计算广告CTR预估系列(四)--Wide&Deep理论与实践

原创 李宁宁 机器学习荐货情报局 2018-05-21

计算广告CTR预估系列(四)—Wide&Deep理论与实践

> 今天的主角是Wide & Deep Model，在推荐系统和CTR预估中都有应用。万字长文，墙裂推荐！

1. 名词解释

1.1 Memorization 和 Generalization

> Google Wide&Deep论文中，通篇都是这两个词，必须搞懂是怎么回事!

这个是从人类的认知学习过程中演化来的。人类的大脑很复杂，它可以记忆(memorize)下每天发生的事情（麻雀可以飞，鸽子可以飞）然后泛化(generalize)这些知识到之前没有看到过的东西（有翅膀的动物都能飞）。
但是泛化的规则有时候不是特别的准，有时候会出错（有翅膀的动物都能飞吗）。那怎么办那，没关系，记忆(memorization)可以修正泛化的规则(generalized rules)，叫做特例（企鹅有翅膀，但是不能飞）。

这就是Memorization和Generalization的来由或者说含义。

> Wide&Deep Mode就是希望计算机可以像人脑一样，可以同时发挥memorization 和 generalization 的 作 用 。 —Heng-Tze Cheng(Wide&Deep作者)

1.2 Wide 和 Deep

> 同样，这两个词也是通篇出现，究竟什么意思你明白了没?

其实，Wide也是一种特殊的神经网络，他的输入直接和输出相连。属于广义线性模型的范畴。Deep就是指Deep Neural Network，这个很好理解。Wide Linear Model用于memorization；Deep Neural Network用于generalization。
左侧是Wide-only，右侧是Deep-only，中间是Wide & Deep:

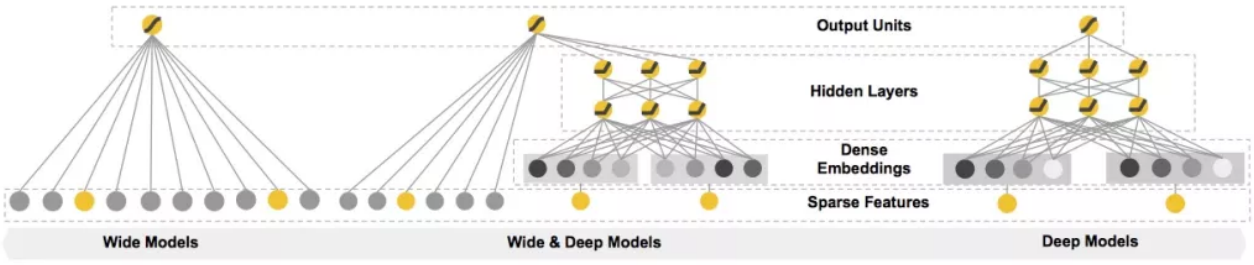


Figure 1: The spectrum of Wide & Deep models.

1.3 Cross-product transformation

Wide中不断提到这样一种变换用来生成组合特征，也必须搞懂才行哦。它的定义如下：

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

k表示第k个组合特征。i表示输入X的第i维特征。C_ki表示这个第i维度特征是否要参与第k个组合特征的构造。d表示输入X的维度。那么到底有哪些维度特征要参与构造组合特征那？这个是你之前自己定好的，在公式中没有体现。

饶了一大圈，整这么一个复杂的公式，其实就是我们之前一直在说的one-hot之后的组合特征：仅仅在输入样本X中的特征gender=female和特征language=en同时为1，新的组合特征 `AND(gender=female, language=en)` 才为1。所以只要把两个特征的值相乘就可以了。

> Cross-product transformation 可以在二值特征中学习到组合特征，并且为模型增加非线性

2. Wide & Deep Model

Memorization:
之前大规模稀疏输入的处理是：通过线性模型 + 特征交叉。所带来的Memorization以及记忆能力非常有效和可解释。但是Generalization（泛化能力）需要更多的人工特征工程。

Generalization:
相比之下，DNN几乎不需要特征工程。通过对低纬度的dense embedding进行组合可以学习到更深层次的隐藏特征。但是，缺点是有点over-generalize（过度泛化）。推荐系统中表现为：会给用户推荐不是那么相关的物品，尤其是user-item矩阵比较稀疏并且是high-rank（高秩矩阵）

两者区别:
Memorization趋向于更加保守，推荐用户之前有过行为的items。相比之下，generalization更加趋向于提高推荐系统的多样性（diversity）。

Wide & Deep:
Wide & Deep包括两部分：线性模型 + DNN部分。结合上面两者的优点，平衡memorization和generalization。

原因：综合memorization和generalizatio的优点，服务于推荐系统。相比于wide-only和deep-only的模型，wide & deep提升显著（这么比较脸是不是有点大。。。）

2.1 推荐系统

2.1.1 介绍

推荐系统分为两种： CF-Based（协同过滤）、Content-Based（基于内容的推荐）

- 协同过滤(collaborative filtering)就是指基于用户的推荐，用户A和B比较相似，那么A喜欢的B也可能喜欢
- 基于内容推荐是指物品item1和item2比较相似，那么喜欢item1的用户多半也喜欢item2

2.1.2 线性模型

大规模的在线推荐系统中，logistic regression应用非常广泛，因为其**简单、易扩展、可解释性。**LR的输入多半是二值化后的one-hot稀疏特征。Memorization可以通过在稀疏特征上cross-product transformations来实现，比如：AND(user_installed_app=QQ, impression_app=WeChat)，当特征user_installed_app=QQ,和特征impression_app=WeChat取值都为1的时候，组合特征AND(user_installed_app=QQ, impression_app=WeChat)的取值才为1，否则为0。

推荐系统可以看成是一个**search ranking问题**，根据query得到items候选列表，然后对items通过ranking算法排序，得到最终的推荐列表。**Wide & Deep模型是用来解决ranking问题的。**

> 如果仅仅使用线性模型：无法学习到训练集中没有的query-item特征组合。Embedding-based Model可以解决这个问题。

2.1.3 Embedding-Based

FM和DNN都算是这样的模型，可以在很少的特征工程情况下，通过学习一个低纬度的embedding vector来学习训练集中从未见过的组合特征。

FM和DNN的缺点在于：当query-item矩阵是稀疏并且是high-rank的时候（比如user有特殊的爱好，或item比较小众），很难非常效率的学习出低维度的表示。这种情况下，大部分的query-item都没有什么关系。但是dense embedding会导致几乎所有的query-item预测值都是非0的，这就导致了推荐过度泛化，会推荐一些不那么相关的物品。

相反，linear model却可以通过cross-product transformation来记住这些**exception rules**，而且仅仅使用了非常少的参数。

总结一下：

> 线性模型无法学习到训练集中未出现的组合特征；
> FM或DNN通过学习embedding vector虽然可以学习到训练集中未出现的组合特征，但是会过度泛化。

Wide & Deep Model通过组合这两部分，解决了这些问题。

2.1.4 工作流程

> 总的来说， 推荐系统 = Retrieval + Ranking

推荐系统工作流程如下：

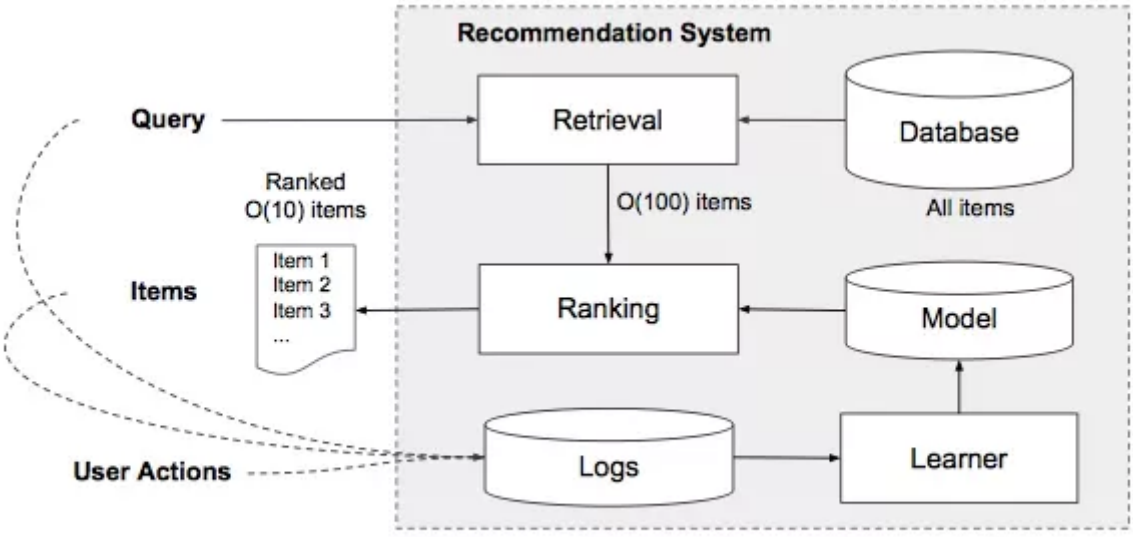


Figure 2: Overview of the recommender system.

想象这样一个实际情况：**我们打开Google APP store，首页展示给我们一些APP，我们点击或者下载或者购买了其中一个APP。**在这样一个流程中，推荐系统是如何工作的那？

我们对比上面的图一点点来说：

Query: 当我们打开APP Store的时候，就产生了一次Query，它包含两部分的特征：**User features, contextual features**。UserFeatures包括性别、年龄等人口统计特征，ContextualFeatures包括设备、时间等上下文特征。

Items:
APP store接着展示给我们一系列的app，这些app就是推荐系统针对我们的Query给出的推荐。这个也被叫做**impression**。

User Actions:
针对推荐给你的任何一个APP，我们都可以点击、下载、购买等操作。也就是说推荐给你的APP，你产生了某种行为。这不正是我们的最终目的吗！

Logs:
`Logs = Query + Impression + UserAction` 查询、展示列表、操作会被记录到logs中作为**训练数据**给Learner来学习。

Retrieval:
假如让你来想一个最简单的推荐系统，**针对这一次Query**，来给出推荐列表。你能想到的最简单，最暴力的做法是什么那？

> 给数据库中所有的APP都打一个分数，然后按照分数从高到低返回前N个（比如说前100个）

但是有个问题，这样数据库中的APP实在是太多了，为了保证响应时间，这样做太慢了！**Retrieval**就是用来解决这个问题的。它会利用机器学习模型和一些人为定义的规则，来返回最匹配当前Query的一个小的items集合，这个集合就是最终的推荐列表的候选集。

Ranking:

> 今天的主角Wide&Deep Model就是用来做这个事情的啦。

前面Learner学习到了一个Model，利用这个Model对Retrieval给出的候选集APP打分！并按照打分从高到低来排序，并返回前10个APP作为最终的推荐结果展示给用户。

Retrieval system完了之后，就是Ranking system。Retrieval减小了候选items池，Ranking system要做的就是对比当前的Query，对Candidate pool里面的所

有item打分！

得分score表示成P(y|x)，表示的是一个条件概率。y是label，表示user可以采取的action，比如点击或者购买。x表示输入，特征包括：

- User features（年龄、性别、语言、民族等）
- Contextual features(上下文特征：设备，时间等)
- Impression features（展示特征：app age、app的历史统计信息等）

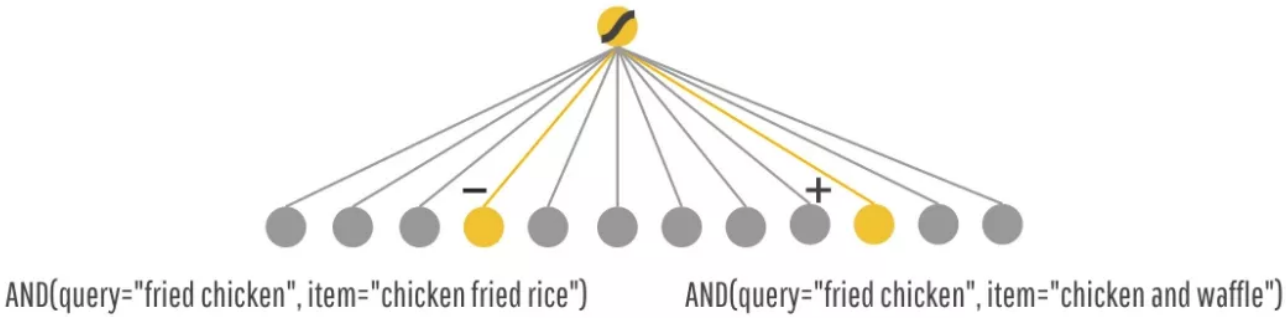
2.2 Wide Part

> Wide Part其实是一个广义的线性模型

使用特征包括：

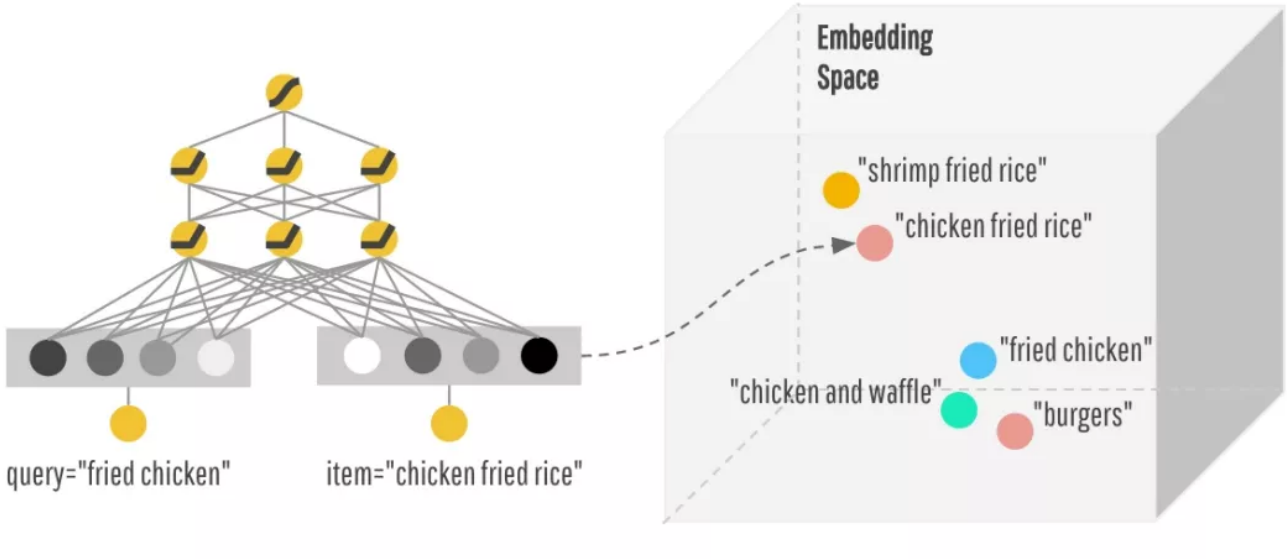
- raw input 原始特征
- cross-product transformation 组合特征

接下来我们用同一个例子来说明：你给model一个query（你想吃的美食），model返回给你一个美食，然后你购买/消费了这个推荐。 也就是说，推荐系统其实要学习的是这样一个条件概率： **P(consumption | query, item)**



Wide Part可以对一些特例进行memorization。比如AND(query="fried chicken", item="chicken fried rice")虽然从字符角度来看很接近，但是实际上完全不同的东西，那么Wide就可以记住这个组合是不好的，是一个特例，下次当你再点炸鸡的时候，就不会推荐给你鸡肉炒米饭了。

2.3 Deep Part

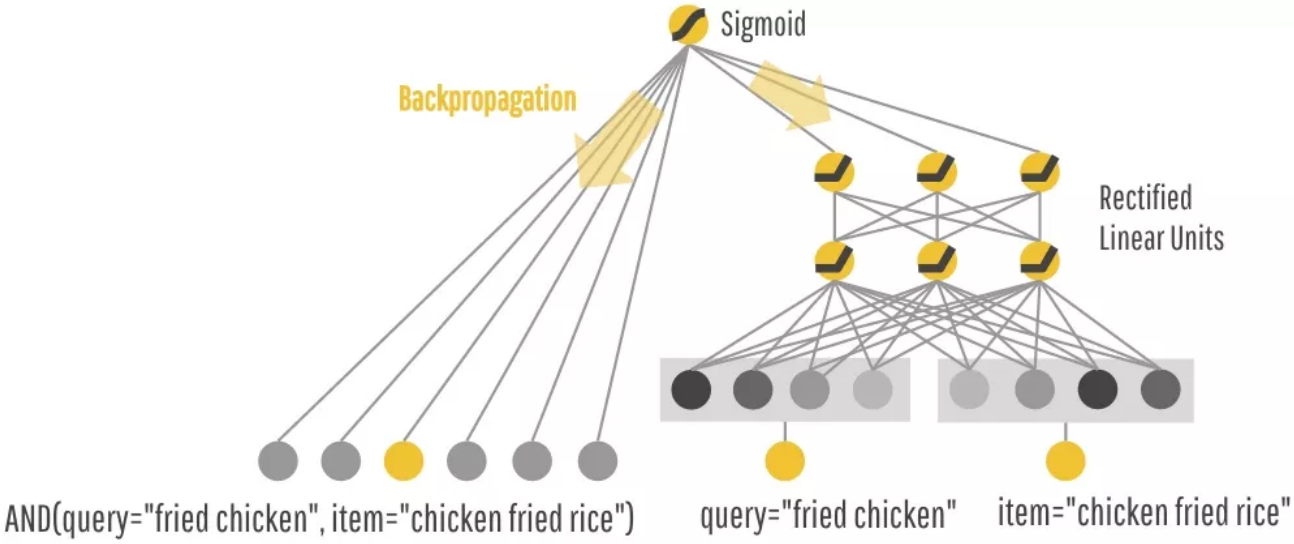


Deep Part通过学习一个低纬度的dense representation（也叫做embedding vector）对于每一个query和item，来**泛化**给你推荐一些字符上看起来不那么相关，但是你可能也是需要的。比如说：你想要炸鸡，Embedding Space中，炸鸡和汉堡很接近，所以也会给你推荐汉堡。

Embedding vectors被随机初始化，并根据最终的loss来反向训练更新。这些低维度的dense embedding vectors被作为第一个隐藏层的输入。隐藏层的激活函数通常使用ReLU。

2.4 模型训练

原始的稀疏特征，在两个组件中都会用到，比如query="fried chicken" item="chicken fried rice"：



在训练的时候，根据最终的loss计算出gradient，反向传播到Wide和Deep两部分中，分别训练自己的参数。也就是说，**两个模块是一起训练的**，注意这不是模型融

合。

- Wide部分中的组合特征可以**记住**那些稀疏的，特定的rules
- Deep部分通过Embedding来**泛化**推荐一些相似的items

Wide模块通过组合特征可以很效率的学习一些特定的组合，但是这也导致了他并不能学习到训练集中没有出现的组合特征。所幸，Deep模块弥补了这个缺点。
另外，因为是一起训练的，wide和deep的size都减小了。wide组件只需要填补deep组件的不足就行了，所以需要比较少的cross-product feature transformations，而不是full-size wide Model。

论文中的实现：

- 训练方法是用mini-batch stochastic optimization。
- Wide组件是用FTRL（Follow-the-regularized-leader） + L1正则化学习。
- Deep组件是用AdaGrad来学习。

3. 系统实现

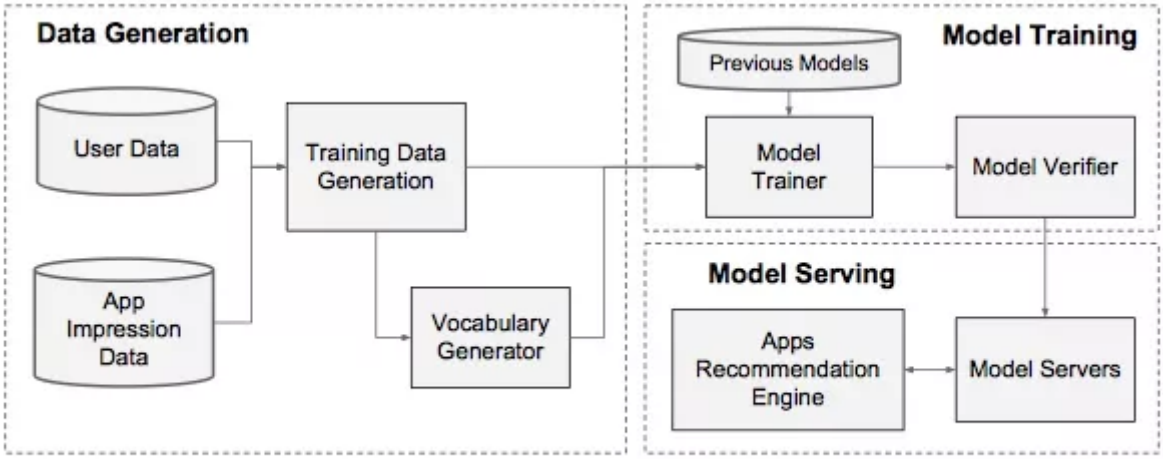


Figure 3: Apps recommendation pipeline overview.

3.1 训练数据生成

请大家一定格外的关注训练数据到底是什么？这对于理解推荐系统到底是怎么回事很重要。
先给出结论：

> 一次展示中的一个Item就是一条样本。

样本的label要根据实际的业务需求来定，比如APP Store中想要提高APP的下载率，那么就以这次展示的这个Item中用户有没有下载，作为label。下载了label为1，否则为0.
说白了，模型需要预测，在当前Query的条件下，对于这个Item，用户下载的条件概率。

> 离散特征map成id

过滤掉出现次数少于设定阈值的离散特征取值，然后把这些全部map成一个ID。离散特征取值少，就直接编号。多的话可能要Hash

> 连续特征通过分位数规范化到[0,1]

先把所有的值分成n份，那么属于第i部分的值规范化之后的值为 (i - 1)/(n - 1)。

3.2 模型训练

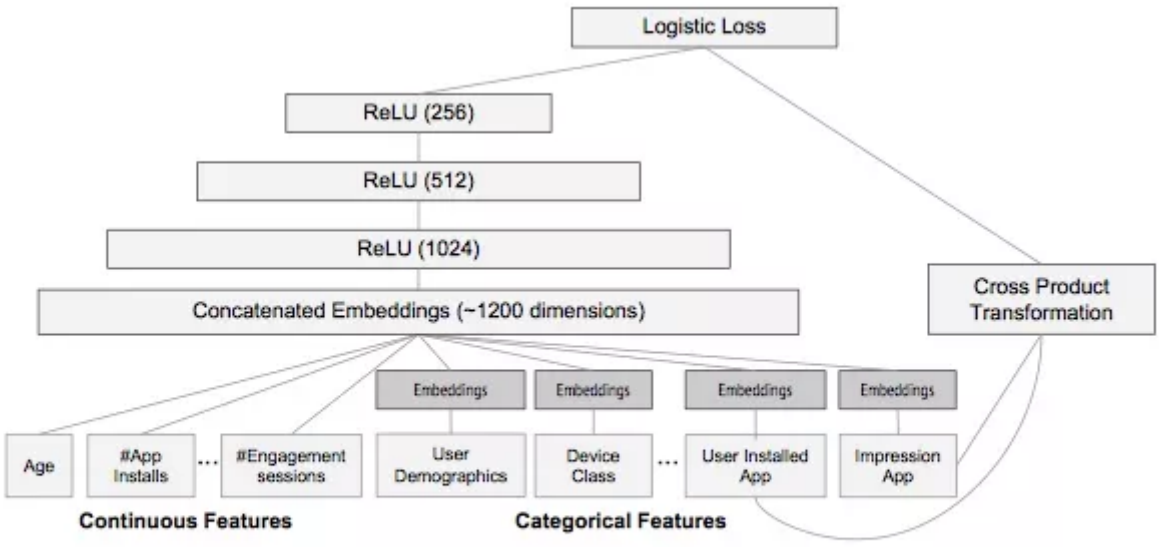


Figure 4: Wide & Deep model structure for apps recommendation.

Deep部分使用的特征：

- 连续特征

- Embedding后的离散特征，Item特征

Wide部分使用的特征：

- Cross Product Transformation生成的组合特征

但是，官方给出的示例代码中，Wide部分还使用了离散特征（没有one-hot）。也有大佬说不用特征交叉效果也很好，这个大家在实际项目中就以实验为准吧。

每当有新的数据到达的时候，就要重新训练。如果每次都从头开始会非常耗时，Google给出的解决办法是：实现了warm-starting system，它可以用之前模型的embeddings 和 线性模型的weights来初始化新的模型。

Embedding维度大小的建议：
Wide&Deep的作者指出，从经验上来讲Embedding层的维度大小可以用如下公式来确定：
$k\sqrt[4]{n}$
n是原始维度上特征不同取值的个数；k是一个常数，通常小于10.

3.3 线上使用

模型被部署之后。每一次请求，服务器会收到一系列的app候选集（从app retrival system输出的）以及user features（用于为每一个app打分）。然后，模型会把APP按照score排序，并展示给user，按照这个顺序展示。score就是对于wide & deep模型的一次 forward pass。为了控制每一次request响应时间在10ms内，引入了并行化技术。将app候选集分成多个小的batches，并行化预测score。

4. 适用范围

Wide & Deep Model适用于输入非常稀疏的大规模分类或回归问题。比如推荐系统、search、ranking问题。
输入稀疏通常是由离散特征有非常非常多个可能的取值造成的，one-hot之后维度非常大。

5. 优缺点

缺点：Wide部分还是需要人为的特征工程。
优点：实现了对memorization和generalization的统一建模。

6.代码实践

> 代码放到github上了：https://github.com/gutouyu/ML_CIA/tree/master/Wide%26Deep
> 数据集：https://archive.ics.uci.edu/ml/machine-learning-databases/adult
> 代码主要包括两部分：Wide Linear Model 和 Wide & Deep Model。

数据集长这样，最后一行是label，预测收入是否超过5万美元，二分类问题。

Column Name	Type	Description
age	Continuous	The age of the individual
workclass	Categorical	The type of employer the individual has (government, military, private, etc.).
fnlwgt	Continuous	The number of people the census takers believe that observation represents (sample weight). Final weight will not be used.
education	Categorical	The highest level of education achieved for that individual.
education_num	Continuous	The highest level of education in numerical form.
marital_status	Categorical	Marital status of the individual.
occupation	Categorical	The occupation of the individual.
relationship	Categorical	Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
race	Categorical	Amer-Indian-Eskimo, Asian-Pac- Islander, Black, White, Other.
gender	Categorical	Female, Male.
capital_gain	Continuous	Capital gains recorded.
capital_loss	Continuous	Capital Losses recorded.
hours_per_week	Continuous	Hours worked per week.
native_country	Categorical	Country of origin of the individual.
income_bracket	Categorical	">50K" or "<=50K", meaning whether the person makes more than $50,000 annually.

6.1 Wide Linear Model

离散特征处理分为两种情况：

- 知道所有的不同取值，而且取值不多。
  tf.feature_column.categorical_column_with_vocabulary_list
- 不知道所有不同取值，或者取值非常多。
  tf.feature_column.categorical_column_with_hash_bucket

```
## 3.1 Base Categorical Feature Columns
# 如果我们知道所有的取值，并且取值不是很多
relationship = tf.feature_column.categorical_column_with_vocabulary_list(
    'relationship', [
        'Husband', 'Not-in-family', 'Wife', 'Own-child', 'Unmarried',
        'Other-relative'
    ]
)
# 如果不知道有多少取值
occupation = tf.feature_column.categorical_column_with_hash_bucket(
    'occupation', hash_bucket_size=1000
)
```

原始连续特征： tf.feature_column.numeric_column

```
# 3.2 Base Continuous Feature Columns
age = tf.feature_column.numeric_column('age')
education_num = tf.feature_column.numeric_column('education_num')
capital_gain = tf.feature_column.numeric_column('capital_gain')
capital_loss = tf.feature_column.numeric_column('capital_loss')
hours_per_week = tf.feature_column.numeric_column('hours_per_week')
```

规范化到[0,1]的连续特征： tf.feature_column.bucketized_column

```
# 3.2.1 连续特征离散化
# 之所以这么做是因为：有些时候连续特征和label之间不是线性的关系。
# bucketization 装桶
# 10个边界，11个桶
age_buckets = tf.feature_column.bucketized_column(
    age, boundaries=[18, 25, 30, 35, 40, 45, 50, 55, 60, 65])
```

组合特征/交叉特征： tf.feature_column.crossed_column

```
# 3.3 组合特征/交叉特征
education_x_occupation = tf.feature_column.crossed_column(
    ['education', 'occupation'], hash_bucket_size=1000)

age_buckets_x_education_x_occupation = tf.feature_column.crossed_column(
    [age_buckets, 'education', 'occupation'], hash_bucket_size=1000
)
```

组装模型：这里主要用了*离散特征 + 组合特征*

```
# 4. 模型
"""
之前的特征:
1. CategoricalColumn
2. NumericalColumn
3. BucketizedColumn
4. CrossedColumn
这些特征都是FeatureColumn的子类，可以放到一起
"""
base_columns = [
    education, marital_status, relationship, workclass, occupation,
    age_buckets,
]

crossed_column = [
    tf.feature_column.crossed_column(
        ['education', 'occupation'], hash_bucket_size=1000
    ),
    tf.feature_column.crossed_column(
        [age_buckets, 'education', 'occupation'], hash_bucket_size=1000
    )
]

model_dir = "./model/wide_component"
model = tf.estimator.LinearClassifier(
    model_dir=model_dir, feature_columns=base_columns + crossed_column
)
```

训练 & 评估：

```
# 5. Train & Evaluate & Predict
model.train(input_fn=lambda: input_fn(data_file=train_file, num_epochs=1, shuffle=True, batch_size=512))
results = model.evaluate(input_fn=lambda: input_fn(val_file, 1, False, 512))
for key in sorted(results):
    print("{0:20}: {1:.4f}".format(key, results[key]))
```

运行截图：

```
Parsing ./data/adult.data
accuracy            : 0.8434
accuracy_baseline   : 0.7592
auc                 : 0.8940
auc_precision_recall: 0.7230
average_loss        : 0.3402
global_step         : 192.0000
label/mean          : 0.2408
loss                : 173.0865
prediction/mean     : 0.2405
```

6.2 Wide & Deep Model

> Deep部分用的特征： 未处理的连续特征 + Embedding(离散特征)

在Wide的基础上，增加Deep部分：

离散特征embedding之后，和连续特征串联。

```
# 3. The Deep Model: Neural Network with Embeddings
"""
1. Sparse Features -> Embedding vector -> 串联(Embedding vector, 连续特征) -> 输入到Hidden Layer
2. Embedding Values随机初始化
3. 另外一种处理离散特征的方法是：one-hot or multi-hot representation. 但是仅仅适用于维度较低的，embedding是更加通用的做法
4. embedding_column(embedding);indicator_column(multi-hot);
"""
deep_columns = [
    age,
    education_num,
    capital_gain,
    capital_loss,
    hours_per_week,
    tf.feature_column.indicator_column(workclass),
    tf.feature_column.indicator_column(education),
    tf.feature_column.indicator_column(marital_status),
    tf.feature_column.indicator_column(relationship),

    # To show an example of embedding
    tf.feature_column.embedding_column(occupation, dimension=8)
]
```

组合Wide & Deep： DNNLinearCombinedClassifier

```
# 4. Combine Wide & Deep
model = tf.estimator.DNNLinearCombinedClassifier(
    model_dir = model_dir,
    linear_feature_columns=base_columns + crossed_columns,
    dnn_feature_columns=deep_columns,
    dnn_hidden_units=[100,50]
)
```

训练 & 评估：

```
for n in range(train_epochs // epochs_per_eval):
    model.train(input_fn=lambda: input_fn(train_file, epochs_per_eval, True, batch_size))
    results = model.evaluate(input_fn=lambda: input_fn(
        test_file, 1, False, batch_size))

    # Display Eval results
    print("Results at epoch {0}".format((n+1) * epochs_per_eval))
    print('-'*30)

    for key in sorted(results):
        print("{0:20}: {1:.4f}".format(key, results[key]))
```

运行结果:

```
Parsing ./data/adult.test
Results at epoch 2
------------------------------------
accuracy            : 0.8486
accuracy_baseline   : 0.7638
auc                 : 0.8931
auc_precision_recall: 0.7550
average_loss        : 0.3385
global_step         : 8145.0000
label/mean          : 0.2362
loss                : 13.5087
prediction/mean     : 0.2346
Parsing ./data/adult.data
Parsing ./data/adult.test
Results at epoch 4
------------------------------------
accuracy            : 0.8457
accuracy_baseline   : 0.7638
auc                 : 0.8955
auc_precision_recall: 0.7421
average_loss        : 0.3433
global_step         : 9774.0000
label/mean          : 0.2362
loss                : 13.6987
prediction/mean     : 0.2429
Parsing ./data/adult.data
Parsing ./data/adult.test
Results at epoch 6
------------------------------------
accuracy            : 0.8476
accuracy_baseline   : 0.7638
auc                 : 0.8954
auc_precision_recall: 0.7495
average_loss        : 0.3367
global_step         : 11403.0000
label/mean          : 0.2362
loss                : 13.4349
prediction/mean     : 0.2404
```

Reference

1. Wide & Deep Learning for Recommender Systems
2. Google AI Blog Wide & Deep Learning: Better Together with TensorFlow https://ai.googleblog.com/2016/06/wide-deep-learning-better-together-with.html
3. TensorFlow Linear Model Tutorialhttps://www.tensorflow.org/tutorials/wide
4. TensorFlow Wide & Deep Learning Tutorialhttps://www.tensorflow.org/tutorials/wide_and_deep
5. TensorFlow 数据集和估算器介绍 http://developers.googleblog.cn/2017/09/tensorflow.html
6. absl https://github.com/abseil/abseil-py/blob/master/smoke_tests/sample_app.py