# 2019中国高校计算机大赛——大数据挑战赛TOP11解决方案



我们是 `lili` 团队，这次比赛我们成绩分别为：

初赛：41名

复赛A榜：14名

复赛B榜：11名

下面我将对我们的方法与模型做个总结。

## 比赛链接

https://www.kesci.com/home/competition/5cc51043f71088002c5b8840

## 赛题描述

搜索中一个重要的任务是根据query和title预测query下doc点击率，本次大赛参赛队伍需要根据脱敏后的数据预测指定doc的点击率，结果按照指定的评价指标使用在线评测数据进行评测和排名，得分最优者获胜。

## 评估指标

qAUC，qAUC为不同query下AUC的平均值，计算如下：

$$qAUC = \frac{sum(AUC_i)}{query\_num}$$

其中AUCi为同一个query_id下的AUC（Area Under Curve）。

最终使用qAUC作为参赛选手得分，qAUC越大，排名越靠前。

## 词向量

我们训练了训练集最后5亿的数据+测试集A榜2000w数据+测试集B榜1亿数据

最终使用的fasttext做的训练，训练时间13小时30分钟

参数如下:

```
model = fasttext.train_unsupervised(input_file,
                                    dim=100,
                                    minCount=5,
                                    ws=5,
                                    model='skipgram',
                                    verbose=2,
                                    thread=16)
```

### 遇到的坑

1. gensim不支持增量训练，或者说增量训练出来的没有用，还是查不到新词，具体参考如下
    1. https://github.com/Embedding/Chinese-Word-Vectors/issues/30
    2. https://www.zhihu.com/question/53093135
2. fasttext支持增量训练，但不支持Python版本，只支持C++版本，具体参考资料如下
    1. https://github.com/facebookresearch/fastText/pull/423
3. 大语料建议使用 `skipgram` ，而不是 `cbow`
4. 构建 `embedding_matrix` 记得要把测试集的语料也放进去
5. 使用gensim可以使用 `word2vec_model.wv.get_keras_embedding(train_embeddings=False)` 直接生成 `embedding` 层
6. fasttext判断一个词存不存在不要直接使用 `word in model` ，正确的方法如下

```
w2v_set = set(w2v.get_words()) # w2v即为fasttext模型
if word not in w2v_set:
    ...
```

### Tricks

1.使用 `shell` 中的 `cut` 方法来切分数据集，而不要使用 `python` ，效率提高N倍，比如我要切分query(第二列)和title(第四列)出来，分割是 `,` ，然后再合并，例子如下：

```
cut -f 2 -d ',' /home/kesci/input/bytedance/train_final.csv >
/home/kesci/word2vec_file/train_data_query
cut -f 4 -d ',' /home/kesci/input/bytedance/train_final.csv >
/home/kesci/word2vec_file/train_data_title
cat /home/kesci/word2vec_file/train_data_query \
/home/kesci/word2vec_file/train_data_title \
> /home/kesci/word2vec_file/all_sentence
```

2.对于大语料训练，gensim建议使用 `LineSentence` 或者 `PathLineSentences` 方法

3.fasttext支持 `子词嵌入` ，所以训练不到的词也能计算出来，但效果肯定比不上训练过的，参考资料

- https://zh.d2l.ai/chapter_natural-language-processing/fasttext.html

# 传统模型

## 特征

我们最终特征维度为28维，但我们不止提取了那么多，具体划分维基础特征8维、fuzz特征8维、距离特征10维、额外特征8维最后rank特征9维。

- 基础特征
  - 句子长度
  - 句子长度差
  - 字符(char)长度
  - 词(word)长度
  - quey和title的common word个数
- fuzz特征
  - fuzz_qratio
  - fuzz_WRatio
  - fuzz_partial_ratio
  - fuzz_partial_token_set_ratio
  - fuzz_partial_token_sort_ratio
  - fuzz_token_set_ratio
  - fuzz_token_sort_ratio
- 距离特征
  - cosine
  - cityblock
  - canberra
  - euclidean
  - minkowski
  - braycurtis
  - skew_q
  - skew_t
  - kurtosis_q
  - kurtosis_t
- 额外特征
  - query与title的第一个word是否相同
  - query与title的前三个word是否相同

    - query_nunique_title，query下title的个数
    - title_nunique_query，title下query的个数
    - quey是否在title里
    - quey与title的Levenshtein ratio
    - quey与title的Levenshtein distance
- rank特征
    - fuzz所有特征的rank排名
    - quey与title的Levenshtein ratio的rank
    - quey与title的Levenshtein distance的rank

上面所有特征一共80维，也是我们初赛最终的特征，但是在复赛中，该套方案不可行，原因可能是数据量大了导致fuzz特征失效，也有可能是我们复赛开始词向量出现的问题。

### 复赛最终特征

```
['q_id', 't_id', 'len_q', 'len_t',
'diff_len', 'len_char_q', 'len_char_t', 'len_word_q', 'len_word_t',
'common_words','cosine', 'cityblock',
'canberra', 'euclidean', 'minkowski', 'braycurtis', 'skew_q', 'skew_t',
'kurtosis_q', 'kurtosis_t', 'is_first_same', 'is_third_same',
'common_words_cnt', 'query_nunique_title', 'title_nunique_query',
'query_isin_title', 'title_query_ratio_list',
'title_query_distance_list', 'query_nunique_title_rank',
'title_nunique_query_rank']
```

这里删除了fuzz特征，因为我们发现FuzzyWuzzy删除后我们的线上得分从0.58770300涨到了0.58882200。

最后加上距离特征，线上到了0.58953500，最后将数据量增加到2亿，之前一直是1亿，线上得到了我们LGB最终的分数0.59238400

### 特征并行提取

由于复赛数据量极大，所以初赛的单进程特征提取方式已经不再适用，如果还按照初赛的方法 `apply(balabala, axis=1)` ,那提取完特征，比赛也就结束了，所以我们选择python的多进程并行方式，使用到了 `multiprocessing` 库，当然，我们也尝过使用Python的多线程，也就是 `threading` ，但由于 `GIL` 全局解释器锁，导致每个CPU在同一时间只能执行一个线程，所以并没有什么效果，所以我们采用了多进程的方法。

我们将需要行计算的特征，也就是上面的fuzz特征、距离特征等，采用边读边写，多核(16核)并行的方法，能把CPU跑满，而几乎占内存，但对硬盘的IO读写要求很高，速度大概是5500条/s，一亿数据的特征5个小时就能跑完，下面讲具体方法：

1.使用 `shell` 的 `split` 方法分割数据，举个例子，我把1亿数据分成16份，也就是每份6250000行，具体代码如下

```
split -l 6250000 /home/kesci/work/tf_data/train_data_9.csv -d -a 2
/home/kesci/work/tf_data/train_data_9_
```

2.下面就是多进程提取特征的具体方法

首先记录日志

```
logger = logging.getLogger()
fhandler = logging.FileHandler(filename='vector_fea_gen.log', mode='w')
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
fhandler.setFormatter(formatter)
logger.addHandler(fhandler)
logger.setLevel(logging.DEBUG)
```

然后编写提取特征代码

```
def cal_vector_fea(filename):
    s = time.time()
    out = open(FEATURE_PATH+"/vector_feature/"+filename+"_out", "w")
    with open(PATH_SAVE_DATA+"/"+filename) as f:
        logger.debug("开始: "+filename)
        for idx, line in enumerate(f):
            if idx%1E+6 == 0:
                logger.debug(filename + ": %d行"%idx)
            one_r = []
            line = line.strip().split(",")
            q_id = line[0]
            t_id = line[2]
            one_r.append(q_id)
            one_r.append(t_id)
            q_vec = sent2vec(line[1])
            t_vec = sent2vec(line[3])
            one_r.append(cosine(q_vec, t_vec))
            one_r.append(cityblock(q_vec, t_vec))
            one_r.append(canberra(q_vec, t_vec))
            one_r.append(euclidean(q_vec, t_vec))
            one_r.append(minkowski(q_vec, t_vec))
            one_r.append(braycurtis(q_vec, t_vec))
            one_r.append(skew(q_vec))
            one_r.append(skew(t_vec))
            one_r.append(kurtosis(q_vec))
            one_r.append(kurtosis(t_vec))
            out.write(",".join([str(r) for r in one_r])+"\n")
            # break
    out.close()
    logger.debug(str(time.time()-s))
```

然后将生成多进程实例

```
process_list = []
for i in range(16):
    filename = "test_data_%02d"%i
    process_list.append(multiprocessing.Process(target = cal_vector_fea, args = (filename,)))
```

最后，发射！

```
for p in process_list:
    p.start()
```

这时候，你的CPU就是嗡嗡的跑起来了，我们可以用 `wc` 方法，看提取行数的速度大概是多少

```
wc -l /home/kesci/work/features/vector_feature/test_data_00_out
```

如果行数在不断增加，基本就没什么问题了，当然做前还是 `break` 测试一下为好。

## LightGBM

### 参数

```
lgb_params = {
    "learning_rate": 0.1,
    "lambda_l1": 0.1,
    "lambda_l2": 0.2,
    "max_depth": -1,
    "num_leaves": 30,
    "objective": "binary",
    "verbose": 100,
    'feature_fraction': 0.8,
    "min_split_gain": 0.1,
    "boosting_type": "gbdt",
    "subsample": 0.8,
    "min_data_in_leaf": 50,
    "colsample_bytree": 0.7,
    'device':'gpu',
    'gpu_platform_id':0,
    'gpu_device_id':0
}
```

### 训练

我们采用 `StratifiedKFold` 分层采样法，5折交叉，只训练第3个fold，所以验证集为20%

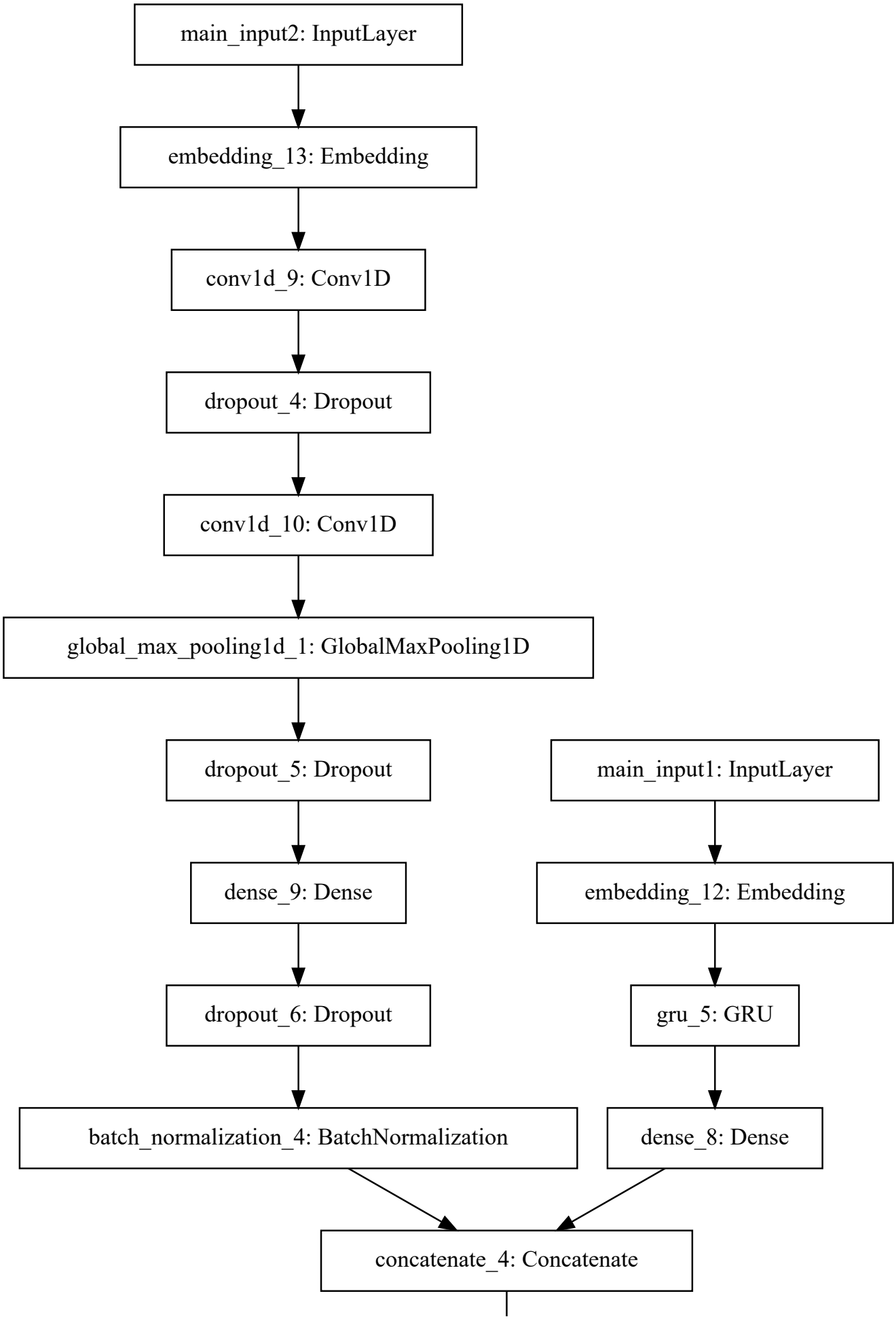
分别跑了

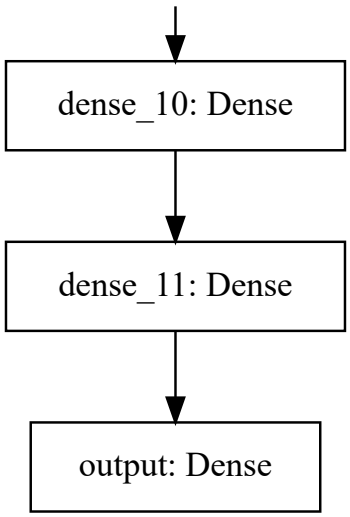
- 1亿数据，线上0.58770300
- 2亿数据，线上0.59216000
- 4亿数据，线上0.59234700

收益越来越小

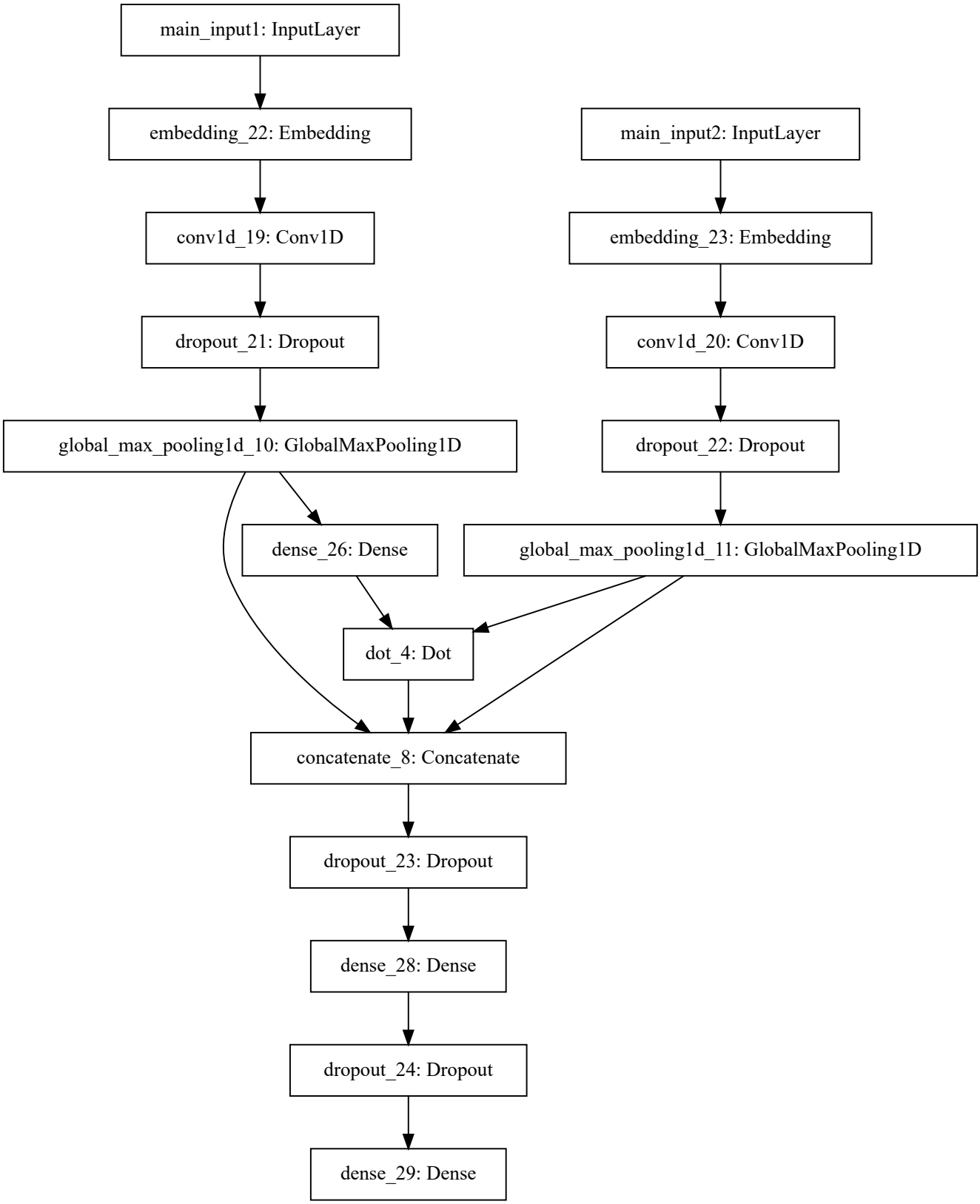
## 深度模型

### 双输入CNN+GRU

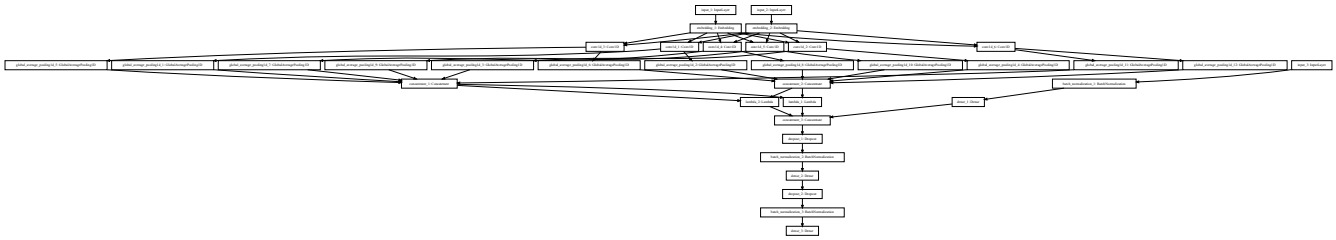

main_input2: InputLayer
embedding_13: Embedding
conv1d_9: Conv1D
dropout_4: Dropout
conv1d_10: Conv1D
global_max_pooling1d_1: GlobalMaxPooling1D
dropout_5: Dropout
dense_9: Dense
dropout_6: Dropout
batch_normalization_4: BatchNormalization
main_input1: InputLayer
embedding_12: Embedding
gru_5: GRU
dense_8: Dense
concatenate_4: Concatenate




PairCNN


main_input1: InputLayer
embedding_22: Embedding
conv1d_19: Conv1D
dropout_21: Dropout
global_max_pooling1d_10: GlobalMaxPooling1D
main_input2: InputLayer
embedding_23: Embedding
conv1d_20: Conv1D
dropout_22: Dropout
dense_26: Dense
global_max_pooling1d_11: GlobalMaxPooling1D
dot_4: Dot
concatenate_8: Concatenate
dropout_23: Dropout
dense_28: Dense
dropout_24: Dropout
dense_29: Dense

CNN 1D



最后一个模型也就是我们最终使用的模型，单模型就能在A榜达到0.61807700，使用数据为最后一亿， `input3` 手工特征也就是使用之前提取的28维特征。

最后B榜也是使用的这个单模型，LGB融合后没有提升，可能是LGB太差了，融合方法使用的RankAVG，具体代码如下

```
data1 = pd.read_csv(RESULT_PATH+"/mysubmission_20190809_lgb_add_vec_2e.csv",header = None)
data1.columns = ['query_id', 'query_title_id', 'prediction']
data2 = pd.read_csv(RESULT_PATH+"/mysubmission_201908011_nn.csv",header = None)
data2.columns = ['query_id', 'query_title_id', 'prediction']
prediction_rank_1 = data1.groupby("query_id").rank(ascending=True, pct=True)['prediction']
prediction_rank_2 = data2.groupby("query_id").rank(ascending=True, pct=True)['prediction']
prediction_rank = prediction_rank_1 * 0.7 + prediction_rank_2 * 0.3
```

## 总结

这次比赛是第一次接触NLP赛题，从5月份就开始，一直到8月才结束，大概2个多月的时间，学到了很多新东西，尤其是认识到了NN的强大，初赛的时候不提供GPU，我们在NN上面耗费了大量的时间，速度慢，效果也不好，最后的时刻才改用LGB，成功上分，进入复赛。复赛的时候，我们又死磕LGB，在LGB上面耗费了大量的时间，效果很差，而且天花板很明显，有些特征真的很难去想出来，到了最后时刻，才改用NN成功上分，虽然就差一名进入决赛，多少还是有些不甘，要是早点改用NN，可能会有不一样的结果，但能得到这个名次也很开心了，下次继续加油吧。