DIN

动机

Deep Interest Network(DIIN)是2018年阿里巴巴提出来的模型， 该模型基于业务的观察，从实际应用的角度进行改进，相比于之前很多“学术风”的深度模型， 该模型更加具有业务气息。该模型的应用场景是阿里巴巴的电商广告推荐业务， 这样的场景下一般**会有大量的用户历史行为信息**， 这个其实是很关键的，因为DIN模型的创新点或者解决的问题就是使用了注意力机制来对用户的兴趣动态模拟， 而这个模拟过程存在的前提就是用户之前有大量的历史行为了，这样我们在预测某个商品广告用户是否点击的时候，就可以参考他之前购买过或者查看过的商品，这样就能猜测出用户的大致兴趣来，这样我们的推荐才能做的更加到位，所以这个模型的使用场景是**非常注重用户的历史行为特征（历史购买过的商品或者类别信息）**，也希望通过这一点，能够和前面的一些深度学习模型对比一下。

在个性化的电商广告推荐业务场景中，也正式由于用户留下了大量的历史交互行为，才更加看出了之前的深度学习模型(作者统称Embeding&MLP模型)的不足之处。如果学习了前面的各种深度学习模型，就会发现Embeding&MLP模型对于这种推荐任务一般有着差不多的固定处理套路，就是大量稀疏特征先经过embedding层， 转成低维稠密的，然后进行拼接，最后喂入到多层神经网络中去。

这些模型在这种个性化广告点击预测任务中存在的问题就是**无法表达用户广泛的兴趣**，因为这些模型在得到各个特征的embedding之后，就蛮力拼接了，然后就各种交叉等。这时候根本没有考虑之前用户历史行为商品具体是什么，究竟用户历史行为中的哪个会对当前的点击预测带来积极的作用。 而实际上，对于用户点不点击当前的商品广告，很大程度上是依赖于他的历史行为的，王喆老师举了个例子

> 假设广告中的商品是键盘， 如果用户历史点击的商品中有化妆品， 包包，衣服， 洗面奶等商品， 那么大概率上该用户可能是对键盘不感兴趣的， 而如果用户历史行为中的商品有鼠标， 电脑，iPad，手机等， 那么大概率该用户对键盘是感兴趣的， 而如果用户历史商品中有鼠标， 化妆品， T-shirt和洗面奶， 鼠标这个商品embedding对预测“键盘”广告的点击率的重要程度应该大于后面的那三个。

这里也就是说如果是之前的那些深度学习模型，是没法很好的去表达出用户这广泛多样的兴趣的，如果想表达的准确些， 那么就得加大隐向量的维度，让每个特征的信息更加丰富， 那这样带来的问题就是计算量上去了，毕竟真实情景尤其是电商广告推荐的场景，特征维度的规模是非常大的。 并且根据上面的例子， 也**并不是用户所有的历史行为特征都会对某个商品广告点击预测起到作用。**所以对于当前某个商品广告的点击预测任务，没必要考虑之前所有的用户历史行为。

这样， DIN的动机就出来了，在业务的角度，我们应该自适应的去捕捉用户的兴趣变化，这样才能较为准确的实施广告推荐；而放到模型的角度， 我们应该**考虑到用户的历史行为商品与当前商品广告的一个关联性**，如果用户历史商品中很多与当前商品关联，那么说明该商品可能符合用户的品味，就把该广告推荐给他。而一谈到关联性的话， 我们就容易想到“注意力”的思想了， 所以为了更好的从用户的历史行为中学习到与当前商品广告的关联性，学习到用户的兴趣变化， 作者把注意力引入到了模型，设计了一个"local activation unit"结构，利用候选商品和历史问题商品之间的相关性计算出权重，这个就代表了对于当前商品广告的预测，用户历史行为的各个商品的重要程度大小， 而加入了注意力权重的深度学习网络，就是这次的主角DIN， 下面具体来看下该模型。

DIN模型结构及原理

在具体分析DIN模型之前， 我们还得先介绍两块小内容，一个是DIN模型的数据集和特征表示， 一个是上面提到的之前深度学习模型的基线模型， 有了这两个， 再看DIN模型，就感觉是水到渠成了。

特征表示

工业上的CTR预测数据集一般都是`multi-group categorial form`的形式，就是类别型特征最为常见，这种数据集一般长这样：

Category	Feature Group	Dimemsionality	Type	#Nonzero Ids per Instance
User Profile Features	gender	2	one-hot	1
	age_level	~ 10	one-hot	1
	...	...	...	...
User Behavior Features	visited_goods_ids	~ 10^9	multi-hot	~ 10^3
	visited_shop_ids	~ 10^7	multi-hot	~ 10^3
	visited_cate_ids	~ 10^4	multi-hot	~ 10^2
Ad Features	goods_id	~ 10^7	one-hot	1
	shop_id	~ 10^5	one-hot	1
	cate_id	~ 10^4	one-hot	1
	...	...	...	...
Context Features	pid	~ 10	one-hot	1
	time	~ 10	one-hot	1
	...	...	...	...

这里的亮点就是框出来的那个特征，这个包含着丰富的用户兴趣信息。

对于特征编码，作者这里举了个例子：`[weekday=Friday, gender=Female, visited_cate_ids={Bag,Book}, ad_cate_id=Book]`， 这种情况我们知道一般是通过one-hot的形式对其编码， 转成系数的二值特征的形式。但是这里我们会发现一个`visted_cate_ids`， 也就是用户的历史商品列表， 对于某个用户来讲，这个值是个多值型的特征， 而且还要知道这个特征的长度不一样长，也就是用户购买的历史商品个数不一样多，这个显然。这个特征的话，我们一般是用到multi-hot编码，也就是可能不止1个1了，有哪个商品，对应位置就是1， 所以经过编码后的数据长下面这个样子：

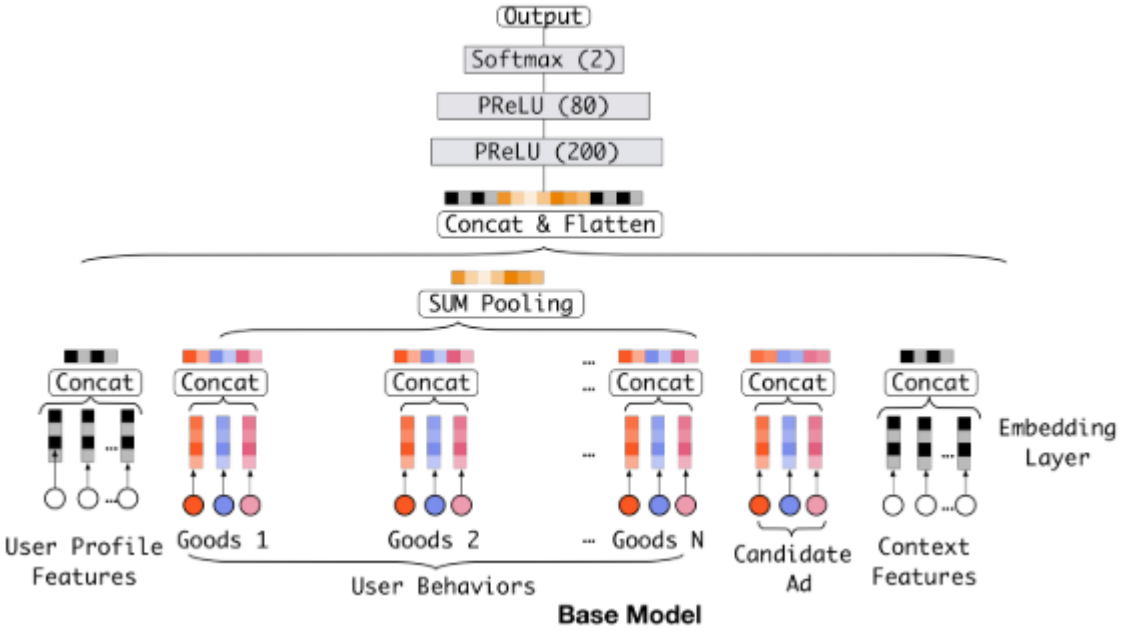
$$\underbrace{[0,0,0,0,1,0,0]}_{\text{weekday=Friday}}\quad\underbrace{[0,1]}_{\text{gender=Female}}\quad\underbrace{[0,..,1,...,1,...0]}_{\text{visited\_cate\_ids=\{Bag,Book\}}}\quad\underbrace{[0,..,1,...,0]}_{\text{ad\_cate\_id=Book}}$$

这个就是喂入模型的数据格式了，这里还要注意一点 就是上面的特征里面没有任何的交互组合，也就是没有做特征交叉。这个交互信息交给后面的神经网络去学习。

基线模型

这里的base 模型，就是上面提到过的Embedding&MLP的形式， 这个之所以要介绍，就是因为DIN网络的基准也是他，只不过在这个的基础上添加了一个新结构(注意力网络)来学习当前候选广告与用户历史行为特征的相关性，从而动态捕捉用户的兴趣。

基准模型的结构相对比较简单，我们前面也一直用这个基准， 分为三大模块：Embedding layer，Pooling & Concat layer和MLP， 结构如下：



前面的大部分深度模型结构也是遵循着这个范式套路， 简介一下各个模块。

1. **Embedding layer**：这个层的作用是把高维稀疏的输入转成低维稠密向量， 每个离散特征下面都会对应着一个embedding词典， 维度是$D\times K$， 这里的D表示的是隐向量的维度， 而K表示的是当前离散特征的唯一取值个数, 这里为了好理解，这里举个例子说明，就比如上面的weekday特征：

> 假设某个用户的weekday特征就是周五，化成one-hot编码的时候，就是[0,0,0,0,1,0,0]表示，这里如果再假设隐向量维度是D， 那么这个特征对应的embedding词典是一个$D\times7$的一个矩阵(每一列代表一个embedding，7列正好7个embedding向量，对应周一到周日)，那么该用户这个one-hot向量经过embedding层之后会得到一个$D\times1$的向量，也就是周五对应的那个embedding，怎么算的，其实就是$embedding矩阵* [0,0,0,0,1,0,0]^T$ 。其实也就是直接把embedding矩阵中one-hot向量为1的那个位置的embedding向量拿出来。 这样就得到了稀疏特征的稠密向量了。

其他离散特征也是同理，只不过上面那个`multi-hot`编码的那个，会得到一个`embedding`向量的列表，因为他开始的那个`multi-hot`向量不止有一个是`1`，这样乘以`embedding`矩阵，就会得到一个列表了。通过这个层，上面的输入特征都可以拿到相应的稠密`embedding`向量了。

2. **pooling layer and Concat layer**： pooling层的作用是将用户的历史行为embedding这个最终变成一个定长的向量，因为每个用户历史购买的商品数是不一样的， 也就是每个用户multi-hot中1的个数不一致，这样经过embedding层，得到的用户历史行为embedding的个数不一样多，也就是上面的embedding列表t_i不一样长， 那么这样的话，每个用户的历史行为特征拼起来就不一样长了。 而后面如果加全连接网络的话，我们知道，他需要定长的特征输入。 所以往往用一个pooling layer先把用户历史行为embedding变成固定长度(统一长度)，所以有了这个公式：

$$e_i = pooling(e_{i1}, e_{i2}, ...e_{ik})$$

这里的e_{ij}是用户历史行为的那些embedding。e_i就变成了定长的向量， 这里的i表示第i个历史特征组(是历史行为，比如历史的商品id，历史的商品类别id等)， 这里的k表示对应历史特种组里面用户购买过的商品数量，也就是历史embedding的数量，看上面图里面的user behaviors系列，就是那个过程了。 Concat layer层的作用就是拼接了，就是把这所有的特征embedding向量，如果再有连续特征的话也算上，从特征维度拼接整合，作为MLP的输入。

3. **MLP**：这个就是普通的全连接，用了学习特征之间的各种交互。

4. **Loss**: 由于这里是点击率预测任务， 二分类的问题，所以这里的损失函数用的负的log对数似然：

$$L = -\frac{1}{N} \sum_{(\boldsymbol{x},y)\in\mathcal{S}} (y \log p(\boldsymbol{x}) + (1-y) \log (1-p(\boldsymbol{x})))$$

这就是base 模型的全貌， 这里应该能看出这种模型的问题， 通过上面的图也能看出来， 用户的历史行为特征和当前的候选广告特征在全都拼起来给神经网络之前，是一点交互的过程都没有， 而拼起来之后给神经网络，虽然是有了交互了，但是原来的一些信息，比如，每个历史商品的信息会丢失了一部分，因为这个与当前候选广告商品交互的是池化后的历史特征embedding， 这个embedding是综合了所有的历史商品信息， 这个通过我们前面的分析，对于预测当前广告点击率，并不是所有历史商品都有用，综合所有的商品信息反而会增加一些噪声性的信息，可以联想上面举得那个键盘鼠标的例子，如果加上了各种洗面奶，衣服啥的反而会起到反作用。其次就是这样综合起来，已经没法再看出到底用户历史行为中的哪个商品与当前商品比较相关，也就是丢失了历史行为中各个商品对当前预测的重要性程度。最后一点就是如果所有用户浏览过的历史行为商品，最后都通过embedding和pooling转换成了固定长度的embedding，这样会限制模型学习用户的多样化兴趣。

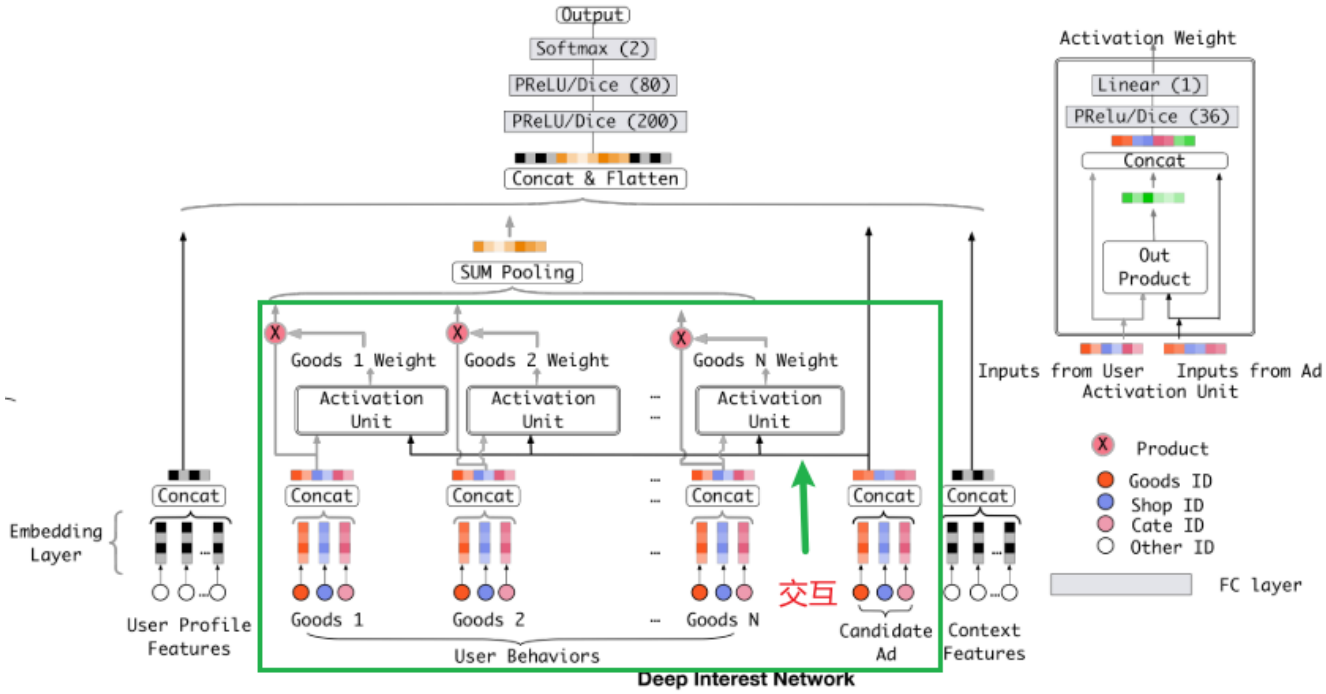
那么改进这个问题的思路有哪些呢？ 第一个就是加大embedding的维度，增加之前各个商品的表达能力，这样即使综合起来，embedding的表达能力也会加强， 能够蕴涵用户的兴趣信息，但是这个在大规模的真实推荐场景计算量超级大，不可取。 另外一个思路就是**在当前候选广告和用户的历史行为之间引入注意力的机制**，这样在预测当前广告是否点击的时候，让模型更关注于与当前广告相关的那些用户历史产品，也就是说**与当前商品更加相关的历史行为更能促进用户的点击行为。** 作者这里又举了之前的一个例子：

> 想象一下，当一个年轻母亲访问电子商务网站时，她发现展示的新手袋很可爱，就点击它。让我们来分析一下点击行为的驱动力。
>
> 展示的广告通过软搜索这位年轻母亲的历史行为，发现她最近曾浏览过类似的商品，如大手提袋和皮包，从而击中了她的相关兴趣

第二个思路就是DIN的改进之处了。DIN通过给定一个候选广告，然后去注意与该广告相关的局部兴趣的表示来模拟此过程。 DIN不会通过使用同一向量来表达所有用户的不同兴趣，而是通过考虑历史行为的相关性来自适应地计算用户兴趣的表示向量（对于给的广告）。 该表示向量随不同广告而变化。下面看一下DIN模型。

DIN模型架构

上面分析完了base模型的不足和改进思路之后，DIN模型的结构就呼之欲出了，首先，它依然是采用了基模型的结构，只不过是在这个的基础上加了一个注意力机制来学习用户兴趣与当前候选广告间的关联程度， 用论文里面的话是，引入了一个新的`local activation unit`， 这个东西用在了用户历史行为特征上面， **能够根据用户历史行为特征和当前广告的相关性给用户历史行为特征embedding进行加权。**我们先看一下它的结构，然后看一下这个加权公式。



这里改进的地方已经框出来了，这里会发现相比于base model， 这里加了一个local activation unit， 这里面是一个前馈神经网络，输入是用户历史行为商品和当前的候选商品， 输出是它俩之间的相关性， 这个相关性相当于每个历史商品的权重，把这个权重与原来的历史行为embedding相乘求和就得到了用户的兴趣表示$\boldsymbol{v}_{U}(A)$, 这个东西的计算公式如下：

$$\boldsymbol{v}_U(A) = f(\boldsymbol{v}_A, \boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_H) = \sum_{j=1}^{H} a(\boldsymbol{e}_j, \boldsymbol{v}_A)\, \boldsymbol{e}_j = \sum_{j=1}^{H} \boldsymbol{w}_j \boldsymbol{e}_j$$

这里的${\boldsymbol{v}*{A}, \boldsymbol{e}*{1}, \boldsymbol{e}*{2}, \ldots, \boldsymbol{e}*{H}}$是用户$U$的历史行为特征embedding， v_{A}表示的是候选广告A的embedding向量， $a(e_j, v_A)=w_j$表示的权重或者历史行为商品与当前广告A的相关性程度。$a(\cdot)$表示的上面那个前馈神经网络，也就是那个所谓的注意力机制， 当然，看图里的话，输入除了历史行为向量和候选广告向量外，还加了一个它俩的外积操作，作者说这里是有利于模型相关性建模的显性知识。

这里有一点需要特别注意，就是这里的权重加和不是1， 准确的说这里不是权重， 而是直接算的相关性的那种分数作为了权重，也就是平时的那种scores(softmax之前的那个值)，这个是为了保留用户的兴趣强度。

DIN实现

下面我们看下DIN的代码复现，这里主要是给大家说一下这个模型的设计逻辑，参考了deepctr的函数API的编程风格， 具体的代码以及示例大家可以去参考后面的GitHub，里面已经给出了详细的注释， 这里主要分析模型的逻辑这块。关于函数API的编程式风格，我们还给出了一份文档， 大家可以先看这个，再看后面的代码部分，会更加舒服些。下面开始：

这里主要和大家说一下DIN模型的总体运行逻辑，这样可以让大家从宏观的层面去把握模型的编写过程。该模型所使用的数据集是movielens数据集， 具体介绍可以参考后面的GitHub。 因为上面反复强调了DIN的应用场景，需要基于用户的历史行为数据， 所以在这个数据集中会有用户过去对电影评分的一系列行为。这在之前的数据集中往往是看不到的。 大家可以导入数据之后自行查看这种行为特征(hist_behavior)。另外还有一点需要说明的是这种历史行为是序列性质的特征， 并且**不同的用户这种历史行为特征长度会不一样**， 但是我们的神经网络是要求序列等长的，所以这种情况我们一般会按照最长的序列进行padding的操作(不够长的填0)， 而到具体层上进行运算的时候，会用

mask掩码的方式标记出这些填充的位置，好保证计算的准确性。 在我们给出的代码中，大家会在AttentionPoolingLayer层的前向传播中看到这种操作。下面开始说编写逻辑：

首先， DIN模型的输入特征大致上分为了三类： Dense(连续型), Sparse(离散型), VarlenSparse(变长离散型)，也就是指的上面的历史行为数据。而不同的类型特征也就决定了后面处理的方式会不同：

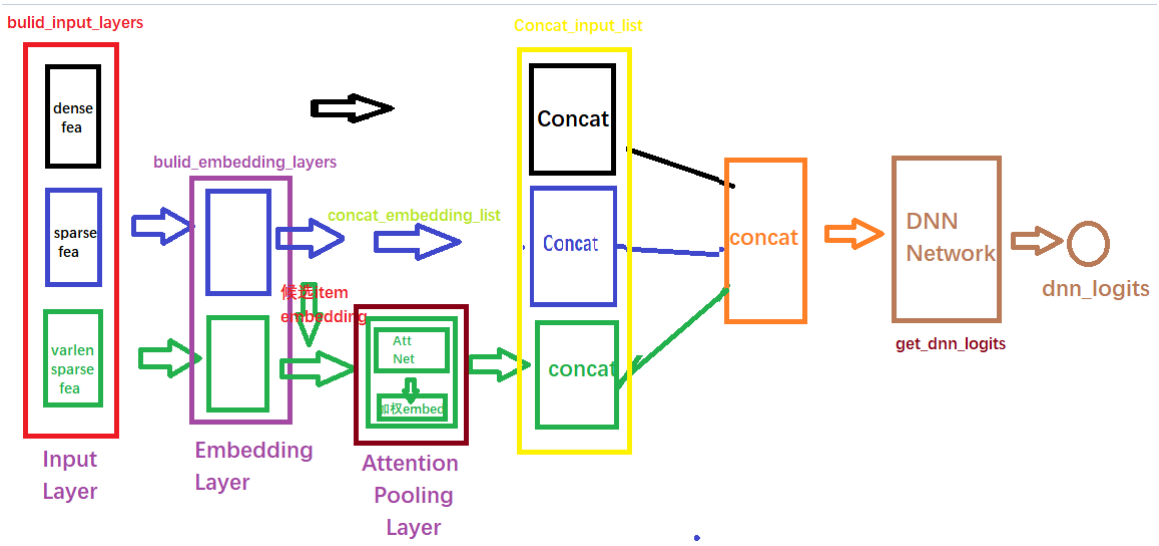
- Dense型特征：由于是数值型了，这里为每个这样的特征建立Input层接收这种输入， 然后拼接起来先放着，等离散的那边处理好之后，和离散的拼接起来进DNN
- Sparse型特征，为离散型特征建立Input层接收输入，然后需要先通过embedding层转成低维稠密向量，然后拼接起来放着，等变长离散那边处理好之后， 一块拼起来进DNN， 但是这里面要注意有个特征的embedding向量还得拿出来用，就是候选商品的embedding向量，这个还得和后面的计算相关性，对历史行为序列加权。
- VarlenSparse型特征：这个一般指的用户的历史行为特征，变长数据， 首先会进行padding操作成等长， 然后建立Input层接收输入，然后通过embedding层得到各自历史行为的embedding向量， 拿着这些向量与上面的候选商品embedding向量进入AttentionPoolingLayer去对这些历史行为特征加权合并，最后得到输出。

通过上面的三种处理， 就得到了处理好的连续特征，离散特征和变长离散特征， 接下来把这三种特征拼接，进DNN网络，得到最后的输出结果即可。所以有了这个解释， 就可以放DIN模型的代码全貌了，大家可以感受下我上面解释的：

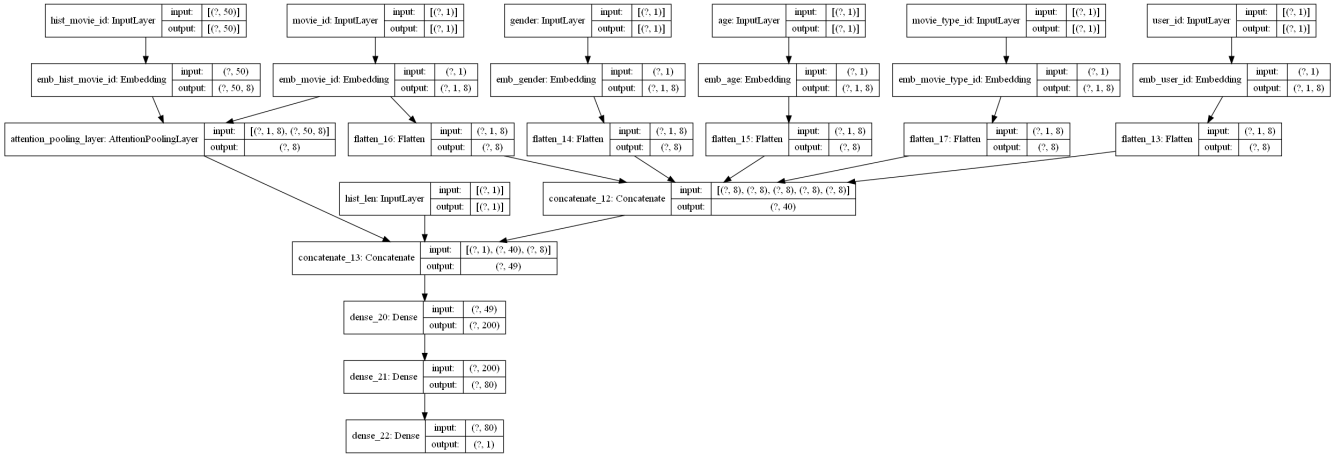
```
# DIN网络搭建
def DIN(feature_columns, behavior_feature_list, behavior_seq_feature_list):
    """
    这里搭建DIN网络，有了上面的各个模块，这里直接拼起来
    :param feature_columns: A list. 里面的每个元素是namedtuple(元组的一种扩展类型，同时支持序号和属性名访问组件)类型，表示的是数据的特征封装版
    :param behavior_feature_list: A list. 用户的候选行为列表
    :param behavior_seq_feature_list: A list. 用户的历史行为列表
    """
    # 构建Input层并将Input层转成列表作为模型的输入
    input_layer_dict = build_input_layers(feature_columns)
    input_layers = list(input_layer_dict.values())

    # 筛选出特征中的sparse和Dense特征， 后面要单独处理
    sparse_feature_columns = list(filter(lambda x: isinstance(x, SparseFeat), feature_columns))
    dense_feature_columns = list(filter(lambda x: isinstance(x, DenseFeat), feature_columns))

    # 获取Dense Input
    dnn_dense_input = []
    for fc in dense_feature_columns:
        dnn_dense_input.append(input_layer_dict[fc.name])

    # 将所有的dense特征拼接
    dnn_dense_input = concat_input_list(dnn_dense_input)    # (None, dense_fea_nums)

    # 构建embedding字典
    embedding_layer_dict = build_embedding_layers(feature_columns, input_layer_dict)

    # 离散的这些特特征embedding之后，然后拼接，然后直接作为全连接层Dense的输入，所以需要进行Flatten
    dnn_sparse_embed_input = concat_embedding_list(sparse_feature_columns, input_layer_dict, embedding_layer_dict, flatten=True)
```

```
    # 将所有的sparse特征embedding特征拼接
    dnn_sparse_input = concat_input_list(dnn_sparse_embed_input)    # (None,
sparse_fea_nums*embed_dim)

    # 获取当前行为特征的embedding， 这里有可能有多个行为产生了行为列表，所以需要列表将其放在一起
    query_embed_list = embedding_lookup(behavior_feature_list, input_layer_dict,
embedding_layer_dict)

    # 获取历史行为的embedding， 这里有可能有多个行为产生了行为列表，所以需要列表将其放在一起
    keys_embed_list = embedding_lookup(behavior_seq_feature_list, input_layer_dict,
embedding_layer_dict)
    # 使用注意力机制将历史行为的序列池化，得到用户的兴趣
    dnn_seq_input_list = []
    for i in range(len(keys_embed_list)):
        seq_embed = AttentionPoolingLayer()([query_embed_list[i], keys_embed_list[i]])  #
(None, embed_dim)
        dnn_seq_input_list.append(seq_embed)

    # 将多个行为序列的embedding进行拼接
    dnn_seq_input = concat_input_list(dnn_seq_input_list)  # (None, hist_len*embed_dim)

    # 将dense特征，sparse特征， 即通过注意力机制加权的序列特征拼接起来
    dnn_input = Concatenate(axis=1)([dnn_dense_input, dnn_sparse_input, dnn_seq_input]) #
(None, dense_fea_num+sparse_fea_nums*embed_dim+hist_len*embed_dim)

    # 获取最终的DNN的预测值
    dnn_logits = get_dnn_logits(dnn_input, activation='prelu')

    model = Model(inputs=input_layers, outputs=dnn_logits)

    return model
```

关于每一块的细节，这里就不解释了，在我们给出的GitHub代码中，我们已经加了非常详细的注释，大家看那个应该很容易看明白， 为了方便大家的阅读，我们这里还给大家画了一个整体的模型架构图，帮助大家更好的了解每一块以及前向传播。（画的图不是很规范，先将就看一下，后面我们会统一在优化一下这个手工图）。



下面是一个通过keras画的模型结构图，为了更好的显示，数值特征和类别特征都只是选择了一小部分，画图的代码也在github中。



思考

DIN模型在工业上的应用还是比较广泛的， 大家可以自由去通过查资料看一下具体实践当中这个模型是怎么用的？有什么问题？比如行为序列的制作是否合理， 如果时间间隔比较长的话应不应该分一下段？ 再比如注意力机制那里能不能改成别的计算注意力的方式会好点？(我们也知道注意力机制的方式可不仅DNN这一种)， 再比如注意力权重那里该不该加softmax？ 这些其实都是可以值的思考探索的一些问题，根据实际的业务场景，大家也可以总结一些更加有意思的工业上应用该模型的技巧和tricks，欢迎一块讨论和分享。

参考资料

- DIN原论文
- deepctr
- AI上推荐 之 AFM与DIN模型（当推荐系统遇上了注意力机制）
- 王喆 - 《深度学习推荐系统》