide既然实现Memorization功能，就需要记忆特征的组合。对于谷歌商店这种场景，可以说用户安装的APP列表与预测的APP之间存在某种联系，用户在安装了[A,B,C,D]四个APP之后，根据以往的历史，也会安装E，那么这个时候模型学习到了这种特征组合，这种特征组合的得分就会较高。

因此输入的特征只有两维不无道理，并且，输入的特征应该和业务场景的目标有关。或许在谷歌开发者在实践过程中，明显发现了这种特征组合对模型的重要性，而不是随便的加入特征进行组合，不仅耗费人力，还增加模型的复杂度。

特征交叉环节，APP的维度是百万级别的，用户安装列表的Muti-hot和预测APP的one-hot进行笛卡尔积，毫无疑问，维度直接爆炸，并且会很稀疏。

3.1.3 FTRL with L1 regularization

这里有两个问题需要解释，分别是关于FTRL和L1正则化。

用FTLR训练的作用?

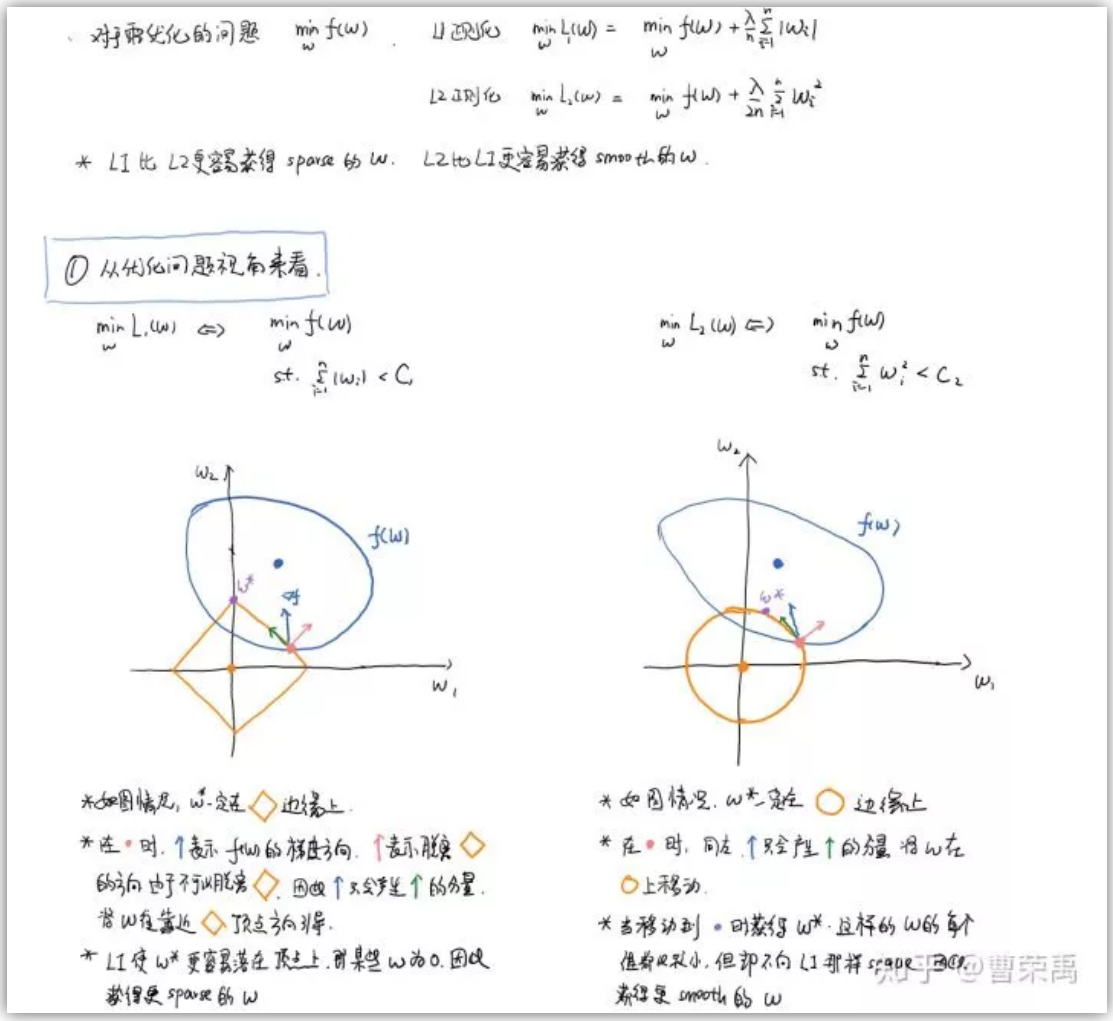
这个问题从两个方面来回答。

- 实时性。FTLR用到的是随机梯度下降算法，因此可以实现模型的在线更新，可以实现在线学习能力。
- 稀疏性。FTLR中采用了阈值截断，小于阈值的设置为0，会增加稀疏性。

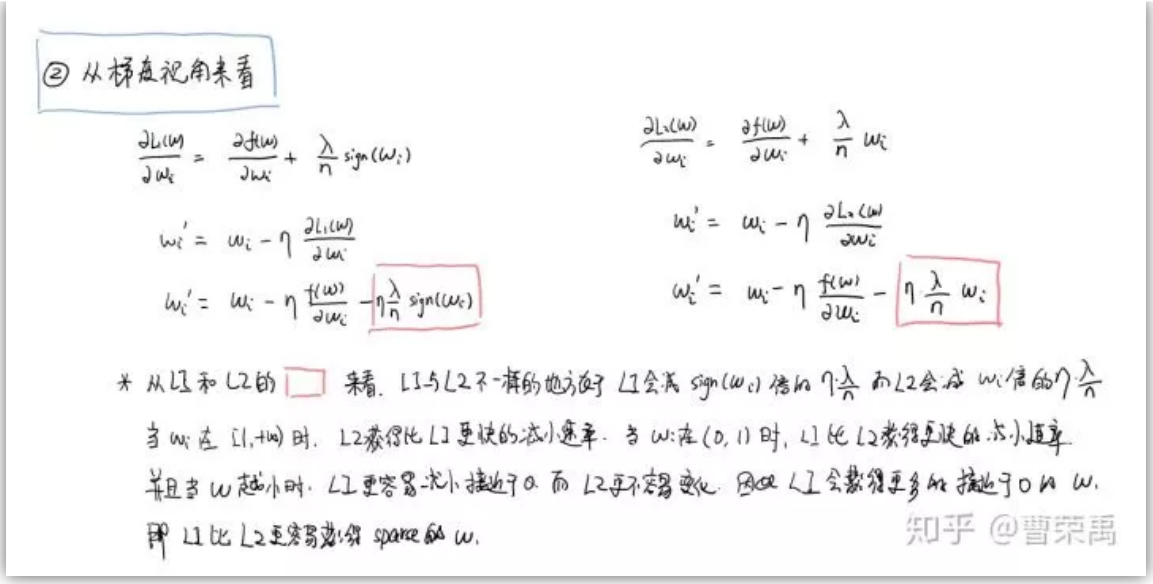
L1正则化的作用?

与L2正则化不同，L1正则化得到的权重更加稀疏，为什么L1正则化比L2正则化更容易产生稀疏解呢？借鉴@曹荣禹的回答，从三个角度来看：

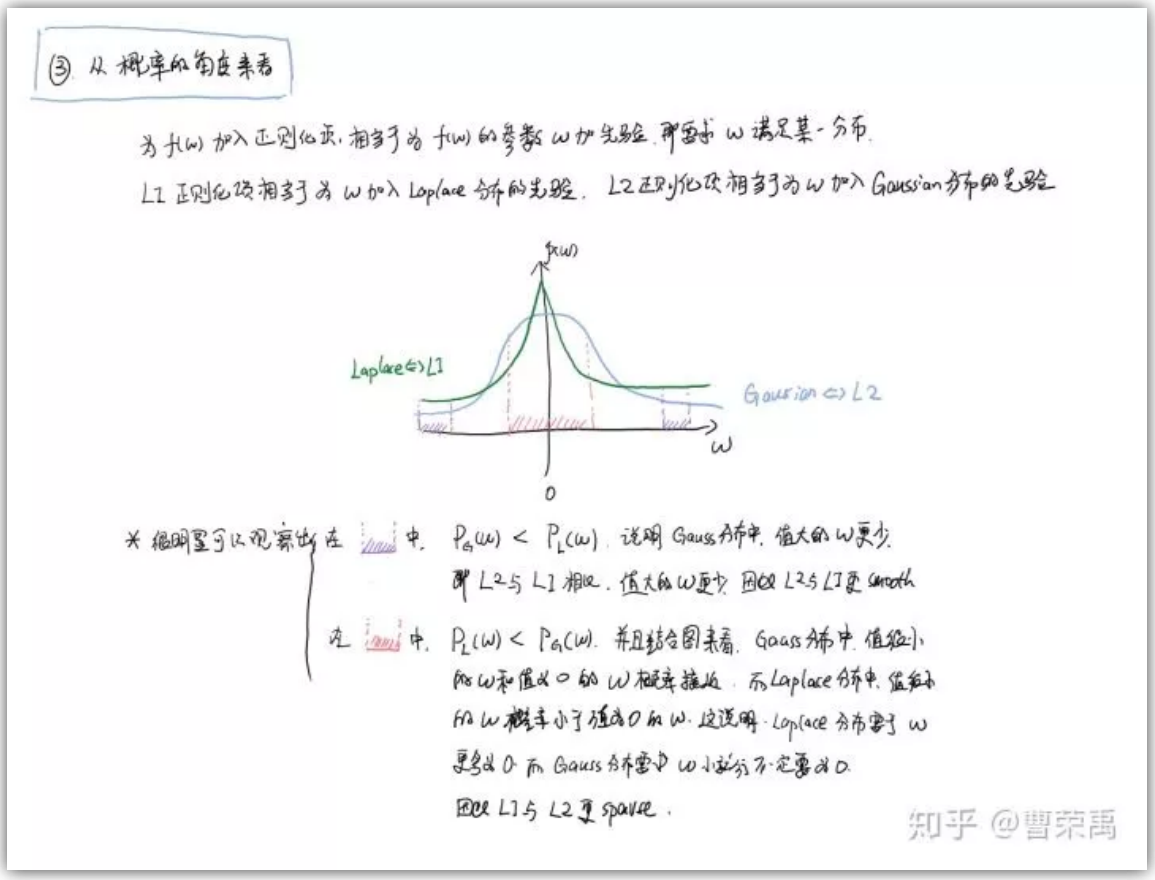
- 优化视角



- 梯度视角



- 概率视角



解释完FTRL和L1正则化这两个问题，那么 FTRL with L1 regularization 的作用不言而喻，就是增加模型的稀疏性，用大白话说就是在wide部分里面，让大部分的权重都为0，这样会减少模型的参数量，提高线上推理的效率。

谷歌开发者“处心积虑”的设计稀疏性，根本原因是什么呢？回到数据本身，在cross-product transformation之后，数据的维度是极其庞大的，但是不可能让所有维度都有一个权重，因为线上serving不允许你这么做，所以给模型“瘦身”是个很好的工程经验。

3.2 Deep Part

Deep Part是一个标准的前馈神经网络。

具体的特征处理在前面已经讲过，简单来说就是，经过CDF归一化分桶处理的数值特征和通过embedding层的category特征拼接在一起。输入到神经网络，大概有1200维。

deep part主要的目的就是实现Generalization，主要是通过低纬度的embedding表达，来泛化推荐一些字符上不相关，但是实际含义有关联或者是用户可能需要的。

还是举个栗子解释下什么叫泛化，小陈除了玩策略游戏外，还喜欢玩实时竞技游戏，虽然这两种游戏不同，但是是很相近的，所以也会给小陈推荐实时竞技游戏。

3.3 joint training or ensemble

组合模型一般有两种训练方式joint training 和 ensemble，那论文中是什么样的?
ensemble 有三种形式：

- bagging：训练集进行子抽样组成每个基模型所需要的子训练集，对所有基模型预测的结果进行综合产生最终的预测结果
- boosting：训练过程为阶梯状，基模型按次序一一进行训练（实现上可以做到并行），基模型的训练集按照某种策略每次都进行一定的转化。对所有基模型预测的结果进行线性综合产生最终的预测结果
- stacking：将训练好的所有基模型对训练基进行预测，第j个基模型对第i个训练样本的预测值将作为新的训练集中第i个样本的第j个特征值，最后基于新的训练集进行训练。同理，预测的过程也要先经过所有基模型的预测形成新的测试集，最后再对测试集进行预测

joint training 将不同的模型结果放在同一个损失函数中进行优化。因此，ensemble 要求模型独立预测时就有些的表现，一般而言模型会比较大。由于 joint training 训练方式的限制，每个模型需要有不同的侧重。对于 Wide&Deep 模型来说，wide 部分只需要处理 Deep 在低阶组合特征学习的不足，所以可以使用简单的结果，最终完美使用 joint traing。

预测时，会将 Wide 和 Deep 的输出加权得到结果。在训练时，使用 logistic loss function 做为损失函数。

3.4 logits

Wide 部分通过 FTRL + L1 优化，而Deep 部分通过 mini-batch的AdaGrad 优化。

这两个部分在训练时候会有一个小的冲突，deep层是batch处理，而wide层是单个样本处理，那用什么样的训练方式统一呢？

参考知乎@石塔西的方案，会有以下步骤：

- Deep侧先完成前代，得到一个batch下所有样本的deep_logits
- Wide侧在逐一学习每个样本时，先得到这条样本的wide_logit，再去已经计算好的deep_logits中找到这个样本的deep_logit
- 将一个样本的wide_logit和deep_logit代入sigmoid函数
- 计算梯度，开始回代

4 Code

代码是@石塔西的，直接拿过来用了。

首先是Wide核心部分

```
def train(self, sp_features, labels):
    """
        :param sp_features: dict from field_name ==> SparseInput
        :return: probabilities from this train batch
        """
    probas = []
    for example_idx, label in enumerate(labels):
        logit = self.__predict_logit(sp_features, example_idx)

        pred_proba = self._proba_fn(example_idx, logit)
        probas.append(pred_proba)

        for _, estimator in self._estimators.items():
            estimator.update(pred_proba=pred_proba, label=label)

    return np.asarray(probas)
```

接着是Deep核心部分

```
def forward(self, features):
```

```
    """
        :param features: dict, mapping from field=>dense ndarray or field=>Sp
        :return: logits, [batch_size]
        """
    dense_input = self._dense_combine_layer.forward(features)

    embed_input = self._embed_combine_layer.forward(features)

    X = np.hstack([dense_input, embed_input])

    for hidden_layer in self._hidden_layers:
        X = hidden_layer.forward(X)

    return X.flatten()

def backward(self, grads2logits):
    """
        :param grads2logits: gradients from loss to logits, [batch_size]
        """
    # ***************** 计算所有梯度
    prev_grads = grads2logits.reshape([-1, 1])  # reshape to [batch_size,1]

    # iterate hidden layers backwards
    for hidden_layer in self._hidden_layers[::-1]:
        prev_grads = hidden_layer.backward(prev_grads)

    col_sizes = [self._dense_combine_layer.output_dim, self._embed_combine_la
        # 抛弃第一个split，因为其对应的是input，无可优化
    _, grads_for_all_embedding = utils.split_column(prev_grads, col_sizes)
  self._embed_combine_layer.backward(grads_for_all_embedding)

    # ***************** 优化
    # 这个操作必须每次backward都调用，这是因为，尽管dense部分的权重是固定的
    # 但是sparse部分，要优化哪个变量，是随着输入不同而不同的
    all_vars, all_grads2var = {}, {}
    for opt_layer in self._optimize_layers:
        all_vars.update(opt_layer.variables)
        all_grads2var.update(opt_layer.grads2var)

    self._optimizer.update(variables=all_vars, gradients=all_grads2var)
```

最后是logits的计算代码，跟前面3.4讲的逻辑保持一致

```
def train_batch(self, features, labels):
    self._current_deep_logits = self._dnn.forward(features)

    pred_probas = self._wide_layer.train(features, labels)

    self._dnn.backward(grads2logits=pred_probas - labels)

    return pred_probas
```

5 总结

Wide&Deep模型不仅仅是提出了Memorization和Generalization，而且开启了不同网络结构融合的新思路。借鉴不同模型的优势结合业务本身，进行互补，确实是个很好的方向点。
目前Wide 结合 Deep的思想已经非常流行，结构虽然简单，从业界的很多反馈来看，合理地结合自身业务借鉴该结构，实际效果确实是efficient，小陈工作中使用的推荐模型中，也借鉴了该思想。

言而总之，推荐系统中，Memorization和Generalization都十分重要。

参考文献

[1]https://xiang578.com/post/wide-and-deep.html

[2]https://zhuanlan.zhihu.com/p/37733208

[3]https://zhuanlan.zhihu.com/p/53361519

[4]https://zhuanlan.zhihu.com/p/74957209

[5]https://zhuanlan.zhihu.com/p/142958834

[6]https://zhuanlan.zhihu.com/p/47293765