DeepFM全方面解析（附pytorch源码）

deepFM 机器AI学习 数据AI挖掘 5天前

注下面公众号“机器AI学习 数据AI挖掘”，回复“deepFM”，获取项目完整源代码。

最近看了DeepFM这个模型。把我学习的思路和总结放上来给大家和未来的自己做个参考和借鉴。文章主要希望能串起学习DeepFM的各个环节，梳理整个学习思路。以“我”的角度浅谈一下DeepFM基础知识+看过的一些有用文献+最后附上可实现的pytorch代码，用具体的Kaggle竞赛案例来梳理DeepFM项目的流程。（文章语言尽量通俗易懂，所以背后的逻辑推导尽可能罗列在附带的参考文献里）

DeepFm的学习路线

DeepFM的paper → 网上的解析文章 →源码复现

> 我一开始是看了一遍原文的paper，缺点是很多概念都比较模糊，比如我看DeepFM的时候根据不知道FM是什么，前面的基础没有，看这些衍生概念就很困惑。优点是我能知道文章想体现一些重点以及一些概念名词，比如说我虽然不知道FM是什么，但是我知道DeepFM是将FM做了一个新的拓展，那我知道了FM这个名词之后，我后面补充基础知识的时候就会有针对性。paper很困惑的点，可以给自己留下一些印象和问题，让你后面再去接触、弄懂这些知识的时候就会有一个似曾相识 → 恍然大悟的感觉。随后的内容是在我看过一遍原文的基础上做的补充。

DeepFM基础知识（来由与作用）

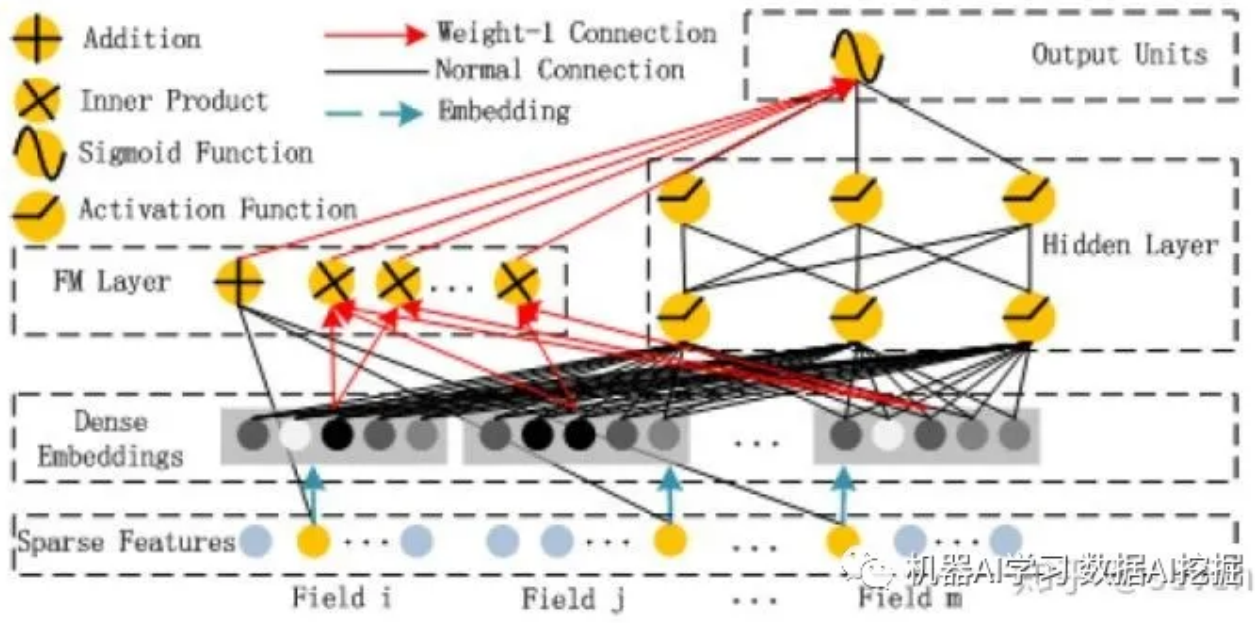
我们大家基本都了解过线性回归方程，也就是

$$\hat{Y}=w_1x_1+w_2x_2+\ldots+w_nx_n$$

但是线性回归有一个不好的地方在于它直接估计**2阶或者高阶特征变量**的时候，难以直接估计出特征前面的权重（也称系数）。那么就衍生出了一个叫做**FM**的算法来解决这个问题。这里背后的逻辑重点看一下这篇文章，把FM诞生的来由及理论讲述得很是到位：

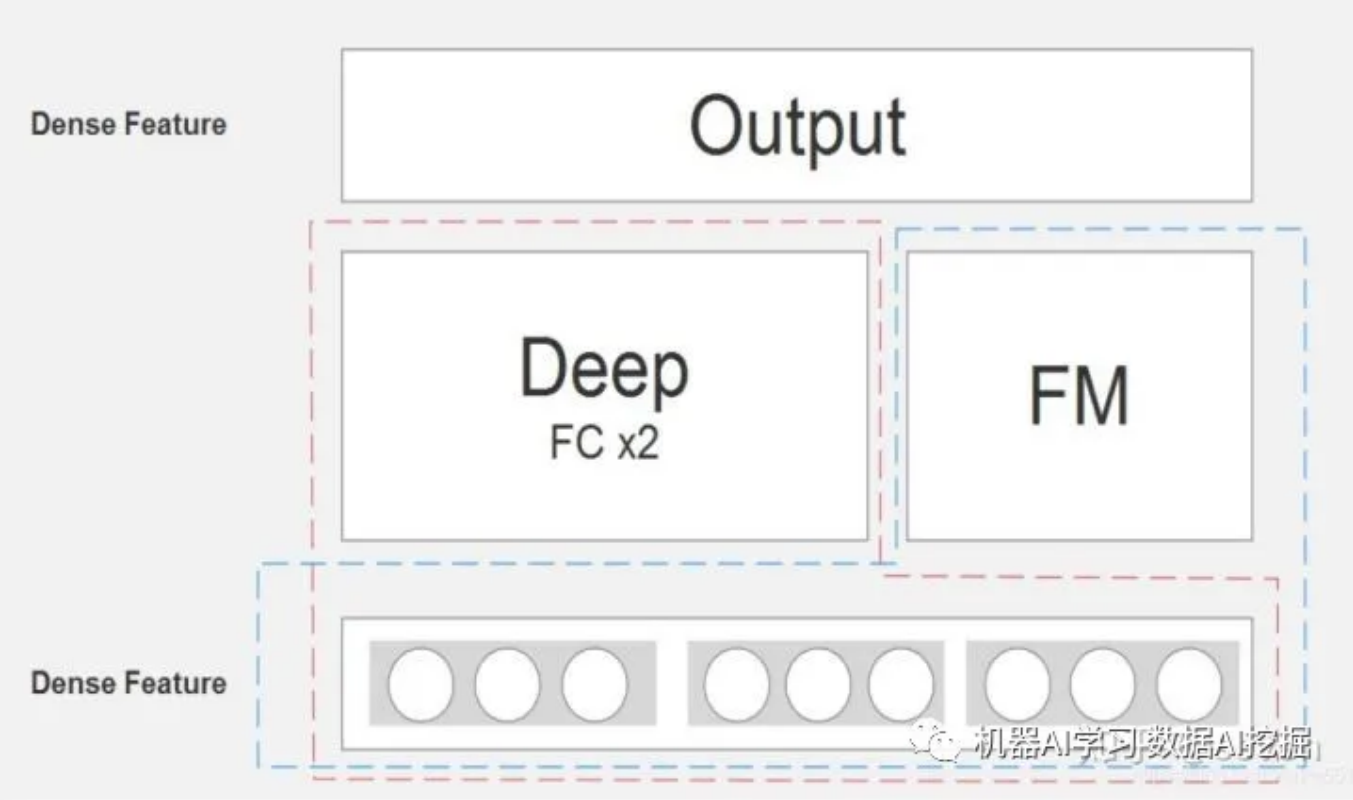
https://blog.csdn.net/itplus/article/details/40534923
blog.csdn.net

DeepFM是深度学习版的FM（DeepLearning+FM=DeepFM嘛），直接上图：



DeepFM的模型框架图

这个图来源于paper原文，看起来比较复杂。那看一下网友的简略图：



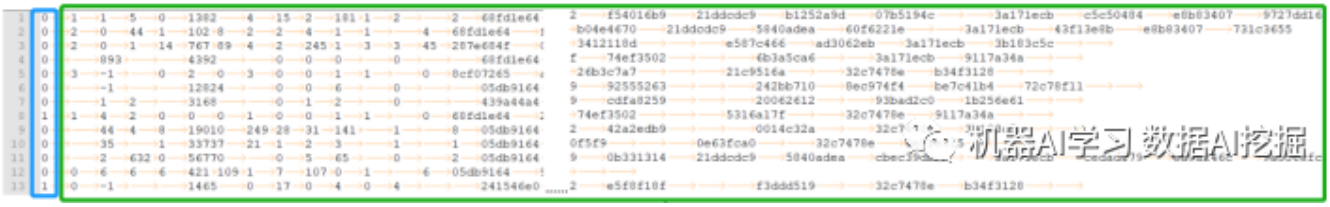
DeepFM模型框架简略图

这样看就清晰多了：FM和Deep两个结构共同在提取特征，最后再把两者提取的特征做一个结合作为整个模型提取的特征。而这些特征都来源于前面的Dense Feature，这个就是模型的亮点之一 —— 权值共享，这些概念比较迷糊。可以看到就是在FM的直接上融入了一些深度学习的知识（主要是帮助映射至数学空间以及高维空间来更好的提取特征）

那说白了，DeepFM也是在处理类似FM，类似线性回归这一类的问题。但它能处理得更好，因为融入了DeepLearning的知识。那既然如此，我们用DeepFM来处理的数据就可以像是很普通的一个矩阵，每一行有多个x（这里是k个，代表k个特征变量，自变量），对应一个y（代表标签，因变量），有n行，代表n行数据：

灵魂画手别介意

也可以是稍微复杂点的数据，这里举例Kaggle的一个竞赛数据集：**Criteo's Kaggle display advertising challenge**。稍后的 **pytorch** 源码也是用的这个竞赛的数据集，数据视图如下：



这里截了前13行数据举例，画蓝框的就是我们所谓的标签y，绿框的就是我们的特征变量x。可以看到大体框架跟我们前面的是一样的，所以道理是类似的。但是详细地看绿框中的变量可以看到有一些是非数值型变量，这里在做数据处理的时候当然需要对非数值化数据进行数值化处理，但其实也不难。概念就是将这些值映射到数字空间的意思。这个工作具体会由DeepFM模型的第一层（**embedding层**或者**dense层**来解决）。这么看下来，我们整体的逻辑结构还是比较清晰的。

Pytorch源码

> 基础知识看下来难免会有似懂非懂的感觉，解决心中的疑惑最直接的方式就是看项目源码。找一个具体的应用场景，看项目整体的处理流程，以及相应的处理步骤。

注下面公众号“机器AI学习 数据AI挖掘”，回复“deepFM”，获取项目源代码。

运行环境

电脑：联想小新Air 13 pro
CPU：i5 ，4G运行内存
显卡：NVIDIA GeForce 940MX，2G显存
系统：windows10 64位系统
软件：Anaconda 5.3.0 python 3.6.6 **Pytorch1.0**

数据来源

数据来源于**kaggle**的一个竞赛：

Display Advertising Challenge | Kaggle
www.kaggle.com

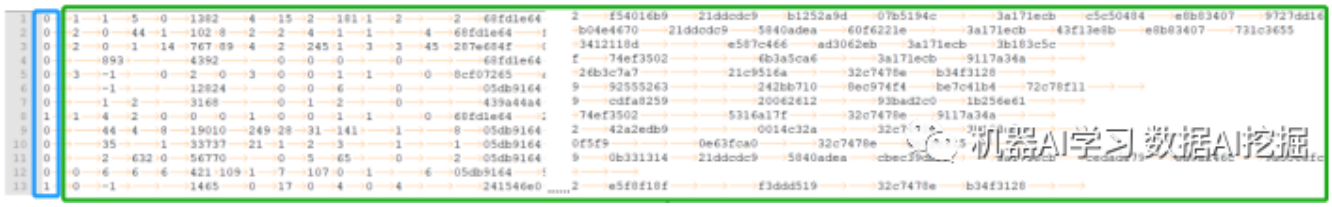
数据集可以从里面下到，注册、登录kaggle账号就可以了。数据压缩包大概**4个G**。这里我只是为了测试模型，就只拿了**前1000行数据**作为**demo dataset**，后面一同放上**github**。

Display Advertising Challenge | Kaggle数据集可以从里面下到，注册、登录kaggle账号就可以了。数据压缩包大概**4个G**。这里我只是为了测试模型，就只拿了**前1000行数据**作为**demo dataset**，后面一同放上**github**。

这个数据竞赛是为了预测**CTR**。

> CTR（Click-Through-Rate）
> 点击通过率，是互联网广告常用的术语，指网络广告（图片广告/文字广告/关键词广告/排名广告/视频广告等）的点击到达率，即该广告的实际点击次数（严格的来说，可以是到达目标页面的数量）除以广告的展现量（Show content）。

数据视图以及数据情况



这个图前面也展示过，**第1列**就是标签（点击与否用0/1表示，1为点击，0为没点击）。后面的**2-40列**数据（**共39列**）是我们的特征变量（指标），也就是原文paper里面提到的“**n-field**”的概念，这里的n是39。然后数据内又分了**前13列**是**连续型变量**，**后26列**是**类别型变量**。

项目源码流程

> 将原始数据（**raw data**）处理成模型的输入数据 —— **datapreprocess.py**
> 将处理好的数据批量输入DeepFM模型训练 —— **main.py**（调用了**DeepFM.pydataset.py**）

项目源码中一些重点提及

datapreprocess.py

主要在做的事情是将我们的数据处理成**embedding层**可输入的形式，这里**embedding层**的基础概念就不赘述了，需要大家自己了解一下。大体思路就是：我们是将39列的每一列做为独立的输入之后再合并，所以根据每一列建立索引字典。建立好之后根据索引字典将**raw data**（上面的数据视图呈现那样）映射到数字空间，即每个值都代表着索引字典里面的索引，可以根据索引找到原来的值。

那我们对连续型特征变量和分类型特征变量作相应的处理（对应着**datapreprocess.py** 中的类**ContinuousFeatureGenerator** 和 **CategoryDictGenerator**）

我们的**raw data**数据是有存在一些缺少值的，我们对缺失值采取的手段是填0处理，对应着两个类下边的**gen**函数：

```
def gen(self, idx, key):
    if key not in self.dicts[idx]:
        res = self.dicts[idx]['<unk>']
    else:
        res = self.dicts[idx][key]
    return res

def gen(self, idx, val):
    if val == '':
        return 0.0
    val = float(val)
    return val
```

在类别型变量处理时，因为把出现频率太低的数据也加进索引字典的话，会导致模型学习的效果下降，所以在建立索引字典的时候我们会将词频太低的数据过滤，词频可以通过 `cutoff` 设置，具体的代码块见：

```
def build(self, datafile, categorial_features, cutoff=0):
    with open(datafile, 'r') as f:
        for line in f:
            features = line.rstrip('\n').split('\t')
            for i in range(0, self.num_feature):
                if features[categorial_features[i]] != '':
                    self.dicts[i][features[categorial_features[i]]] +=
    for i in range(0, self.num_feature):
        self.dicts[i] = filter(lambda x: x[1] >= cutoff,
                               self.dicts[i].items())
        self.dicts[i] = sorted(self.dicts[i], key=lambda x: (-x[1], x[0
        vocabs, _ = list(zip(*self.dicts[i]))
        self.dicts[i] = dict(zip(vocabs, range(1, len(vocabs) + 1)))
        self.dicts[i]['<unk>'] = 0
```

这里过滤的命令就是用的 `filter` ，语法格式就是过滤的条件+对象。

那分类型变量处理整体的思路再强调一下就是词频太低的不加进索引字典中，没有在索引字典出现的类别最后都会填为0，意思就是不只是缺失值会被填为0，一些词频较低的也会填充为0。

将整个 **datapreprocess.py** 运行下来就会在".\data"路径生成处理好的训练数据"**train.txt**"、测试数据"**test.txt**"以及特征表"**feature_size.txt**"。**feature_size**的后26列是根据索引字典得到的，这个地方涉及到**embedding**的原始知识。我们在使用**embedding layer**的时候切记三步走，一是建立索引字典，二是根据索引字典映射原始数据。三是根据索引字典得到**feature_size**之后才建立**embedding layer**。

这份源码就是在映射数据和建立**feature_size**的时候出现了问题，导致后面模型建立的时候，**embedding layer** 的**feature size**太小，对应不上，就会报错：

```
  File "D:\Anaconda3\envs\DL-cpu\lib\site-packages\torch\nn\modules\sparse.py", line 118, in forward
    self.norm_type, self.scale_grad_by_freq, self.sparse)

  File "D:\Anaconda3\envs\DL-cpu\lib\site-packages\torch\nn\functional.py", line 1454, in embedding
    return torch.embedding(weight, input, padding_idx, scale_grad_by_freq, sparse)

RuntimeError: index out of range at c:\a\w\1\s\tmp_conda_3.6_070023\conda\conda-bld\pytorch-cpu_1544079880394\work\aten
\src\th\generic/THTensorEvenMoreMath.cpp:191
```


github上大家也是遇到同样的问题：RuntimeError: index out of range at /pytorch/aten/src/TH/generic/THTensorEvenMoreMath.cpp:191

这里就需要我们做好前面提到的那三个步骤，这是模型建立的前提。

DeepFM.py

DeepFM模型方面整体分为了四大部分：

第一个：一开始的数据处理部分——**embedding layer** 以及**dense layer**（词嵌入层）

这里主要提醒的就是数据格式问题，因为我们的**前13行**数据是**连续型**数据，是**float型**的。这种数据想进入**embedding layer**需要做**离散化处理**，这里我延续了它连续性变量的本质，**前13列**数据使用的是**dense layer**的概念（也就是全连接神经网络层），这个层在keras建立使用的是 `Dense()`，在**pytorch**是 `nn.Linear()`

在源码中具体的地方为：

```
        fm_first_order_Linears = nn.ModuleList(
                [nn.Linear(feature_size, self.embedding_size) for feature_s
        fm_first_order_embeddings = nn.ModuleList(
                [nn.Embedding(feature_size, self.embedding_size) for featur
        self.fm_first_order_models = fm_first_order_Linears.extend(fm_first

        fm_second_order_Linears = nn.ModuleList(
                [nn.Linear(feature_size, self.embedding_size) for feature_s
        fm_second_order_embeddings = nn.ModuleList(
```

```
            [nn.Embedding(feature_size, self.embedding_size) for featur
        self.fm_second_order_models = fm_second_order_Linears.extend(fm_sec
```

所以步骤大的方向是将数据分列（这里是**39列**）输入对应的层，最后再来做**合并**。

第二个：FM 部分（主要提取1阶和2阶特征）

这里想提到的就是2阶特征怎么提取。简略版公式就是：

$$xy = 0.5 \times [(x+y)^2 - (x^2 + y^2)]$$

这里详细推导还是请大家看回上面提到的 这篇资料：

那这里对应到的源码就是：

```
        fm_sum_second_order_emb = sum(fm_second_order_emb_arr)
        fm_sum_second_order_emb_square = fm_sum_second_order_emb * \
            fm_sum_second_order_emb  # (x+y)^2
        fm_second_order_emb_square = [
            item*item for item in fm_second_order_emb_arr]
        fm_second_order_emb_square_sum = sum(
            fm_second_order_emb_square)  # x^2+y^2
        fm_second_order = (fm_sum_second_order_emb_square -
                           fm_second_order_emb_square_sum) * 0.5
```

如果在这个地方存疑，大家可以接着看一下：

deepFM in pytorch
blog.csdn.net

第三个：Deep部分

这里对应着源码的：

```
        all_dims = [self.field_size * self.embedding_size] + \
            self.hidden_dims + [self.num_classes]
        for i in range(1, len(hidden_dims) + 1):
```

```
            setattr(self, 'linear_'+str(i),
                    nn.Linear(all_dims[i-1], all_dims[i]))
            # nn.init.kaiming_normal_(self.fc1.weight)
            setattr(self, 'batchNorm_' + str(i),
                    nn.BatchNorm1d(all_dims[i]))
            setattr(self, 'dropout_'+str(i),
                    nn.Dropout(dropout[i-1]))
```

其实就是两个带着**BatchNorm**和**Dropout**的全连接层， `setattr` 就是让这两个层跟在了前面的模型后边。

deep结构这里的拓展空间还是比较大的，因为这里只是用了两个全连接层 而已。剩下的由大家自由发挥了。

第四个：整合部分

整合部分就是将前面的几大部分特征作为一个整合来作为模型的输出，方便后面跟 **label** 比对进行学习。

对应源码的

```
        bias = torch.nn.Parameter(torch.randn(Xi.size(0)))
        total_sum = torch.sum(fm_first_order, 1) + \
                    torch.sum(fm_second_order, 1) + \
                    torch.sum(deep_out, 1) + bias
```

总结

个人认为这篇文章的学习路线对小白来说还是比较适用的，但是相对的这样的学习时长可能就稍微长一些。但是积累多了，后期的学习策略和方式有相应的调整及改变之后，就会比较快了。

算法和模型方面的感悟就是，其背后真的蕴藏挺多数学原理的。要从0-1推出一套算法是很不容易的。

源码方面的感觉就是得学会调试项目源码，就跟我之前看过的一个方法一样，不断地在源码中插入断点先定位问题，之后再定义问题，然后才是想一下针对性的办法。这个过程比较繁琐，

要不断试错。找到问题关键之后，查资料跟别人讨论，为问题量身定做针对的解决方案。才可以慢慢处理好。