

在个性化推荐实战项目中，对你有用的算法

原创 Devin Jiang 数据人说 1月26日

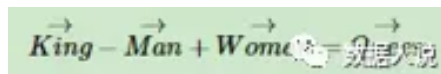
背景

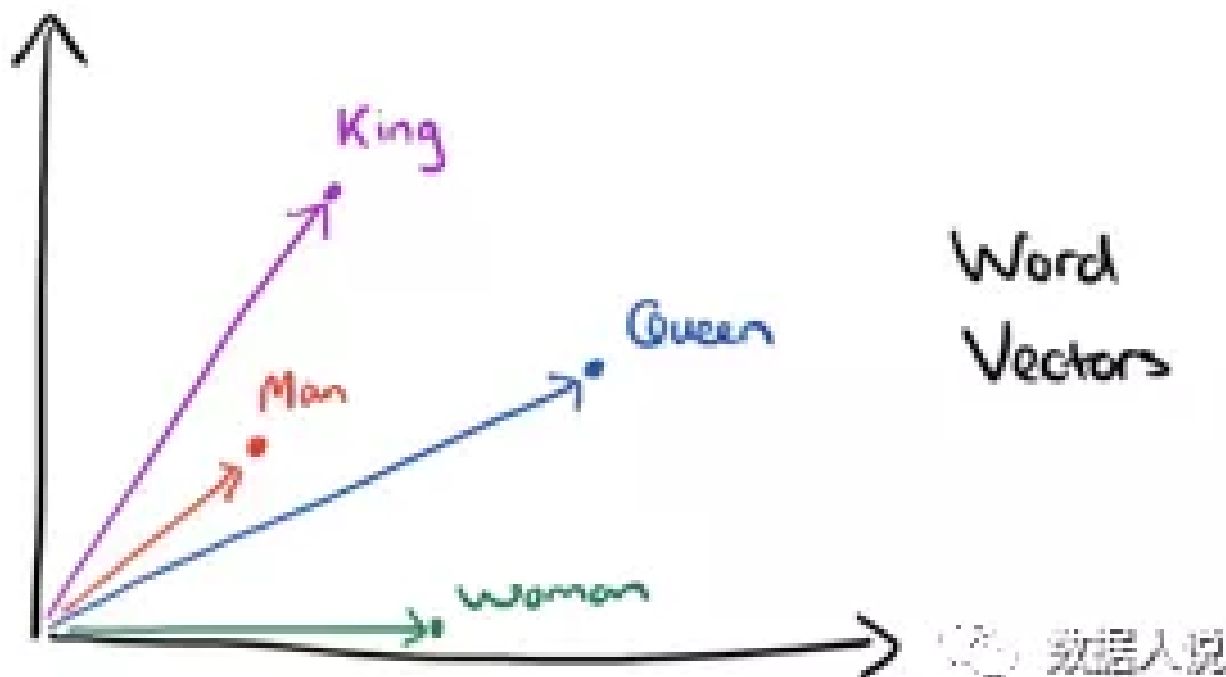
最近，梳理了这毕业后一年半做的一些基础性的工作，按照项目分成个性化推荐、文本处理及精准营销画像这三部分，一方面是想形成自己的知识体系；另一方面是培养自己经常思考并持续输出的习惯。本节主要梳理了本人在个性化推荐中应用到的几个算法，阐述的内容都直截了当，直奔主题。

1.Word2vec

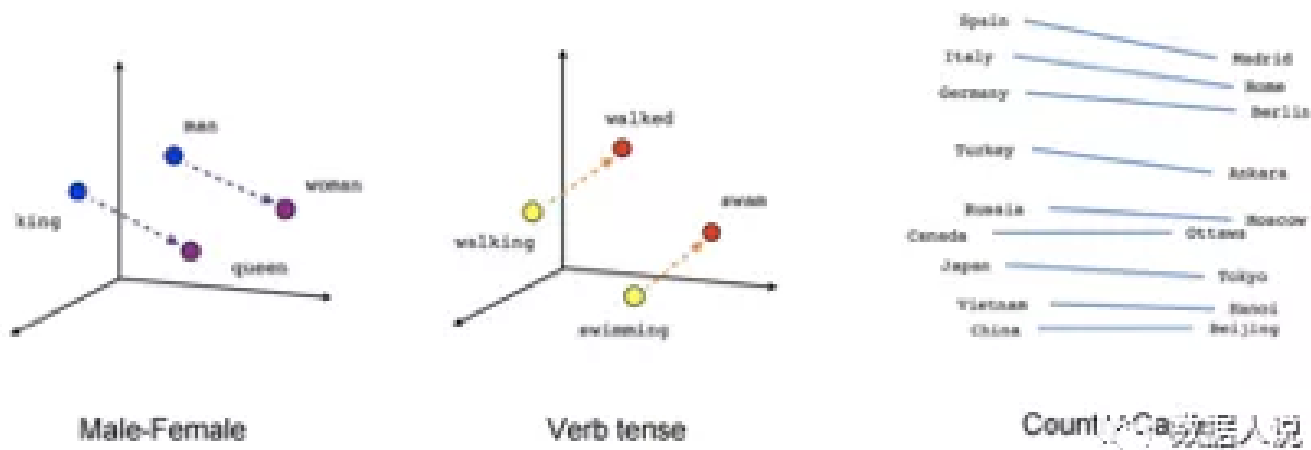
首先需要介绍词的独热表示（One-hot representation）和词分布式表示（Distributed representation）：

- 独热表示是词的一种符号化表式，词袋模型的一种，任意词之间是孤立的，每个样本中的单个特征只有1位处于状态1，其他的都是0，比如一句话有10个词，灯泡这个词出现在第六个位置，则灯泡的独热表式:[0,0,0,0,0,1,0,0,0,0]。
- 词的分布式表示是一种低维实数向量，长度自己定义，灯泡的词分布式表式:[0.234,-0.432,0.345,-0.543,...],其向量表式让相关或相似的词在距离上更接近。看到一个例子，比如我们将词的维度降维到2维，用下图的词向量表示我们的词时，我们可以发现为了得到一个词的一个向量，在训练过程中，同时考虑了它左右的上下文，即语义关系。


$$\vec{King} - \vec{Man} + \vec{Woman} = \vec{Queen}$$

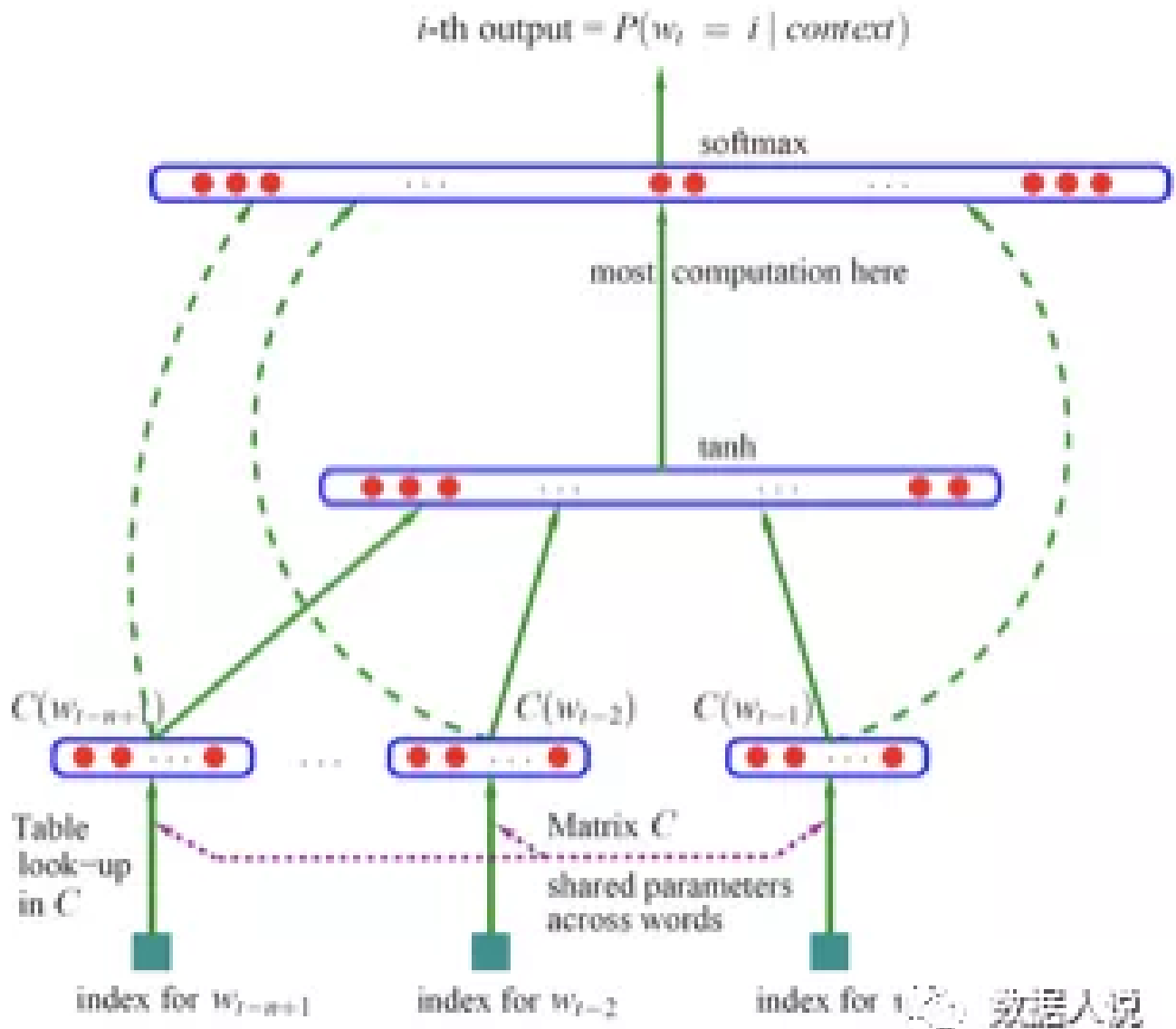


下图是部分二词向量的效果图，从图中可以得到king的相似词是queen,walked的相似词是walking等关系。



另外，词向量的思想来源是基于统计语言模型的，其假设词的序列看作一个随机事件，并赋予x相应的概率来描述其属于某种词的集合的可能性。这里涉及到的N-Gram求解词概率的方式，word2vec用CBOW和skip-gram两种神经网络来求解词的概率，由于训练语料限制，随着N的增大导致计算量也会变大，目前使用较多的是三元模型（假设出现的概率与前面两个词相关）。

Word2vec是通过一个三层神经网络得出，由约书亚.本吉奥（Yoshua Bengio）提出的，word2vec是实现词向量的一种工具。word2vec模型神经网络雏形：



- 通过窗口输入句子中的连续三个词， w_1, w_2, w_3
- 输入网络中已是随机初始化的向量，如 $w_1: [0, 0, 0, 0, \dots, 0]$ ，值的向量长度自定义，三个词向量，输入到网络中。
- 目标值为原句子的后面一个 w_4 ，通过onehot编码定义
- 神经网络训练过程的目的是计算交叉熵损失，网络参数更新，自动调整 w_1, w_2, w_3 的向量值，达到经过最后的softmax输出预测概率，与目标值计算损失，让预测值与真实值更接近。其中词向量、权重和偏置都是网络参数。

在实际的工作中，使用python的gensim及spark的Word2Vec训练模型是非常快捷的，熟悉下API的就可以快速上手，如spark Word2Vec模型的API及调用：

```
1 模块: from pyspark.ml.feature import Word2Vec
2 API: class pyspark.ml.feature.Word2Vec(vectorSize=100, minCount=5, numPartiti
3 vectorSize=100: 词向量长度
4 minCount: 过滤次数小于默认5次的词
```

```
5  windowSize=5: 训练时候的窗口大小
6  inputCol=None: 输入列名
7  outputCol=None: 输出列名
8
9  #model train
10 new_word2Vec = Word2Vec(vectorSize=100, inputCol="words", outputCol="model", mi
11 model = new_word2Vec.fit(words_df) # words_df分词后的词语集
12
13 ...
14 ...
15 #model predict
16 def predict_proba(oword, iword):
17     iword_vec = model[iword]
18     oword = model.wv.vocab[oword]
19     oword_l = model.syn1[oword.point].T
20     dot = np.dot(iword_vec, oword_l)
21     lprob = -sum(np.logaddexp(0, -dot) + oword.code*dot)
22     return lprob
23
24 #get the top20 most common words
25 def relative_words_adjust(word):
26     r = {i:predict_proba(i, word)-0.9*np.log(j.count) for i,j in model.wv.vocab}
    return Counter(r).most_common(20)
```

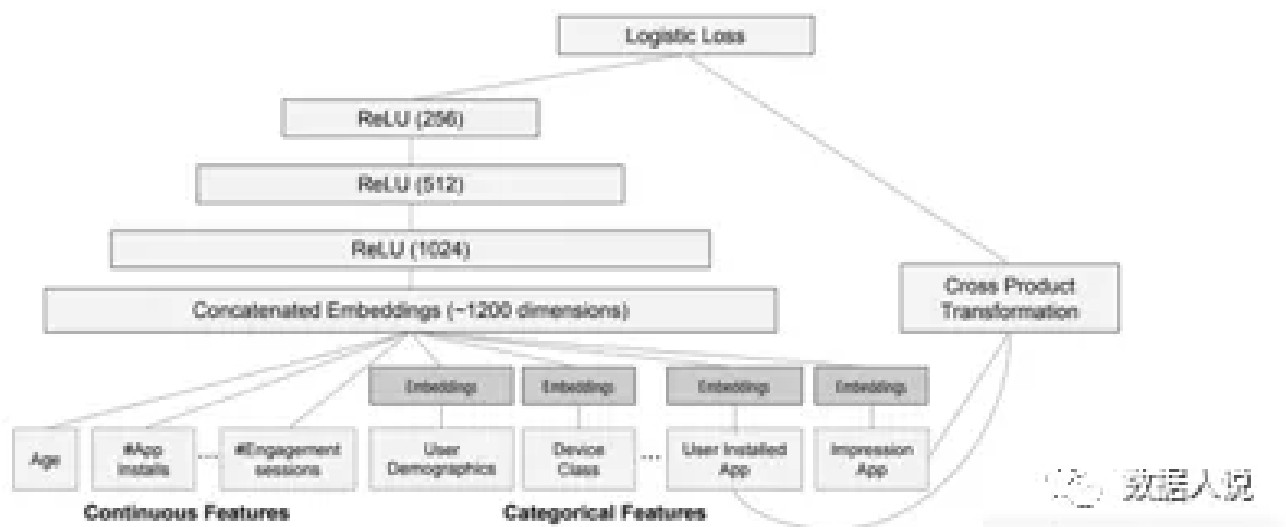
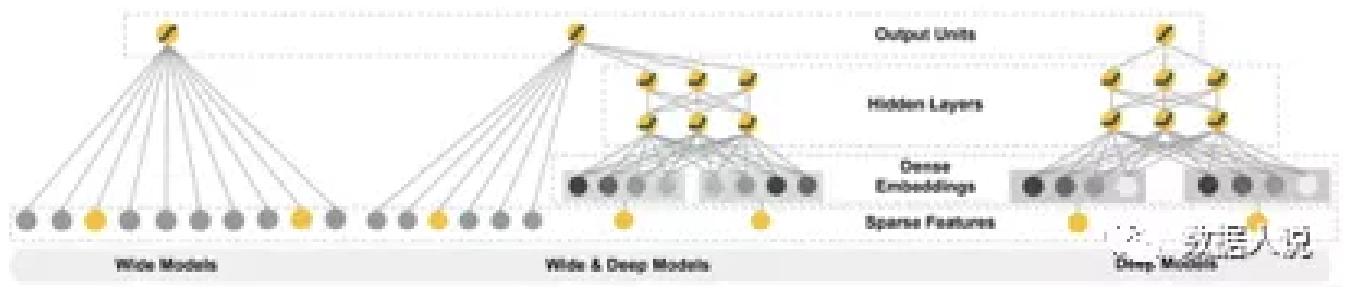
最后，word2vec这部分神经网络训练过程推导及hierarchicalsoftmax/negative sampling等trick建议看论文和《动手学深度学习》第十六课可以慢慢熟悉，初级算法工程师面试会问到的几个点，也请读者一起思考：

- 词向量来源思想，统计语言模型相关方面
- Word2vec输入层、隐藏层、输出层具体的形式，一般都需要手写出来
- Word2vec中心词和背景词向量区别，相似度评估理论依据
- Word2Vec受语料库的影响，如何优化相似词都是热点词汇的问题

2.Wide&Deep model

Wide&Deep model包括两部分：线性模型(Wide Linear Model) + DNN(Deep Netural Network，旨在使得训练得到的模型能够同时获得记忆(memorization)和泛化(generalization)能力。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 model 刚好结合了这部分，如图1是原论文模型网络图，图2是实验中采用的模型结构图



定义， $y = W^T x + b$ 其中 y 表示预测值， $x = [x_1, x_2, \dots, x_d]$ 表示 d 个特征， $w = [w_1, w_2, \dots, w_d]$ 表示模型参数， b 表示偏移项，在二分类任务中，最终模型采用联合训练得到模型的预测值公式如下，其中 $\phi_k(x)$ 为特征交叉变换， $a(l)$ 表示 l 层的输出， f 为激活函数，一般为 ReLU 函数。

$$P(Y = 1|x) = \sigma(w_{wide}^T [x, \phi(x)] + w_{deep}^T a(l))$$

Wide部分的输入特征：

- raw input features and transformed features
- W&D这里的cross-product transformation：只在离散特征之间做组合，不管是文本策略型的，还是离散值的；与连续值特征无关，在W&D的paper里面是这样使用的。

特别，举一个交叉特征的例子：特征

AND(user_installed_category=netflix, impression_app=pandora), 值为1表示用户

安装了netflix，喜欢的app为Pandora. 此特征可以解释这一对原始特征与目标标签的相关性，这样人工做交叉局限是需要具备有业务背景并且也没法提取出历史数据中未出现的特征组合。

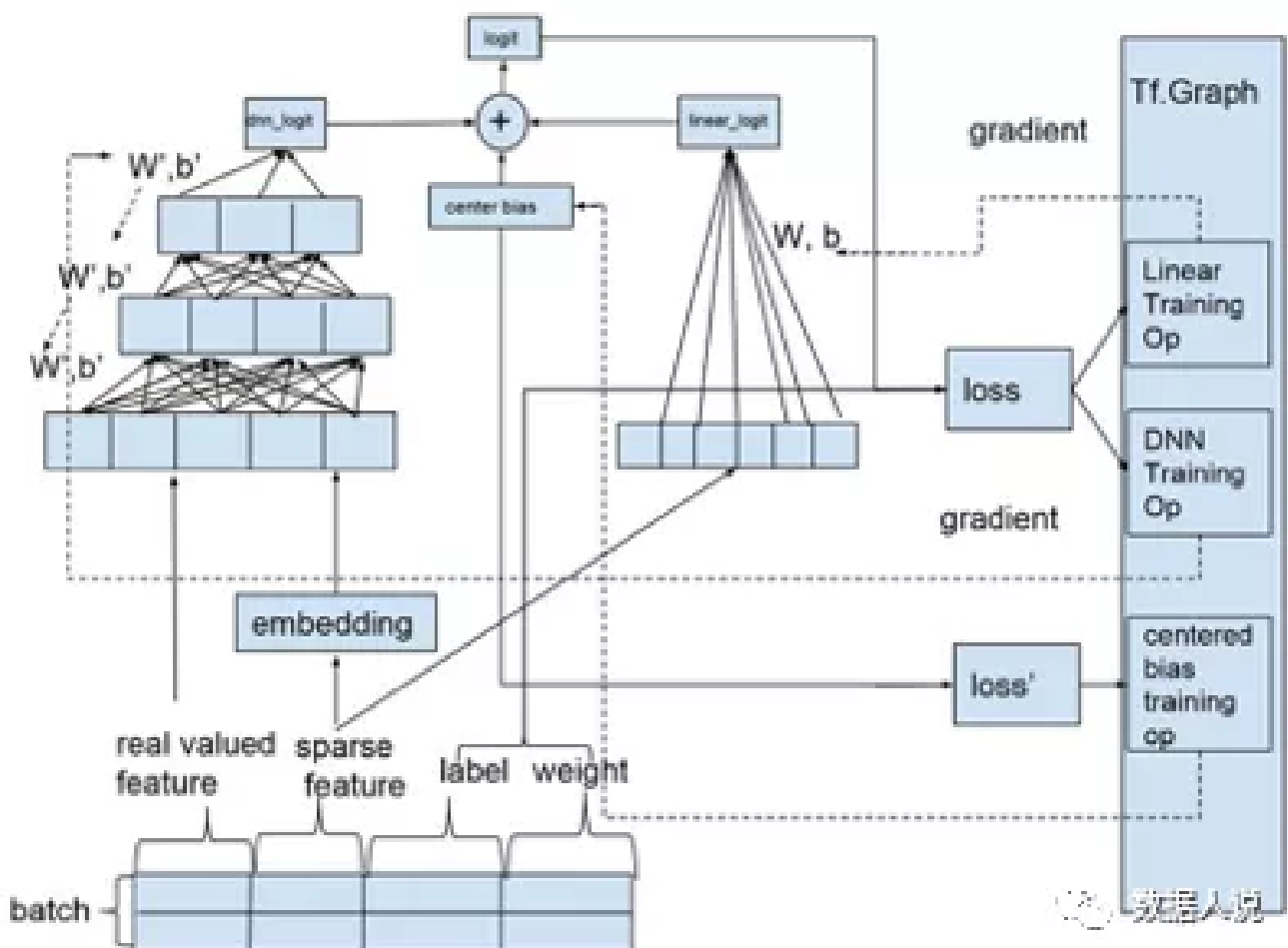
Deep部分的输入特征：

- raw input+embedding处理
- 对非连续值之外的特征做embedding处理，这里都是策略特征，乘以个embedding-matrix。在TensorFlow里面的接口是：`tf.feature_column.embedding_column`，默认trainable=True.
- 对连续值特征的处理是：将其按照累积分布函数 $P(X \leq x)$ ，压缩至[0,1]内。

其中模型训练的核心：

- 模型训练方法是用mini-batch stochastic optimization
- Wide部分是用FTRL (Follow-the-regularized-leader) + L1正则化学习。
- Deep部分是用AdaGrad来学习
- 在Deep模型中需要将稀疏矩阵进行embedding，Wide&Deep的作者指出，从经验上来讲Embedding层的维度大小可以用公式来确定：
- 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.

我以前只用过，现在正在分析这方面的源码，但看到一前辈基于TensorFlow中W&D模型源码深入分析总结一张图，模型训练过程特别直观



在工作中调用TensorFlow里面的API，其实现baseline如下：

```

1  def build_estimator(self):
2      """建立模型
3      :param dataset:
4      :return:
5      """
6      #离散分类
7      article_id = tf.feature_column.categorical_column_with_identity('char
8      #连续类型
9      vector = tf.feature_column.numeric_column('vector')
10     user_weights = tf.feature_column.numeric_column('user_weights')
11     article_weights = tf.feature_column.numeric_column('article_weights')
12
13     wide_columns = [article_id]
14
15     # embedding_column用来表示类别型的变量
16     deep_columns = [tf.feature_column.embedding_column(article_id, dimensi
17                     vector, user_weights, article_weights)]
18
19     estimator = tf.estimator.DNNLinearCombinedClassifier(model_dir="./jdz
20                                                         linear_feature_c
21                                                         dnn_feature_colu
22                                                         dnn_hidden_units
23
24     return estimator

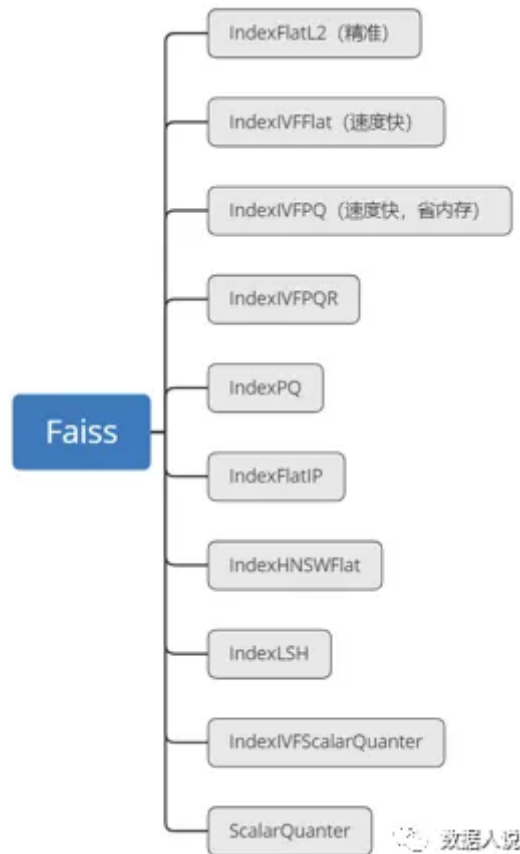
```

初级算法工程师面试会问到的几个点：

- 模型单机和分布式部署
- Deep部分针对一些特征梯度消失或爆炸的一些优化算法和正则化算法
- W&D模型实际给业务带来的提升分析

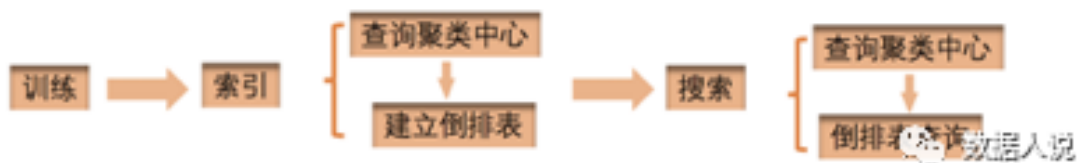
3.Faiss

Faiss是Facebook AI团队开源的针对聚类和相似性搜索库，为稠密向量提供高效相似度搜索和聚类，支持十亿级别向量的搜索，是目前较为成熟的近似近邻搜索库。下面是Faiss类库方法集合



Faiss类库方法集合

我们使用Faiss计算用户之前的相似度，最新baseline模型采用倒排表索引GpuIndexIVFFlat，其大致原理是首先将数据库向量通过聚类方法分割成若干子类，每个子类用类中心表示，当查询向量来临，选择距离最近的类中心，然后在子类中应用精确查询方法，通过增加相邻的子类个数提高索引的精确度。测试结果，data:2000万用户*82维特征，query:40万用户*82维特征(rfm)，每个用户取top50,检索时间：32s,准确率0.79，整个流程和重要的代码如下：



```

1 nlist=256
2 k = 10
3 res = [faiss.StandardGpuResources() for i in range(ngpus)]
4 flat_config = []
5 for i in range(ngpus):
6     cfg = faiss.GpuIndexIVFFlatConfig() #faiss.GpuIndexFlatConfig() faiss.
7     cfg.useFloat16 = False
8     cfg.device = i
  
```

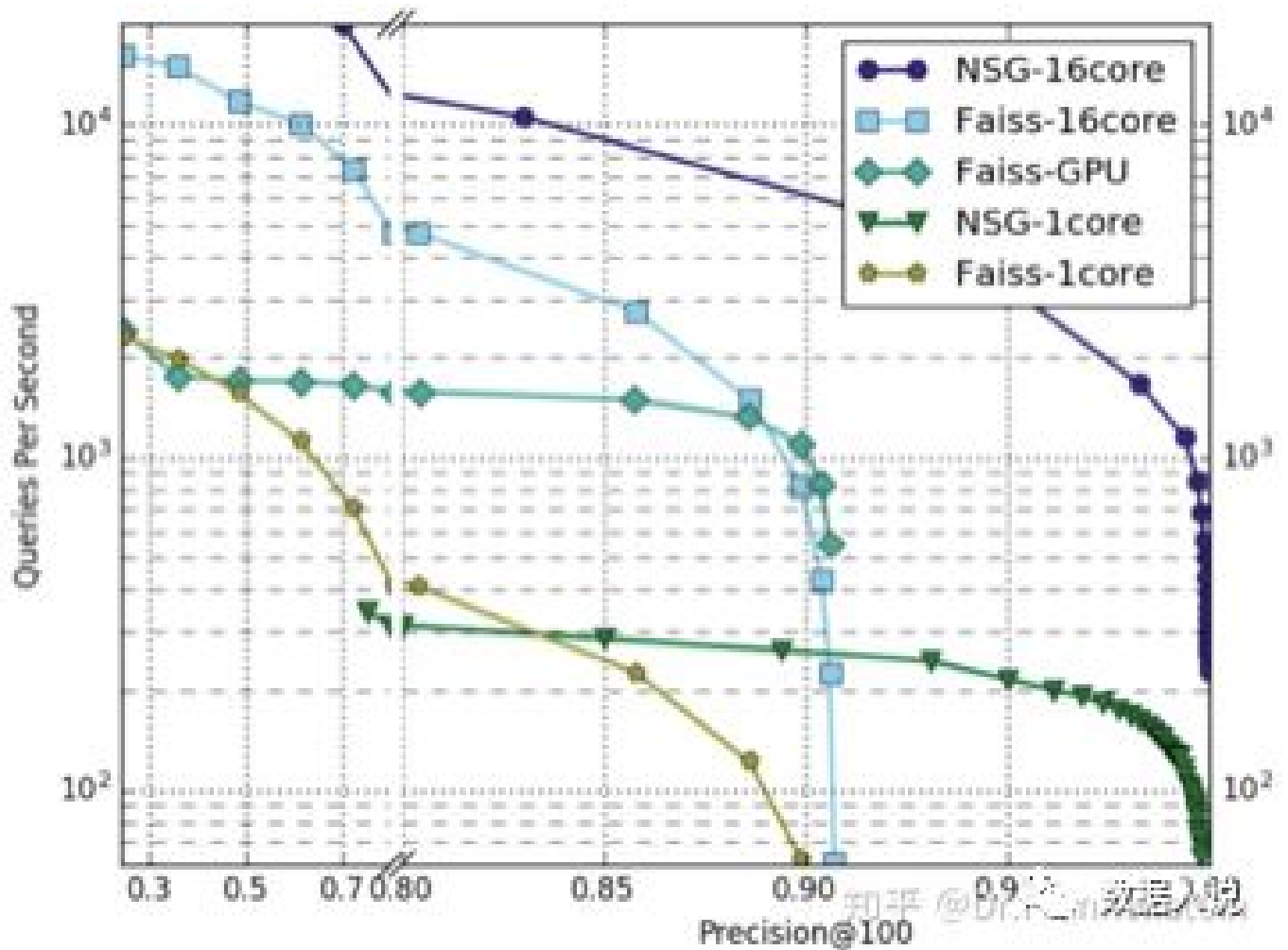


```
9         flat_config.append(cfg)
10     #indexes = [faiss.GpuIndexFlatL2(res[i],d,flat_config[i]) for i in range(n)]
11     #indexes = [faiss.GpuIndexIVFPQ(res[i],d,nlist, m,4,faiss.METRIC_L2,flat_config[i]) for i in range(n)]
12     indexes = [faiss.GpuIndexIVFFlat(res[i],d,nlist,faiss.METRIC_L2,flat_config[i]) for i in range(n)]
13
14     index = faiss.IndexProxy()
15     for sub_index in indexes:
16         index.addIndex(sub_index)
```

应用过程中问题总结和思考如下：

- 速度与nlist和nprobe相关，较暴力搜索量级速度提高
- nprobe 越大，召回效果越好,速度越慢
- nlist与nprobe配合使用，效果才最佳
- GpuIndexIVFPQ理论上乘积向量编码后速度要快，但实际train过程非常慢，最后选择GpuIndexIVFFlat
- Faiss分布式部署及大数据集群里的数据如何跟GPU服务器之间交互

最后，这类高维数据检索工业级解决方案，阿里的NSG模型论文中有个实验结果图，因为我个人这方面实践比理论少，所以感兴趣的同学我们可以一起深入实践学习。



4.DeepFM

正如W&D模型结合了DNN和LR，DeepFM结合了DNN和FM，FM算法负责对一阶特征以及由一阶特征两两组合而成的二阶特征进行特征的提取；DNN算法负责对由输入的一阶特征进行全连接等操作形成的高阶特征进行特征的提取。下面是论文中DeepFM并行结构

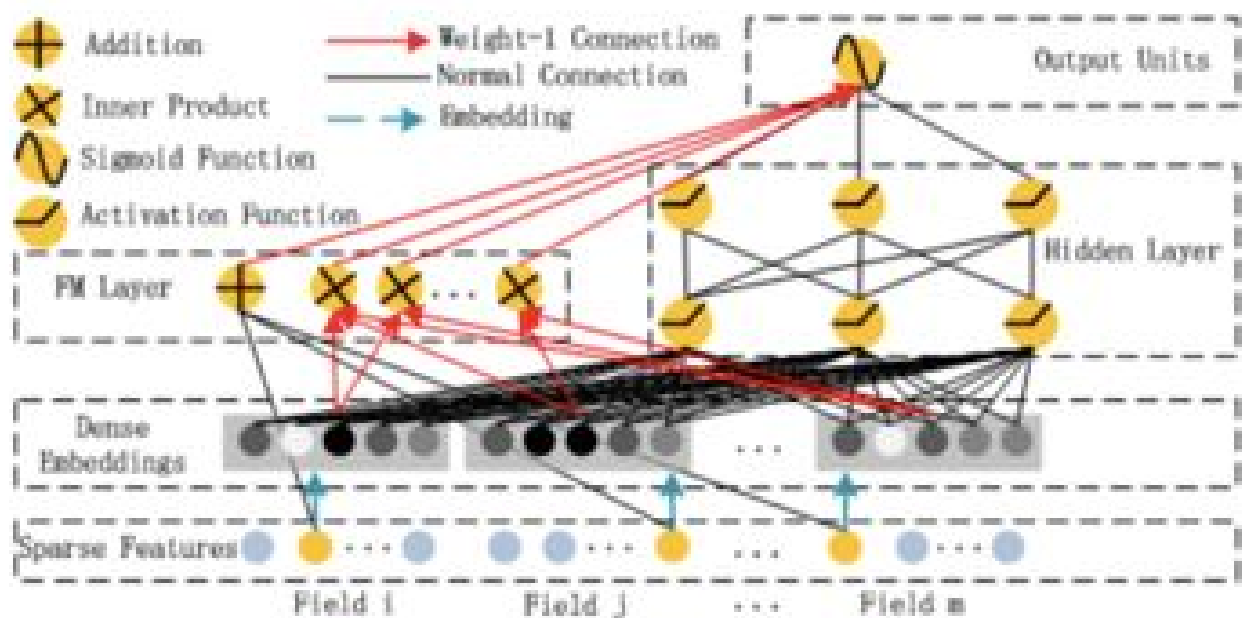


Figure 1: Wide & deep architecture of DeepFM. The wide and deep component share the same input raw feature vector, which enables DeepFM to learn low- and high-order feature interactions simultaneously from the input raw features.

其中，DeepFM的输入可由连续型变量和类别型变量共同组成，且类别型变量需要进行One-Hot编码。而正由于One-Hot编码，导致了输入特征变得高维且稀疏。模型最终的输出也由这两部分组成： $\hat{y} = \text{sigmoid}(y_{FM} + y_{DNN})$ ，具体的模型细节看原论文就好，具有以下特点：

- FM通过隐向量latent vector做内积来表示组合特征，从理论上解决了低阶和高阶组合特征提取的问题
- DeepFM是端到端的训练，不需要人工特征工程。
- FM和Deep共享输入和feature embedding不但使得训练更快，而且使得训练更加准确。
- DeepFM共享feature embedding 这个特性使得在反向传播的时候，模型学习feature embedding，而后又会在前向传播的时候影响低阶和高阶特征的学习，这使得学习更加的准确。

深度学习模型调优：优化函数和学习率、网络的深度和宽度、网络结构选型、隐藏层后添加dropout层和BN层、激活函数、正则化系数；论文的第3.3节介绍了部分重要的超参数的调参结果，下面是前辈总结的一个表格，针对深度学习模型调参，我也只是借鉴论文和前人的一些经验，然后在具体的业务中各种尝试。

超参数	建议	备注
激活函数	1. IPNN使用tanh 2. 其余使用ReLU	
学习方法	Adam	
Dropout	0.6~0.9	
隐藏层数量	3~5	根据实际数据大小调整
神经元数量	200~600	根据实际数据大小调整
网络形状	constant	一共有四种：固定、增长、下降、菱形。 

最后，针对DeepFM模型在猜你喜欢点击率预测中得到的一点结果：

- 由于正负样本不平衡，使用论文Focal Loss损失函数，用原始正负样本1：50的数据训练，AUC提升很明显，在本场景上能很好解决类极不平衡的问题和挖掘难分样本
- 提升模型效果简单并有效的方式添加有效的特征
- 控制好Epoch/Early Stopping，防止模型过拟合和欠拟合

5.总结

本文主要梳理了一些自己在个性化推荐场景中应用的一些算法，希望能给一些同学一点参考性的建议。其实还有网络表式学习模型DeepWalk、基于图结构的实时推荐算法swing、及正想尝试的多任务学习模型ESMM/ESM2，这几个模型我放在后面分享。

最后，水平有限，如有错误，请指正，并且感谢前辈们分享的文章和经验。

6.参考资料

[1] Mikolov T, Chen K, Corrado G, et al. Efficient Estimation of Word Representations in Vector Space[J]. Computer Science, 2013.