知乎 首发于
推荐系统、CTR预估模型的理解与复现

广告点击率预估模型---DIN的Tensorflow2.0代码分析


潜心
公众号：推荐算法的小齿轮；很菜的在读研二

关注他

前言

最近看了2018年阿里在KDD上发表的论文《Deep Interest Network for Click-Through Rate Prediction》，想复现下，看了文章给出的github开源代码，发现环境是TF1.4的，并且注释太少，有些没大理解【还是太菜了】，因此准备参考原有代码使用TF2.0来对模型进行简单的复现。如果有些地方有些出入或者错误，请大佬们给我指出，感谢【因为现在没服务器，所以没像开源中跑完50个epoch】

数据分析【采取开源代码】

1、数据集为论文中的**Amazon Dataset**，下载并解压：

```
wget -c http://snap.stanford.edu/data/amazon/productGraph/categoryFiles/reviews_Electr
gzip -d reviews_Electronics_5.json.gz
wget -c http://snap.stanford.edu/data/amazon/productGraph/categoryFiles/meta_Electroni
gzip -d meta_Electronics.json.gz
```

其中 `reviews_Electronics_5.json` 为用户的行为数据， `meta_Electronics` 为广告的元数据。

`reviews` 某单个样本如下：

```
{
  "reviewerID": "A2SUAM1J3GNN3B",
  "asin": "0000013714",
  "reviewerName": "J. McDonald",
  "helpful": [2, 3],
  "reviewText": "I bought this for my husband who plays the piano.  He is having a won
  "overall": 5.0,
  "summary": "Heavenly Highway Hymns",
  "unixReviewTime": 1252800000,
  "reviewTime": "09 13, 2009"
```

- reviewerID ： 用户ID;
- asin ： 物品ID;
- reviewerName ： 用户姓名;
- helpful ： 评论帮助程度，例如上述为 2/3 ;
- reviewText ： 文本信息;
- overall ： 物品评分;
- summary ： 评论总结
- unixReviewTime ： 时间戳
- reviewTime ： 时间

meta 某样本如下:


```
{
  "asin": "0000031852",
  "title": "Girls Ballet Tutu Zebra Hot Pink",
  "price": 3.17,
  "imUrl": "http://ecx.images-amazon.com/images/I/51fAmVkTbyL._SY300_.jpg",
  "related":
  {
    "also_bought": ["B00JHONN1S", "B002BZX8Z6", "B00D2K1M3O", "0000031909", "B00613WDT
    "also_viewed": ["B002BZX8Z6", "B00JHONN1S", "B008F0SU0Y", "B00D23MC6W", "B00AFDOPD
    "bought_together": ["B002BZX8Z6"]
  },
  "salesRank": {"Toys & Games": 211836},
  "brand": "Coxlures",
  "categories": [["Sports & Outdoors", "Other Sports", "Dance"]]
}
```


各字段分别为:

- asin ： 物品ID;
- title ： 物品名称;
- price ： 物品价格;
- imUrl ： 物品图片的URL;
- related ： 相关产品(也买，也看，一起买，看后再买);
- salesRank ： 销售排名信息;

```
def to_df(file_path):
    """
    转化为DataFrame结构
    :param file_path: 文件路径
    :return:
    """
    with open(file_path, 'r') as fin:
        df = {}
        i = 0
        for line in fin:
            df[i] = eval(line)
            i += 1
        df = pd.DataFrame.from_dict(df, orient='index')
        return df

reviews_df = to_df('../raw_data/reviews_Electronics_5.json')

# 可以直接调用pandas的read_json方法，但会改变列的顺序
# reviews2_df = pd.read_json('../raw_data/reviews_Electronics_5.json', lines=True)

# 序列化保存
with open('../raw_data/reviews.pkl', 'wb') as f:
    pickle.dump(reviews_df, f, pickle.HIGHEST_PROTOCOL)

meta_df = to_df('../raw_data/meta_Electronics.json')
# 只保留review_df出现过的广告
meta_df = meta_df[meta_df['asin'].isin(reviews_df['asin'].unique())]
meta_df = meta_df.reset_index(drop=True)

with open('../raw_data/meta.pkl', 'wb') as f:
    pickle.dump(meta_df, f, pickle.HIGHEST_PROTOCOL)
```

3、对 reaviews 和 meta 数据进行处理：

- reviews选取 'reviewerID', 'asin', 'unixReviewTime' 列，并将用户ID、物品ID【通过meta】映射为数值；
- meta选取 'asin', 'categories' 列，物品种类只选取列表最后一个，并将物品ID、种类ID进行映射；
- 统计用户人数 user_count，物品总数 item_count，总样本数 sample_count；

```
    制作一个映射，键为列名，值为序列数字
    :param df: reviews_df / meta_df
    :param col_name: 列名
    :return: 字典，键
    """
    key = sorted(df[col_name].unique().tolist())
    m = dict(zip(key, range(len(key))))
    df[col_name] = df[col_name].map(lambda x: m[x])
    return m, key


# reviews
reviews_df = pd.read_pickle('../raw_data/reviews.pkl')
reviews_df = reviews_df[['reviewerID', 'asin', 'unixReviewTime']]

# meta
meta_df = pd.read_pickle('../raw_data/meta.pkl')
meta_df = meta_df[['asin', 'categories']]
# 类别只保留最后一个
meta_df['categories'] = meta_df['categories'].map(lambda x: x[-1][-1])

# meta_df文件的物品ID映射
asin_map, asin_key = build_map(meta_df, 'asin')
# meta_df文件物品种类映射
cate_map, cate_key = build_map(meta_df, 'categories')
# reviews_df文件的用户ID映射
revi_map, revi_key = build_map(reviews_df, 'reviewerID')

# user_count: 192403 item_count: 63001 cate_count: 801 example_count: 1689188
user_count, item_count, cate_count, example_count = \
    len(revi_map), len(asin_map), len(cate_map), reviews_df.shape[0]
# print('user_count: %d\titem_count: %d\tcate_count: %d\texample_count: %d' %
#       (user_count, item_count, cate_count, example_count))

# 按物品id排序，并重置索引
meta_df = meta_df.sort_values('asin')
meta_df = meta_df.reset_index(drop=True)

# reviews_df文件物品id进行映射，并按照用户id、浏览时间进行排序，重置索引
reviews_df['asin'] = reviews_df['asin'].map(lambda x: asin_map[x])
reviews_df = reviews_df.sort_values(['reviewerID', 'unixReviewTime'])
```

```
cate_list = np.array(meta_df['categories'], dtype='int32')

# 保存所需数据为pkl文件
with open('../raw_data/remap.pkl', 'wb') as f:
    pickle.dump(reviews_df, f, pickle.HIGHEST_PROTOCOL)
    pickle.dump(cate_list, f, pickle.HIGHEST_PROTOCOL)
    pickle.dump((user_count, item_count, cate_count, example_count),
                f, pickle.HIGHEST_PROTOCOL)
    pickle.dump((asin_key, cate_key, revi_key), f, pickle.HIGHEST_PROTOCOL)
```

4、构建数据集，这里有所不同，我还求出了整个数据集的最大序列长度，为了后面构建用户浏览历史记录矩阵。

```
with open('raw_data/remap.pkl', 'rb') as f:
    reviews_df = pickle.load(f)
    cate_list = pickle.load(f)
    user_count, item_count, cate_count, example_count = pickle.load(f)

train_set, test_set = [], []

# 最大的序列长度
max_sl = 0

"""
生成训练集、测试集，每个用户所有浏览的物品（共n个）前n-1个为训练集（正样本），并生成相应的负样本，共有n-2个训练集（第1个无浏览历史），第n个作为测试集。
"""
for reviewerID, hist in reviews_df.groupby('reviewerID'):
    # 每个用户浏览过的物品，即为正样本
    pos_list = hist['asin'].tolist()
    max_sl = max(max_sl, len(pos_list))

    # 生成负样本
    def gen_neg():
        neg = pos_list[0]
        while neg in pos_list:
            neg = random.randint(0, item_count - 1)
        return neg
```

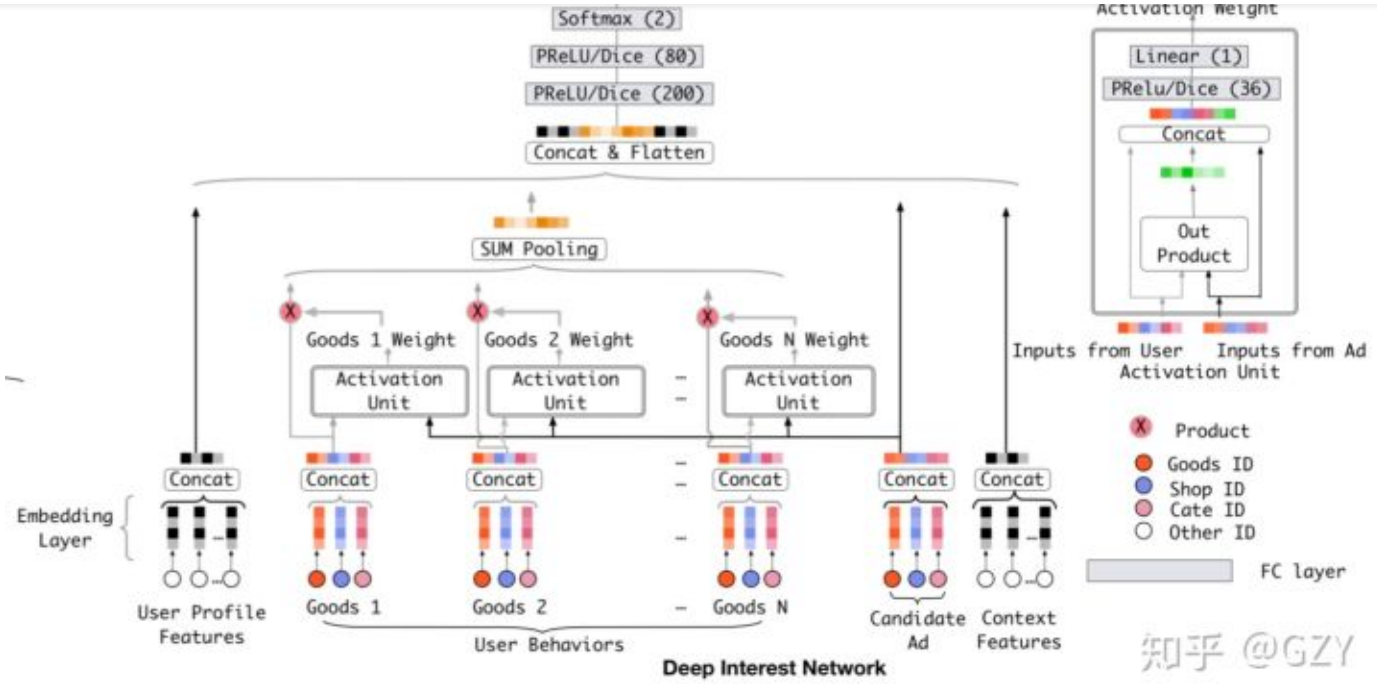
```
            # 生成每一次的历史记录，即之前的浏览历史
            hist = pos_list[:i]
            sl = len(hist)
            if i != len(pos_list) - 1:
                # 保存正负样本，格式：用户ID，正/负物品id，浏览历史，浏览历史长度，标签（1/0）
                train_set.append((reviewerID, pos_list[i], hist, sl, 1))
                train_set.append((reviewerID, neg_list[i], hist, sl, 0))
            else:
                # 最后一次保存为测试集
                test_set.append((reviewerID, pos_list[i], hist, sl, 1))
                test_set.append((reviewerID, neg_list[i], hist, sl, 0))

    # 打乱顺序
    random.shuffle(train_set)
    random.shuffle(test_set)

    assert len(test_set) == user_count

    # 写入dataset.pkl文件
    with open('dataset/dataset.pkl', 'wb') as f:
        pickle.dump(train_set, f, pickle.HIGHEST_PROTOCOL)
        pickle.dump(test_set, f, pickle.HIGHEST_PROTOCOL)
        pickle.dump(cate_list, f, pickle.HIGHEST_PROTOCOL)
        pickle.dump((user_count, item_count, cate_count, max_sl), f, pickle.HIGHEST_PROTOC
```

模型构建



1、定义模型所需的各种层

```
class DIN(tf.keras.Model):
    def __init__(self, user_num, item_num, cate_num, cate_list, hidden_units):
        """
        :param user_num: 用户数量
        :param item_num: 物品数量
        :param cate_num: 物品种类数量
        :param cate_list: 物品种类列表
        :param hidden_units: 隐藏层单元
        """
        super(DIN, self).__init__()
        self.cate_list = tf.convert_to_tensor(cate_list, dtype=tf.int32)
        self.hidden_units = hidden_units
        # self.user_embed = tf.keras.layers.Embedding(
        #     input_dim=user_num, output_dim=hidden_units, embeddings_initializer='ran
        #     embeddings_regularizer=tf.keras.regularizers.l2(0.01), name='user_embed'
        self.item_embed = tf.keras.layers.Embedding(
            input_dim=item_num, output_dim=self.hidden_units, embeddings_initializer='
            embeddings_regularizer=tf.keras.regularizers.l2(0.01), name='item_embed')
        self.cate_embed = tf.keras.layers.Embedding(
            input_dim=cate_num, output_dim=self.hidden_units, embeddings_initializer='
            embeddings_regularizer=tf.keras.regularizers.l2(0.01), name='cate_embed'
        )
```

```
        self.att_dense2 = tf.keras.layers.Dense(40, activation='sigmoid')
        self.att_dense3 = tf.keras.layers.Dense(1)
        self.bn2 = tf.keras.layers.BatchNormalization()
        self.concat2 = tf.keras.layers.Concatenate(axis=-1)
        self.dense1 = tf.keras.layers.Dense(80, activation='sigmoid')
        self.activation1 = tf.keras.layers.PReLU()
        # self.activation1 = Dice()
        self.dense2 = tf.keras.layers.Dense(40, activation='sigmoid')
        self.activation2 = tf.keras.layers.PReLU()
        # self.activation2 = Dice()
        self.dense3 = tf.keras.layers.Dense(1, activation=None)
```

2、根据模型图，首先是对 User Behaviors 、 Candidate Ad 的embedding进行构建。在该数据集中，需要联合Goods ID和Cate ID。【因为User的gender、age信息不存在，并不需要进行User自身属性的embedding】

注：这里我并没有像【开源代码】在分为item_i（正样本）和item_j（负样本）然后联合进行求出损失。

```
    def call(self, inputs):
      # user为用户ID，item为物品id，hist为之前的历史记录，即物品id列表，sl为最大列表长度
        user, item, hist, sl = inputs[0], tf.squeeze(inputs[1], axis=1), inputs[2], tf
        # user_embed = self.u_embed(user)
        item_embed = self.concat_embed(item)
        hist_embed = self.concat_embed(hist)
        ......

    def concat_embed(self, item):
        """
        拼接物品embedding和物品种类embedding
        :param item: 物品id
        :return: 拼接后的embedding
        """
        # cate = tf.transpose(tf.gather_nd(self.cate_list, [item]))
        cate = tf.gather(self.cate_list, item)
        cate = tf.squeeze(cate, axis=1) if cate.shape[-1] == 1 else cate
        item_embed = self.item_embed(item)
        item_cate_embed = self.cate_embed(cate)
        embed = self.concat([item_embed, item_cate_embed])
```

```
  def call(self, inputs):
  ......
    # 经过attention的物品embedding
    hist_att_embed = self.attention(item_embed, hist_embed, sl)
    hist_att_embed = self.bn1(hist_att_embed)
    hist_att_embed = tf.reshape(hist_att_embed, [-1, self.hidden_units * 2])
    u_embed = self.dense(hist_att_embed)
    ......

def attention(self, queries, keys, keys_length):
    """
    activation unit
    :param queries: 候选广告（物品）embedding
    :param keys: 用户行为（历史记录）embedding
    :param keys_length: 用户行为embedding中的有效长度
    :return:
    """
    # 候选物品的隐藏向量维度，hidden_unit * 2
    queries_hidden_units = queries.shape[-1]
    # 每个历史记录的物品embed都需要与候选物品的embed拼接，故候选物品embed重复keys.shape[:
    # keys.shape[1]为最大的序列长度，即431，为了方便矩阵计算
    # [None, 431 * hidden_unit * 2]
    queries = tf.tile(queries, [1, keys.shape[1]])
    # 重塑候选物品embed的shape
    # [None, 431, hidden_unit * 2]
    queries = tf.reshape(queries, [-1, keys.shape[1], queries_hidden_units])
    # 拼接候选物品embed与hist物品embed
    # [None, 431, hidden * 2 * 4]
    embed = tf.concat([queries, keys, queries - keys, queries * keys], axis=-1)
    # 全连接，得到权重W
    d_layer_1 = self.att_dense1(embed)
    d_layer_2 = self.att_dense2(d_layer_1)
    # [None, 431, 1]
    d_layer_3 = self.att_dense3(d_layer_2)
    # 重塑输出权重类型，每个hist物品embed有对应权重值
    # [None, 1, 431]
    outputs = tf.reshape(d_layer_3, [-1, 1, keys.shape[1]])

    # Mask
    # 此处将为历史记录的物品embed令为True
```

```
        # [None, 1, 431]
        key_masks = tf.expand_dims(key_masks, 1)
        # 填充矩阵
        paddings = tf.ones_like(outputs) * (-2 ** 32 + 1)
        # 构造输出矩阵，其实就是为了实现【sum pooling】。True即为原outputs的值，False为上述填
        # [None, 1, 431] ----> 每个历史浏览物品的权重
        outputs = tf.where(key_masks, outputs, paddings)
        # Scale，keys.shape[-1]为hist_embed的隐藏单元数
        outputs = outputs / (keys.shape[-1] ** 0.5)
        # Activation，归一化
        outputs = tf.nn.softmax(outputs)
        # 对hist_embed进行加权
        # [None, 1, 431] * [None, 431, hidden_unit * 2] = [None, 1, hidden_unit * 2]
        outputs = tf.matmul(outputs, keys)
        return outputs
```

4、对候选广告embedding、经过sum pooling的历史记录embedding进行拼接：

```
    def call(self, inputs):
        ......
        item_embed = tf.reshape(item_embed, [-1, item_embed.shape[-1]])
        # 联合用户行为embedding、候选物品embedding、【用户属性、上下文内容特征】
        embed = self.concat2([u_embed, item_embed])
```

5、进行MLP过程

```
    def call(self, inputs):
        ......
        x = self.bn2(embed)
        x = self.dense1(x)
        x = self.activation1(x)
        x = self.dense2(x)
        x = self.activation2(x)
        x = self.dense3(x)
        outputs = tf.nn.sigmoid(x)
        return outputs
```

输入处理

处理，不过在【开源代码】中，作者是取每个batch_size中的所有用户中最长的历史记录长度作为矩阵的列数，但这里我们是取所有用户的最长(max_sl)，对长度不够的在最后进行添0处理【这样增加了内存消耗，但我不知道如何在TF2.0中如何处理】

```
def input_data(dataset, max_sl):
    user = np.array(dataset[:, 0], dtype='int32')
    item = np.array(dataset[:, 1], dtype='int32')
    hist = dataset[:, 2]
    hist_matrix = tf.keras.preprocessing.sequence.pad_sequences(hist, maxlen=max_sl, p

    sl = np.array(dataset[:, 3], dtype='int32')
    y = np.array(dataset[:, 4], dtype='float32')

    return user, item, hist_matrix, sl, y
```

训练

然后就是正常的进行模型编译、训练。

Github

上传了自己的github：

https://github.com/ZiyaoGeng/Recommender-System-with-TF2.0
github.com

并还实现了NCF的TF2.0实现。【大佬给个star吧】

微信公众号

推荐算法的小齿轮

编辑于 09-01