[阿里DIN] 深度兴趣网络源码分析 之 如何建模用户序列

原创 罗西的思考 罗西的思考 10月20日

[阿里DIN] 深度兴趣网络源码分析 之 如何建模用户序列

0x00 摘要

Deep Interest Network（DIN）是阿里妈妈精准定向检索及基础算法团队在2017年6月提出的。其针对电子商务领域（e-commerce industry）的CTR预估，重点在于充分利用/挖掘用户历史行为数据中的信息。

本系列文章将解读论文以及源码，顺便梳理一些深度学习相关概念和TensorFlow的实现。本文是第二篇，将分析如何产生训练数据，建模用户序列。

0x01 DIN 需要什么数据

我们先总述下 DIN 的行为：

- CTR预估一般是将用户的行为序列抽象出一个特征，这里称之为行为emb。
- 之前的预估模型对用户的一组行为序列，都是平等对待，比如同权pooling，或者加时间衰减。
- DIN 则深刻分析了用户行为意图，即用户的每个行为和候选商品的相关性是不同的，以此为契机，利用一个计算相关性的模块（后来也叫attention），对序列行为加权pooling，得到想要的embedding。

可见用户序列是输入核心数据，围绕此数据，又需要用户，商品，商品属性等一系列数据。所以 DIN 需要如下数据：

- 用户字典，用户名对应的id；
- movie字典，item对应的id；
- 种类字典，category对应的id；
- item对应的category信息；
- 训练数据，格式为：label、用户名、目标item、 目标item类别、历史item、历史item对应类别；
- 测试数据，格式同训练数据；

0x02 如何产生数据

prepare_data.sh文件进行了数据处理，生成各种数据，其内容如下。

```
export PATH="~/anaconda4/bin:$PATH"

wget http://snap.stanford.edu/data/amazon/productGraph/categoryFiles/reviews_Books.json.gz
wget http://snap.stanford.edu/data/amazon/productGraph/categoryFiles/meta_Books.json.gz

gunzip reviews_Books.json.gz
gunzip meta_Books.json.gz

python script/process_data.py meta_Books.json reviews_Books_5.json
python script/local_aggretor.py
python script/split_by_user.py
python script/generate_voc.py
```

我们可以看到这些处理文件的作用如下：

- process_data.py : 生成元数据文件，构建负样本，样本分离；
- local_aggretor.py : 生成用户行为序列；

- split_by_user.py : 分割成数据集；
- generate_voc.py : 对用户，电影，种类分别生成三个数据字典；

2.1 基础数据

论文中用的是Amazon Product Data数据，包含两个文件：reviews_Electronics_5.json, meta_Electronics.json。

其中：

- reviews主要是用户买了相关商品产生的上下文信息，包括商品id, 时间，评论等。
- meta文件是关于商品本身的信息，包括商品id, 名称，类别，买了还买等信息。

具体格式如下：

reviews_Electronics数据	
reviewerID	评论者id，例如[A2SUAM1J3GNN3B]
asin	产品的id，例如[0000013714]
reviewerName	评论者昵称
helpful	评论的有用性评级，例如2/3
reviewText	评论文本
overall	产品的评级
summary	评论摘要
unixReviewTime	审核时间（unix时间）
reviewTime	审核时间（原始）

meta_Electronics 数据	
asin	产品的ID
title	产品名称
imUrl	产品图片地址
categories	产品所属的类别列表
description	产品描述

此数据集中的用户行为很丰富，每个用户和商品都有超过5条评论。特征包括goods_id，cate_id，用户评论 goods_id_list 和 cate_id_list。用户的所有行为都是（b1，b2，...，bk，...，bn）。

任务是通过利用前 k 个评论商品来预测第（k + 1）个评论的商品。训练数据集是用每个用户的 k = 1,2，...，n-2　生成的。

2.2 处理数据

2.2.1 生成元数据

通过处理这两个json文件，我们可以生成两个元数据文件：item-info，reviews-info。

```
python script/process_data.py meta_Books.json reviews_Books_5.json
```

具体代码如下，就是简单提取：

```
def process_meta(file):
    fi = open(file, "r")
    fo = open("item-info", "w")
    for line in fi:
        obj = eval(line)
        cat = obj["categories"][0][-1]
        print>>fo, obj["asin"] + "\t" + cat

def process_reviews(file):
    fi = open(file, "r")
    user_map = {}
    fo = open("reviews-info", "w")
    for line in fi:
        obj = eval(line)
        userID = obj["reviewerID"]
        itemID = obj["asin"]
        rating = obj["overall"]
        time = obj["unixReviewTime"]
        print>>fo, userID + "\t" + itemID + "\t" + str(rating) + "\t" + str(time)
```

生成文件如下。

reviews-info格式为：userID，itemID，评分，时间戳

```
A2S166WSCFIFP5 000100039X 5.0 1071100800
A1BM81XB4QHOA3 000100039X 5.0 1390003200
```

```
A1MOSTXNIO5MPJ 000100039X 5.0 1317081600
A2XQ5LZHTD4AFT 000100039X 5.0 1033948800
A3V1MKC2BVWY48 000100039X 5.0 1390780800
A12387207U8U24 000100039X 5.0 1206662400
```

item-info格式为：产品的id，产品所属的类别列表，这里就相当于一个映射表。即 0001048791 这个产品对应 Books这个种类。

```
0001048791 Books
0001048775 Books
0001048236 Books
0000401048 Books
0001019880 Books
0001048813 Books
```

2.2.2 构建样本列表

通过 manual_join 函数构建了负样本，具体逻辑如下：

- 得到所有商品id列表item_list;
- 得到所有用户的行为序列。每一个用户有一个执行序列，每一个序列item的内容是一个 tuple2（userid + item id + rank + timestamp, timestamp);
- 遍历每一个用户
  - 对于该用户的行为序列，按照timestamp排序。
  - 对于排序后的每一个用户行为，构建两个样本：
    - 一个负样本。即把用户行为的item id替换成一个随机选择的item id，click设置为0。
    - 一个正样本，就是用户行为，然后click设置为1。
    - 分别把样本写入文件。

比如：

商品列表是：

```
item_list =
```

```
0000000 = {str} '000100039X'
0000001 = {str} '000100039X'
0000002 = {str} '000100039X'
0000003 = {str} '000100039X'
0000004 = {str} '000100039X'
0000005 = {str} '000100039X'
```

用户的行为序列是：

```
user_map = {dict: 603668}
'A1BM81XB4QHOA3' = {list: 6}
 0 = {tuple: 2} ('A1BM81XB4QHOA3\t000100039X\t5.0\t1390003200', 1390003200.0)
 1 = {tuple: 2} ('A1BM81XB4QHOA3\t0060838582\t5.0\t1190851200', 1190851200.0)
 2 = {tuple: 2} ('A1BM81XB4QHOA3\t0743241924\t4.0\t1143158400', 1143158400.0)
 3 = {tuple: 2} ('A1BM81XB4QHOA3\t0848732391\t2.0\t1300060800', 1300060800.0)
 4 = {tuple: 2} ('A1BM81XB4QHOA3\t0884271781\t5.0\t1403308800', 1403308800.0)
 5 = {tuple: 2} ('A1BM81XB4QHOA3\t1885535104\t5.0\t1390003200', 1390003200.0)
'A1MOSTXNIO5MPJ' = {list: 9}
 0 = {tuple: 2} ('A1MOSTXNIO5MPJ\t000100039X\t5.0\t1317081600', 1317081600.0)
 1 = {tuple: 2} ('A1MOSTXNIO5MPJ\t0143142941\t4.0\t1211760000', 1211760000.0)
 2 = {tuple: 2} ('A1MOSTXNIO5MPJ\t0310325366\t1.0\t1259712000', 1259712000.0)
 3 = {tuple: 2} ('A1MOSTXNIO5MPJ\t0393062112\t5.0\t1179964800', 1179964800.0)
 4 = {tuple: 2} ('A1MOSTXNIO5MPJ\t0872203247\t3.0\t1211760000', 1211760000.0)
 5 = {tuple: 2} ('A1MOSTXNIO5MPJ\t1455504181\t5.0\t1398297600', 1398297600.0)
 6 = {tuple: 2} ('A1MOSTXNIO5MPJ\t1596917024\t5.0\t1369440000', 1369440000.0)
 7 = {tuple: 2} ('A1MOSTXNIO5MPJ\t1600610676\t5.0\t1276128000', 1276128000.0)
 8 = {tuple: 2} ('A1MOSTXNIO5MPJ\t9380340141\t3.0\t1369440000', 1369440000.0)
```

具体代码如下：

```
def manual_join():
    f_rev = open("reviews-info", "r")
    user_map = {}
    item_list = []
    for line in f_rev:
        line = line.strip()
        items = line.split("\t")
        if items[0] not in user_map:
            user_map[items[0]]= []
        user_map[items[0]].append(("\t".join(items), float(items[-1])))
```

```
        item_list.append(items[1])

    f_meta = open("item-info", "r")
    meta_map = {}
    for line in f_meta:
        arr = line.strip().split("\t")
        if arr[0] not in meta_map:
            meta_map[arr[0]] = arr[1]
            arr = line.strip().split("\t")

    fo = open("jointed-new", "w")
    for key in user_map:
        sorted_user_bh = sorted(user_map[key], key=lambda x:x[1]) #把用户行为序列按照时间排序
        for line, t in sorted_user_bh:
            # 对于每一个用户行为
            items = line.split("\t")
            asin = items[1]
            j = 0
            while True:
                asin_neg_index = random.randint(0, len(item_list) - 1) #获取随机item id index
                asin_neg = item_list[asin_neg_index] #获取随机item id
                if asin_neg == asin: #如果恰好是那个item id，则继续选择
                    continue
                items[1] = asin_neg
                # 写入负样本
                print>>fo, "0" + "\t" + "\t".join(items) + "\t" + meta_map[asin_neg]
                j += 1
                if j == 1: #negative sampling frequency
                    break
            # 写入正样本
            if asin in meta_map:
                print>>fo, "1" + "\t" + line + "\t" + meta_map[asin]
            else:
                print>>fo, "1" + "\t" + line + "\t" + "default_cat"
```

最后文件摘录如下，生成了一系列正负样本。

```
0 A10000012B7CGYKOMPQ4L 140004314X 5.0 1355616000 Books
1 A10000012B7CGYKOMPQ4L 000100039X 5.0 1355616000 Books
0 A10000012B7CGYKOMPQ4L 1477817603 5.0 1355616000 Books
```

```
1 A10000012B7CGYKOMPQ4L 0393967972 5.0 1355616000 Books
0 A10000012B7CGYKOMPQ4L 0778329933 5.0 1355616000 Books
1 A10000012B7CGYKOMPQ4L 0446691437 5.0 1355616000 Books
0 A10000012B7CGYKOMPQ4L B006P5CH1O 4.0 1355616000 Collections & Anthologies
```

2.2.3 分离样本

这步骤把样本分离，目的是确定时间线上最后两个样本。

- 读取上一步生成的 jointed-new；
- 用 user_count 计算每个用户的记录数；
- 再次遍历 jointed-new。
  - 如果是该用户记录的最后两行，则在行前面写入 20190119；
  - 如果是该用户记录的前面若干行，则在行前面写入 20180118；
  - 新记录写入到 jointed-new-split-info；

所以，jointed-new-split-info 文件中，前缀为 20190119 的两条记录就是用户行为的最后两条记录，正好是一个正样本，一个负样本，时间上也是最后两个。

代码如下：

```
def split_test():
    fi = open("jointed-new", "r")
    fo = open("jointed-new-split-info", "w")
    user_count = {}
    for line in fi:
        line = line.strip()
        user = line.split("\t")[1]
        if user not in user_count:
            user_count[user] = 0
        user_count[user] += 1
    fi.seek(0)
    i = 0
    last_user = "A26ZDKC53OP6JD"
    for line in fi:
        line = line.strip()
        user = line.split("\t")[1]
        if user == last_user:
            if i < user_count[user] - 2:  # 1 + negative samples
```

```
                print>> fo, "20180118" + "\t" + line
            else:
                print>>fo, "20190119" + "\t" + line
        else:
            last_user = user
            i = 0
            if i < user_count[user] - 2:
                print>> fo, "20180118" + "\t" + line
            else:
                print>>fo, "20190119" + "\t" + line
        i += 1
```

最后文件如下：

```
20180118 0 A10000012B7CGYKOMPQ4L 140004314X 5.0 1355616000 Books
20180118 1 A10000012B7CGYKOMPQ4L 000100039X 5.0 1355616000 Books
20180118 0 A10000012B7CGYKOMPQ4L 1477817603 5.0 1355616000 Books
20180118 1 A10000012B7CGYKOMPQ4L 0393967972 5.0 1355616000 Books
20180118 0 A10000012B7CGYKOMPQ4L 0778329933 5.0 1355616000 Books
20180118 1 A10000012B7CGYKOMPQ4L 0446691437 5.0 1355616000 Books
20180118 0 A10000012B7CGYKOMPQ4L B006P5CH1O 4.0 1355616000 Collections & Anthologies
20180118 1 A10000012B7CGYKOMPQ4L 0486227081 4.0 1355616000 Books
20180118 0 A10000012B7CGYKOMPQ4L B00HWI5OP4 4.0 1355616000 United States
20180118 1 A10000012B7CGYKOMPQ4L 048622709X 4.0 1355616000 Books
20180118 0 A10000012B7CGYKOMPQ4L 1475005873 4.0 1355616000 Books
20180118 1 A10000012B7CGYKOMPQ4L 0486274268 4.0 1355616000 Books
20180118 0 A10000012B7CGYKOMPQ4L 098960571X 4.0 1355616000 Books
20180118 1 A10000012B7CGYKOMPQ4L 0486404730 4.0 1355616000 Books
20190119 0 A10000012B7CGYKOMPQ4L 1495459225 4.0 1355616000 Books
20190119 1 A10000012B7CGYKOMPQ4L 0830604790 4.0 1355616000 Books
```

2.2.4 生成行为序列

local_aggretor.py 用来生成用户行为序列。

例如对于 reviewerID=0 的用户，他的pos_list为[13179, 17993, 28326, 29247, 62275]，生成的训练集格式为（reviewerID, hist, pos_item, 1), (reviewerID, hist, neg_item, 0)。

这里需要注意hist并不包含pos_item或者neg_item，hist只包含在pos_item之前点击过的item，因为DIN采用类似attention的机制，只有历史行为的attention才对后续的有影响，所以hist只包含pos_item之前点击的item才有意义。

具体逻辑是：

- 遍历 "jointed-new-split-info" 的所有行
  - 不停累计 click 状态的 item id 和 cat id。
    - 如果是 20180118 开头，则写入 local_train。
    - 如果是 20190119 开头，则写入 local_test。

因为 20190119 是时间上排最后的两个序列，所以最终 local_test 文件中，得到的是每个用户的两个累积行为序列，即这个行为序列从时间上包括从头到尾所有时间。

这里文件命名比较奇怪，因为实际训练测试使用的是 **local_test** 文件中的数据。

一个正样本，一个负样本。两个序列只有最后一个item id 和 click 与否不同，其余都相同。

具体代码如下：

```
fin = open("jointed-new-split-info", "r")
ftrain = open("local_train", "w")
ftest = open("local_test", "w")

last_user = "0"
common_fea = ""
line_idx = 0
for line in fin:
    items = line.strip().split("\t")
    ds = items[0]
    clk = int(items[1])
    user = items[2]
    movie_id = items[3]
    dt = items[5]
    cat1 = items[6]

    if ds=="20180118":
        fo = ftrain
    else:
        fo = ftest
    if user != last_user:
```

```
            movie_id_list = []
            cate1_list = []
        else:
            history_clk_num = len(movie_id_list)
            cat_str = ""
            mid_str = ""
            for c1 in cate1_list:
                cat_str += c1 + ""
            for mid in movie_id_list:
                mid_str += mid + ""
            if len(cat_str) > 0: cat_str = cat_str[:-1]
            if len(mid_str) > 0: mid_str = mid_str[:-1]
            if history_clk_num >= 1:    # 8 is the average length of user behavior
                print >> fo, items[1] + "\t" + user + "\t" + movie_id + "\t" + cat1 +"\t" + mid_str +
        last_user = user
        if clk: #如果是click状态
            movie_id_list.append(movie_id) # 累积对应的movie id
            cate1_list.append(cat1) # 累积对应的cat id
        line_idx += 1
```

最后local_test数据摘录如下：

```
0 A10000012B7CGYKOMPQ4L 1495459225 Books 000100039X0393967972044669143704862270810486227O9X048627
1 A10000012B7CGYKOMPQ4L 0830604790 Books 000100039X0393967972044669143704862270810486227O9X048627
```

2.2.5 分成训练集和测试集

split_by_user.py 的作用是分割成数据集。

是随机从1~10中选取整数，如果恰好是2，就作为验证数据集。

```
fi = open("local_test", "r")
ftrain = open("local_train_splitByUser", "w")
ftest = open("local_test_splitByUser", "w")

while True:
    rand_int = random.randint(1, 10)
    noclk_line = fi.readline().strip()
```

```
    clk_line = fi.readline().strip()
    if noclk_line == "" or clk_line == "":
        break
    if rand_int == 2:
        print >> ftest, noclk_line
        print >> ftest, clk_line
    else:
        print >> ftrain, noclk_line
        print >> ftrain, clk_line
```

举例如下：

格式为：label, 用户id，候选item id，候选item 种类，行为序列，种类序列

```
0 A3BI7R43VUZ1TY B00JNHU0T2 Literature & Fiction 0989464105B00B01691C14778097321608442845 BooksLi
1 A3BI7R43VUZ1TY 0989464121 Books 0989464105B00B01691C14778097321608442845 BooksLiterature & Fict
```

2.2.6 生成数据字典

generate_voc.py 的作用是对用户，电影，种类分别生成三个数据字典。三个字典分别包括所有用户id，所有电影id，所有种类id。这里就是简单的把三种元素从1 开始排序。

可以理解为 用movie id，categories，reviewerID分别生产三个 map(movie_map, cate_map, uid_map)，key为对应的原始信息，value为按key排序后的index（从0开始顺序排序），然后将原数据的对应列原始数据转换成key对应的index。

```
import cPickle

f_train = open("local_train_splitByUser", "r")
uid_dict = {}
mid_dict = {}
cat_dict = {}

iddd = 0
for line in f_train:
    arr = line.strip("\n").split("\t")
    clk = arr[0]
    uid = arr[1]
    mid = arr[2]
    cat = arr[3]
```

```
    mid_list = arr[4]
    cat_list = arr[5]
    if uid not in uid_dict:
        uid_dict[uid] = 0
    uid_dict[uid] += 1
    if mid not in mid_dict:
        mid_dict[mid] = 0
    mid_dict[mid] += 1
    if cat not in cat_dict:
        cat_dict[cat] = 0
    cat_dict[cat] += 1
    if len(mid_list) == 0:
        continue
    for m in mid_list.split(""):
        if m not in mid_dict:
            mid_dict[m] = 0
        mid_dict[m] += 1
    iddd+=1
    for c in cat_list.split(""):
        if c not in cat_dict:
            cat_dict[c] = 0
        cat_dict[c] += 1

sorted_uid_dict = sorted(uid_dict.iteritems(), key=lambda x:x[1], reverse=True)
sorted_mid_dict = sorted(mid_dict.iteritems(), key=lambda x:x[1], reverse=True)
sorted_cat_dict = sorted(cat_dict.iteritems(), key=lambda x:x[1], reverse=True)

uid_voc = {}
index = 0
for key, value in sorted_uid_dict:
    uid_voc[key] = index
    index += 1

mid_voc = {}
mid_voc["default_mid"] = 0
index = 1
for key, value in sorted_mid_dict:
    mid_voc[key] = index
    index += 1

cat_voc = {}
cat_voc["default_cat"] = 0
```

```
index = 1
for key, value in sorted_cat_dict:
    cat_voc[key] = index
    index += 1

cPickle.dump(uid_voc, open("uid_voc.pkl", "w"))
cPickle.dump(mid_voc, open("mid_voc.pkl", "w"))
cPickle.dump(cat_voc, open("cat_voc.pkl", "w"))
```

最终，得到DIN模型处理的几个文件:

- **uid_voc.pkl**：用户字典，用户名对应的id；
- **mid_voc.pkl**：movie字典,item对应的id；
- **cat_voc.pkl**：种类字典，category对应的id；
- **item-info**：item对应的category信息；
- **reviews-info**：review 元数据，格式为：userID，itemID，评分，时间戳，用于进行负采样的数据；
- **local_train_splitByUser**：训练数据，一行格式为：label、用户名、目标item、 目标item类别、历史item、历史item对应类别；
- **local_test_splitByUser**：测试数据，格式同训练数据；

0x03 如何使用数据

3.1 训练数据

train.py部分，做的事情就是先用初始模型评估一遍测试集，然后按照batch训练，每1000次评估测试集。

精简版代码如下：

```
def train(
        train_file = "local_train_splitByUser",
        test_file = "local_test_splitByUser",
        uid_voc = "uid_voc.pkl",
        mid_voc = "mid_voc.pkl",
        cat_voc = "cat_voc.pkl",
        batch_size = 128,
        maxlen = 100,
```

```
        test_iter = 100,
        save_iter = 100,
        model_type = 'DNN',
    seed = 2,
):
    with tf.Session(config=tf.ConfigProto(gpu_options=gpu_options)) as sess:
        # 获取训练数据和测试数据
        train_data = DataIterator(train_file, uid_voc, mid_voc, cat_voc, batch_size, maxlen, shuf
        test_data = DataIterator(test_file, uid_voc, mid_voc, cat_voc, batch_size, maxlen)
        n_uid, n_mid, n_cat = train_data.get_n()
        # 建立模型
        model = Model_DIN(n_uid, n_mid, n_cat, EMBEDDING_DIM, HIDDEN_SIZE, ATTENTION_SIZE)
        iter = 0
        lr = 0.001
        for itr in range(3):
            loss_sum = 0.0
            accuracy_sum = 0.
            aux_loss_sum = 0.
            for src, tgt in train_data:
                # 准备数据
                uids, mids, cats, mid_his, cat_his, mid_mask, target, sl, noclk_mids, noclk_cats =
                # 训练
                loss, acc, aux_loss = model.train(sess, [uids, mids, cats, mid_his, cat_his, mid_
                loss_sum += loss
                accuracy_sum += acc
                aux_loss_sum += aux_loss
                iter += 1
                if (iter % test_iter) == 0:
    eval(sess, test_data, model, best_model_path)
                    loss_sum = 0.0
                    accuracy_sum = 0.0
                    aux_loss_sum = 0.0
                if (iter % save_iter) == 0:
                    model.save(sess, model_path+"--"+str(iter))
            lr *= 0.5
```

3.2 迭代读入

DataInput 是一个迭代器，作用就是每次调用返回下一个batch的数据。这段代码涉及到数据如何按照batch划分，以及如何构造一个迭代器。

前面提到，训练数据集格式为：label, 用户id，候选item id，候选item 种类，行为序列，种类序列

3.2.1 初始化

基本逻辑是：

`__init__` 函数中：

- 从三个pkl文件中读取，生成三个字典，分别放在 self.source_dicts 里面，对应 [uid_voc, mid_voc, cat_voc]；
- 从 "item-info" 读取，生成映射关系，最后 self.meta_id_map 中的就是每个movie id 对应的 cateory id，即构建 movie id 和 category id 之间的映射关系。关键代码是：self.meta_id_map[mid_idx] = cat_idx ；
- 从 "reviews-info" 读取，生成负采样所需要的id list；
- 得倒各种基础数据，比如用户列表长度，movie列表长度等等；

代码如下：

```
class DataIterator:

    def __init__(self, source,
                 uid_voc,
                 mid_voc,
                 cat_voc,
                 batch_size=128,
                 maxlen=100,
                 skip_empty=False,
                 shuffle_each_epoch=False,
                 sort_by_length=True,
                 max_batch_size=20,
                 minlen=None):
        if shuffle_each_epoch:
            self.source_orig = source
            self.source = shuffle.main(self.source_orig, temporary=True)
        else:
            self.source = fopen(source, 'r')
```

```
self.source_dicts = []
# 从三个pkl文件中读取，生成三个字典，分别放在 self.source_dicts 里面，对应 [uid_voc, mid_voc,
for source_dict in [uid_voc, mid_voc, cat_voc]:
    self.source_dicts.append(load_dict(source_dict))

# 从 "item-info" 读取，生成映射关系，最后 self.meta_id_map 中的就是每个movie id 对应的 cateor
f_meta = open("item-info", "r")
meta_map = {}
for line in f_meta:
    arr = line.strip().split("\t")
    if arr[0] not in meta_map:
        meta_map[arr[0]] = arr[1]
self.meta_id_map ={}
for key in meta_map:
    val = meta_map[key]
    if key in self.source_dicts[1]:
        mid_idx = self.source_dicts[1][key]
    else:
        mid_idx = 0
    if val in self.source_dicts[2]:
        cat_idx = self.source_dicts[2][val]
    else:
        cat_idx = 0
    self.meta_id_map[mid_idx] = cat_idx

# 从 "reviews-info" 读取，生成负采样所需要的id list；
f_review = open("reviews-info", "r")
self.mid_list_for_random = []
for line in f_review:
    arr = line.strip().split("\t")
    tmp_idx = 0
    if arr[1] in self.source_dicts[1]:
        tmp_idx = self.source_dicts[1][arr[1]]
    self.mid_list_for_random.append(tmp_idx)

# 得倒各种基础数据，比如用户列表长度，movie列表长度等等；
self.batch_size = batch_size
self.maxlen = maxlen
self.minlen = minlen
self.skip_empty = skip_empty

self.n_uid = len(self.source_dicts[0])
self.n_mid = len(self.source_dicts[1])
self.n_cat = len(self.source_dicts[2])
```

```
        self.shuffle = shuffle_each_epoch
        self.sort_by_length = sort_by_length

        self.source_buffer = []
        self.k = batch_size * max_batch_size

        self.end_of_data = False
```

最后数据如下：

```
self = {DataIterator} <data_iterator.DataIterator object at 0x000001F56CB44BA8>
 batch_size = {int} 128
 k = {int} 2560
 maxlen = {int} 100
 meta_id_map = {dict: 367983} {0: 1572, 115840: 1, 282448: 1, 198250: 1, 4275: 1, 260890: 1, 2605
 mid_list_for_random = {list: 8898041} [4275, 4275, 4275, 4275, 4275, 4275, 4275, 4275...
 minlen = {NoneType} None
 n_cat = {int} 1601
 n_mid = {int} 367983
 n_uid = {int} 543060
 shuffle = {bool} False
 skip_empty = {bool} False
 sort_by_length = {bool} True
 source = {TextIOWrapper} <_io.TextIOWrapper name='local_train_splitByUser' mode='r' encoding='cp
 source_buffer = {list: 0} []
 source_dicts = {list: 3}
  0 = {dict: 543060} {'ASEARD9XL1EWO': 449136, 'AZPJ9LUT0FEPY': 0, 'A2NRV79GKAU726': 16, 'A2GEQVD
  1 = {dict: 367983} {'1594483752': 47396, '0738700797': 159716, '1439110239': 193476...
  2 = {dict: 1601} {'Residential': 1281, 'Poetry': 250, 'Winter Sports': 1390...
```

3.2.2 迭代读取

当迭代读取时候，逻辑如下：

- 如果 `self.source_buffer` 没有数据，则读取总数为 k 的文件行数。可以理解为一次性读取最大buffer；
- 如果设定，则按照用户历史行为长度排序；

- 内部迭代开始，从 `self.source_buffer` 取出一条数据：
  - 取出用户这次历史行为 movie id list 到 mid_list；
  - 取出用户这次历史行为 cat id list 到 cat_list；
  - 针对mid_list中的每一个pos_mid，制造5个负采样历史行为数据；具体就是从 mid_list_for_random 中随机获取5个id（如果与pos_mid相同则再次获取新的）；即对于每一个用户的历史行为，代码中选取了5个样本作为负样本；
  - 把 [uid, mid, cat, mid_list, cat_list, noclk_mid_list, noclk_cat_list] 放入souce之中，作为训练数据；
  - 把 [float(ss[0]), 1-float(ss[0])] 放入target之中，作为label；
  - 如果达到了batch_size，则跳出内部迭代，返回本batch数据，即一个最大长度为128的列表；

具体代码见下文：

```
def __next__(self):
        if self.end_of_data:
            self.end_of_data = False
            self.reset()
            raise StopIteration

        source = []
        target = []

        # 如果 self.source_buffer没有数据，则读取总数为 k 的文件行数。可以理解为一次性读取最大buffer
        if len(self.source_buffer) == 0:
            #for k_ in xrange(self.k):
            for k_ in range(self.k):
                ss = self.source.readline()
                if ss == "":
                    break
                self.source_buffer.append(ss.strip("\n").split("\t"))

            # sort by  history behavior length
            # 如果设定，则按照用户历史行为长度排序；
            if self.sort_by_length:
                his_length = numpy.array([len(s[4].split("")) for s in self.source_buffer])
                tidx = his_length.argsort()

                _sbuf = [self.source_buffer[i] for i in tidx]
                self.source_buffer = _sbuf
            else:
```

```
            self.source_buffer.reverse()

        if len(self.source_buffer) == 0:
            self.end_of_data = False
            self.reset()
            raise StopIteration

        try:

            # actual work here，内部迭代开始
            while True:

                # read from source file and map to word index
                try:
                    ss = self.source_buffer.pop()
                except IndexError:
                    break

                uid = self.source_dicts[0][ss[1]] if ss[1] in self.source_dicts[0] else 0
                mid = self.source_dicts[1][ss[2]] if ss[2] in self.source_dicts[1] else 0
                cat = self.source_dicts[2][ss[3]] if ss[3] in self.source_dicts[2] else 0

                # 取出用户一个历史行为 movie id 列表 到 mid_list；
                tmp = []
                for fea in ss[4].split(""):
                    m = self.source_dicts[1][fea] if fea in self.source_dicts[1] else 0
                    tmp.append(m)
                mid_list = tmp

                # 取出用户一个历史行为 cat id 列表 到 cat_list；
                tmp1 = []
                for fea in ss[5].split(""):
                    c = self.source_dicts[2][fea] if fea in self.source_dicts[2] else 0
                    tmp1.append(c)
                cat_list = tmp1

                # read from source file and map to word index

                #if len(mid_list) > self.maxlen:
                #    continue
                if self.minlen != None:
                    if len(mid_list) <= self.minlen:
                        continue
                if self.skip_empty and (not mid_list):
                    continue
```

```
                # 针对mid_list中的每一个pos_mid，制造5个负采样历史行为数据；具体就是从 mid_list_for_ra
                noclk_mid_list = []
                noclk_cat_list = []
                for pos_mid in mid_list:
                    noclk_tmp_mid = []
                    noclk_tmp_cat = []
                    noclk_index = 0
                    while True:
                        noclk_mid_indx = random.randint(0, len(self.mid_list_for_random)-1)
                        noclk_mid = self.mid_list_for_random[noclk_mid_indx]
                        if noclk_mid == pos_mid:
                            continue
                        noclk_tmp_mid.append(noclk_mid)
                        noclk_tmp_cat.append(self.meta_id_map[noclk_mid])
                        noclk_index += 1
                        if noclk_index >= 5:
                            break
                    noclk_mid_list.append(noclk_tmp_mid)
                    noclk_cat_list.append(noclk_tmp_cat)
                source.append([uid, mid, cat, mid_list, cat_list, noclk_mid_list, noclk_cat_list]
                target.append([float(ss[0]), 1-float(ss[0])])

                if len(source) >= self.batch_size or len(target) >= self.batch_size:
                    break
            except IOError:
                self.end_of_data = True

        # all sentence pairs in maxibatch filtered out because of length
        if len(source) == 0 or len(target) == 0:
            source, target = self.next()

        return source, target
```

3.2.3 处理数据

在获取迭代数据之后，还需要进一步处理。

```
uids, mids, cats, mid_his, cat_his, mid_mask, target, sl, noclk_mids, noclk_cats = prepare_data(s
```

可以理解为把这个batch的数据（假设是128条）分类整合起来。比如把这128条的uids, mids, cats，历史序列 分别聚合起来，最后统一发给模型进行训练。

这里重要的一点是生成了mask，其意义是：

mask 表示掩码，它对某些值进行掩盖，使其在参数更新时不产生效果。padding mask 是掩码的一种，

- 什么是 padding mask 呢？因为每个批次输入序列长度是不一样的，也就是说，我们要对输入序列进行对齐。具体来说，就是给在较短的序列后面填充 0。但是如果输入的序列太长，则是截取左边的内容，把多余的直接舍弃。因为这些填充的位置，其实是没什么意义的，所以attention机制不应该把注意力放在这些位置上，需要进行一些处理。
- 具体的做法是，把这些位置的值加上一个非常大的负数(负无穷)，这样的话，经过 softmax，这些位置的概率就会接近0！而我们的 padding mask 实际上是一个张量，每个值都是一个Boolean，值为 false 的地方就是我们要进行处理的地方。

DIN这里，由于一个 Batch 中的用户行为序列不一定都相同，其真实长度保存在 keys_length 中，所以这里要产生 masks 来选择真正的历史行为。

- 首先把mask都设置为0；
- 然后如果该条数据有意义，则把mask设置为1；

具体代码如下：

```
def prepare_data(input, target, maxlen = None, return_neg = False):
    # x: a list of sentences
    #s[4]是mid_list, input的每个item中，mid_list长度不同
    lengths_x = [len(s[4]) for s in input]
    seqs_mid = [inp[3] for inp in input]
    seqs_cat = [inp[4] for inp in input]
    noclk_seqs_mid = [inp[5] for inp in input]
    noclk_seqs_cat = [inp[6] for inp in input]

    if maxlen is not None:
        new_seqs_mid = []
        new_seqs_cat = []
        new_noclk_seqs_mid = []
        new_noclk_seqs_cat = []
        new_lengths_x = []
        for l_x, inp in zip(lengths_x, input):
            if l_x > maxlen:
                new_seqs_mid.append(inp[3][l_x - maxlen:])
                new_seqs_cat.append(inp[4][l_x - maxlen:])
```

```
                new_noclk_seqs_mid.append(inp[5][l_x - maxlen:])
                new_noclk_seqs_cat.append(inp[6][l_x - maxlen:])
                new_lengths_x.append(maxlen)
            else:
                new_seqs_mid.append(inp[3])
                new_seqs_cat.append(inp[4])
                new_noclk_seqs_mid.append(inp[5])
                new_noclk_seqs_cat.append(inp[6])
                new_lengths_x.append(l_x)
        lengths_x = new_lengths_x
        seqs_mid = new_seqs_mid
        seqs_cat = new_seqs_cat
        noclk_seqs_mid = new_noclk_seqs_mid
        noclk_seqs_cat = new_noclk_seqs_cat

        if len(lengths_x) < 1:
            return None, None, None, None

    # lengths_x 保存用户历史行为序列的真实长度，maxlen_x 表示序列中的最大长度;
    n_samples = len(seqs_mid)
    maxlen_x = numpy.max(lengths_x) #选取mid_list长度中最大的，本例中是583
    neg_samples = len(noclk_seqs_mid[0][0])

    # 由于用户历史序列的长度是不固定的, 因此引入 mid_his 等矩阵, 将序列长度固定为 maxlen_x. 对于长度不足
    mid_his = numpy.zeros((n_samples, maxlen_x)).astype('int64') #tuple<128, 583>
    cat_his = numpy.zeros((n_samples, maxlen_x)).astype('int64')
    noclk_mid_his = numpy.zeros((n_samples, maxlen_x, neg_samples)).astype('int64') #tuple<128, 5
    noclk_cat_his = numpy.zeros((n_samples, maxlen_x, neg_samples)).astype('int64') #tuple<128, 5
    mid_mask = numpy.zeros((n_samples, maxlen_x)).astype('float32')
    # zip函数用于将可迭代的对象作为参数，将对象中对应的元素打包成一个个元组，然后返回由这些元组组成的列表
    for idx, [s_x, s_y, no_sx, no_sy] in enumerate(zip(seqs_mid, seqs_cat, noclk_seqs_mid, noclk_
        mid_mask[idx, :lengths_x[idx]] = 1.
        mid_his[idx, :lengths_x[idx]] = s_x
        cat_his[idx, :lengths_x[idx]] = s_y
        # noclk_mid_his 和 noclk_cat_his 都是 (128, 583, 5)
        noclk_mid_his[idx, :lengths_x[idx], :] = no_sx # 就是直接赋值
        noclk_cat_his[idx, :lengths_x[idx], :] = no_sy # 就是直接赋值

    uids = numpy.array([inp[0] for inp in input])
    mids = numpy.array([inp[1] for inp in input])
    cats = numpy.array([inp[2] for inp in input])

    # 把input（128长的list)中的每个UID，mid, cat ... 都提出来，聚合，返回
    if return_neg:
```

```
            return uids, mids, cats, mid_his, cat_his, mid_mask, numpy.array(target), numpy.array(len
        else:
            return uids, mids, cats, mid_his, cat_his, mid_mask, numpy.array(target), numpy.array(len
```

3.2.4 送入模型

最后，送入模型训练，也就是train.py中的这一步：

```
loss, acc, aux_loss = model.train(sess, [uids, mids, cats, mid_his, cat_his, mid_mask, target, sl
```

0xFF 参考

Deep Interest Network解读

深度兴趣网络(DIN,Deep Interest Network)

DIN论文官方实现解析

阿里DIN源码之如何建模用户序列（1）：base方案

阅读原文