**NL2SQL竞赛解析**

**Written by siyu1992**

# 业务及技术背景

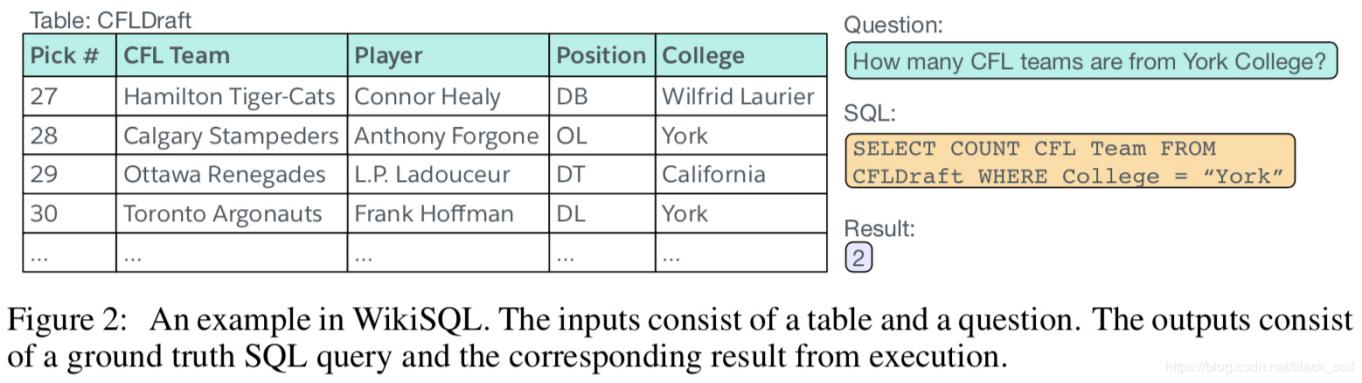
NL2SQL是CUI（Conversation User Interface）的新兴研究热点，其研究目的是将用户输入的自然语言转为可用的SQL语句，是语义分析 (Semantic Parsing) 领域的一个子任务。NL2SQL在推理机、聊天机器人等应用上有很大前景。

NL2SQL的历史悠久，早在1973年，Woods等人就开发了一个名为LUNAR的系统，可以回答关于从月球带回的岩石样本的问题。到了1978年，Hendrix设计了一个连接美国海军舰艇信息数据库的自然语言接口，名为LIFER/LADDER。这些系统仅仅支持特定数据库的单表操作，受限于当时的技术发展，NL2SQL并没有受到太多的关注。直到2015年AI的复苏和NLP的发展，它渐渐回到人们的视线。

目前比较火的英文数据集有WikiSQL、Spider、WikiTableQuestions、ATIS等，中文数据集有刚刚结束的中文首届NL2SQL挑战赛公开的数据，各个数据集都有各自的特点，下面简单介绍下这几个数据集：

**1. WikiSQL**

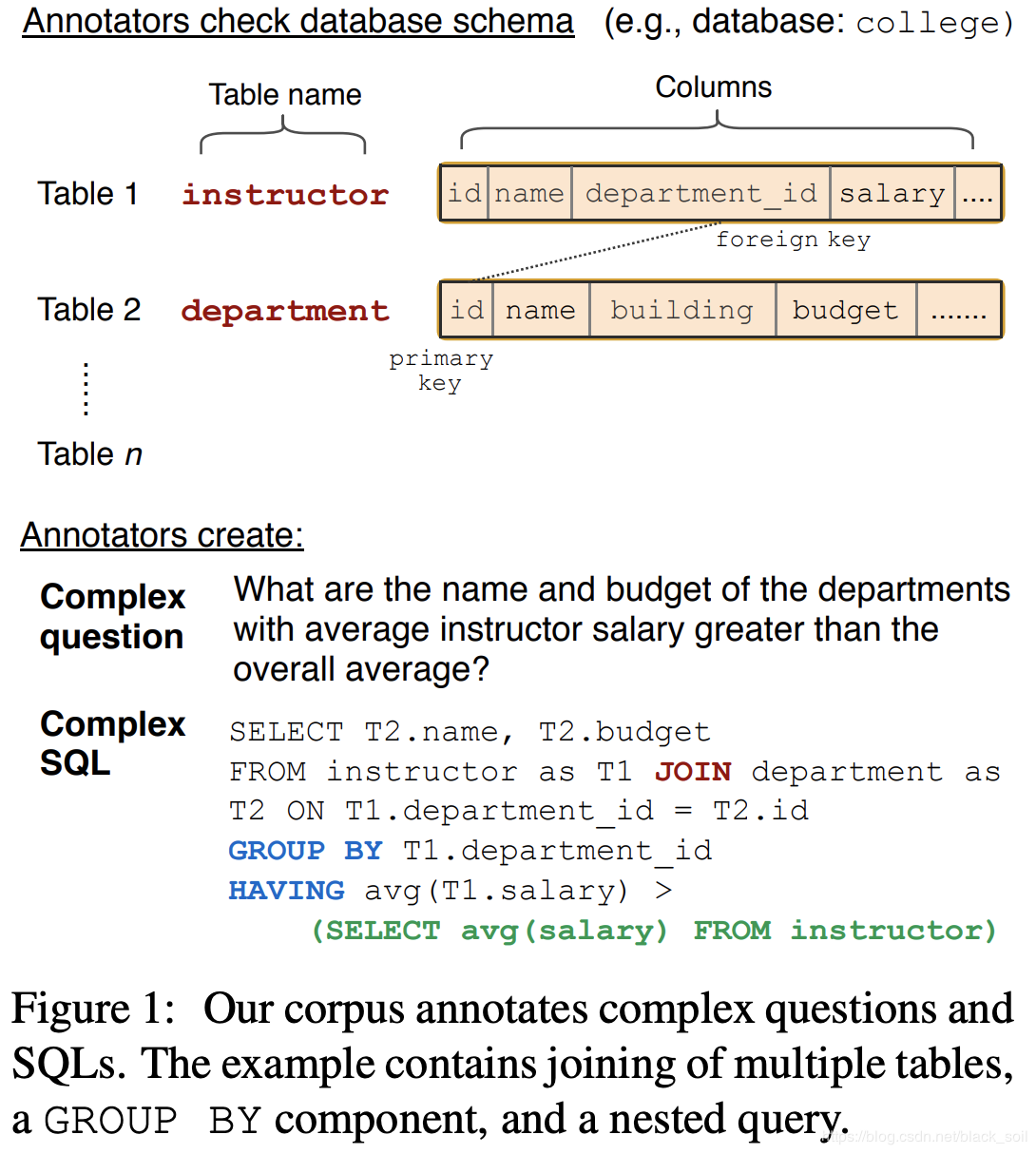
WikiSQL数据集是Salesforce在2017年提出的大型标注NL2SQL数据集，也是目前规模最大的NL2SQL数据集。它包含了 24,241张表，80,645条自然语言问句及相应的SQL语句。目前学术界的预测准确率可达91.8%。样例如下图。



WikiSQL的问题长度8~15个词居多，查询长度8~11个词居多，表的列数5~7个居多，另外，大多数问题是what类型，其次是which、name、how many、who等类型。

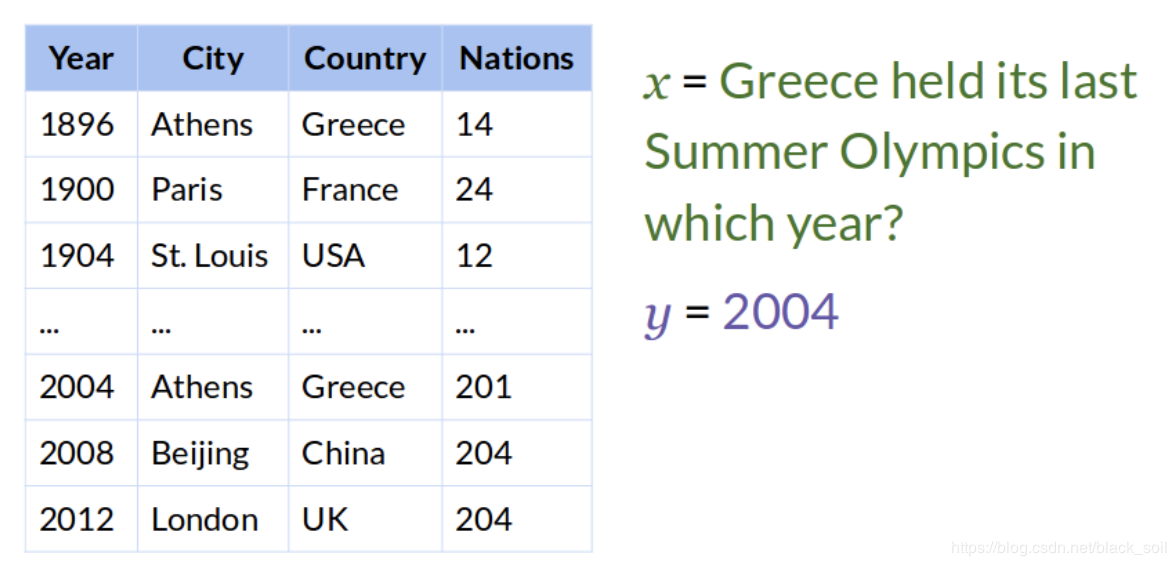
**2. Spider**

Spider数据集是耶鲁大学于2018年新提出的一个较大规模的NL2SQL数据集。该数据集包含了10,181条自然语言问句，分布在200个独立数据库中的5,693条SQL，内容覆盖了138个不同的领域。虽然在数据数量上不如WikiSQL，但Spider引入了更多的SQL用法，例如Group By、Order By、Having等高阶操作，甚至需要Join不同表，更贴近真实场景，所以难度也更大。目前准确率最高只有54.7%。样例数据如下：



3. WikiTableQuestions

该数据集是斯坦福大学于2015年提出的一个针对维基百科中那些半结构化表格问答的数据集，内部包含22,033条真实问句以及2,108张表格。由于数据的来源是维基百科，因此表格中的数据是真实且没有经过归一化的，一个cell内可能包含多个实体或含义，比如「Beijing, China」或「200 km」；同时，为了很好地泛化到其它领域的数据，该数据集测试集中的表格主题和实体之间的关系都是在训练集中没有见到过的。样例数据如下：



**4. ATIS**

ATIS（The Air Travel Information System）是一个年代较为久远的经典数据集，由德克萨斯仪器公司在1990年提出。该数据集获取自关系型数据库Official Airline Guide (OAG, 1990)，包含27张表以及不到2,000次的问询，每次问询平均7轮，93%的情况下需要联合3张以上的表才能得到答案，问询的内容涵盖了航班、费用、城市、地面服务等信息。

数据下载地址：

WikiSQL：https://github.com/salesforce/WikiSQL

Spider：https://yale-lily.github.io/spider

ATIS：https://www.kaggle.com/siddhadev/ms-cntk-atis

WikiTableQuestions：https://github.com/ppasupat/WikiTableQuestions

# NL2SQL比赛介绍

**竞赛题目**

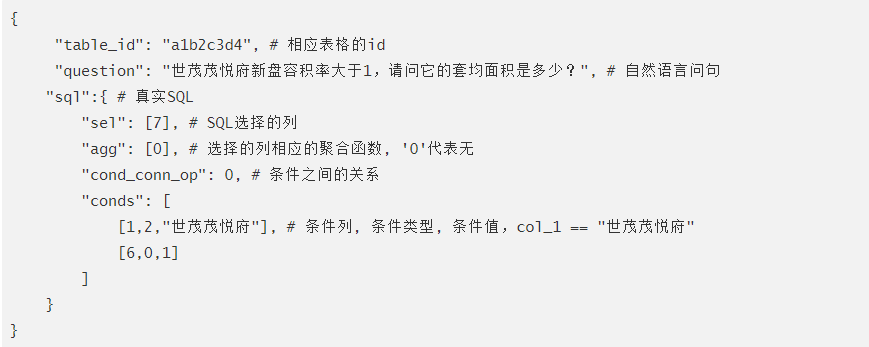
首届中文NL2SQL挑战赛，使用金融以及通用领域的表格数据作为数据源，提供在此基础上标注的自然语言与SQL语句的匹配对，希望选手可以利用数据训练出可以准确转换自然语言到SQL的模型。

**数据说明**

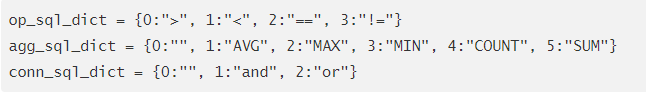
本次赛题将提供4万条有标签数据作为训练集，1万条无标签数据作为测试集。其中，5千条测试集数据作为初赛测试集，对选手可见；5千条作为复赛测试集，对选手不可见。

提供的数据集主要由3个文件组成，以训练集为例，包括train.json、train.tables.json及train.db。

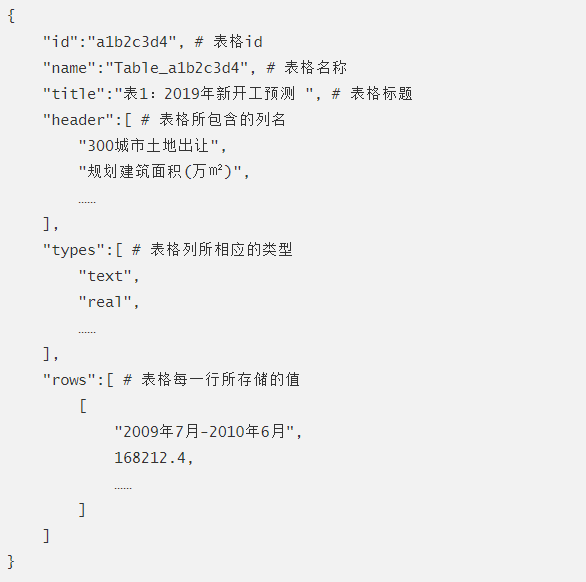
train.json文件中，每一行为一条数据样本。数据样例及字段说明例如下：



其中，SQL的表达字典说明如下：



train.tables.json 文件中，每一行为一张表格数据。数据样例及字段说明例如下：



tables.db为sqlite格式的数据库形式的表格文件。各个表的表名为tables.json中相应表格的name字段。为避免部分列名中的特殊符号导致无法存入数据库文件，表格中的列名为经过归一化的字段，col\_1, col\_2, …, col\_n。db文件将后续更新。

另外，也提供用于baseline方案的字向量文件char\_embedding，每一行的内容为字符及其300维的向量表达，以空格分隔。

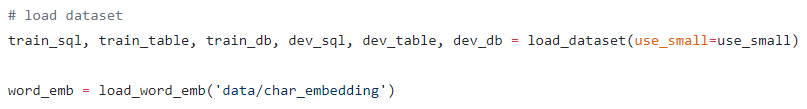
# Baseline详解

**数据加载**

数据样例请参考比赛介绍中的数据样例，数据可以在比赛链接中下载，比赛地址：

<https://tianchi.aliyun.com/competition/entrance/231716/introduction>

数据加载方法：通过util.py工具脚本中的load\_dataset和load\_word\_emb分别加载数据与向量，baseline中用到的词嵌入方式为character embedding：

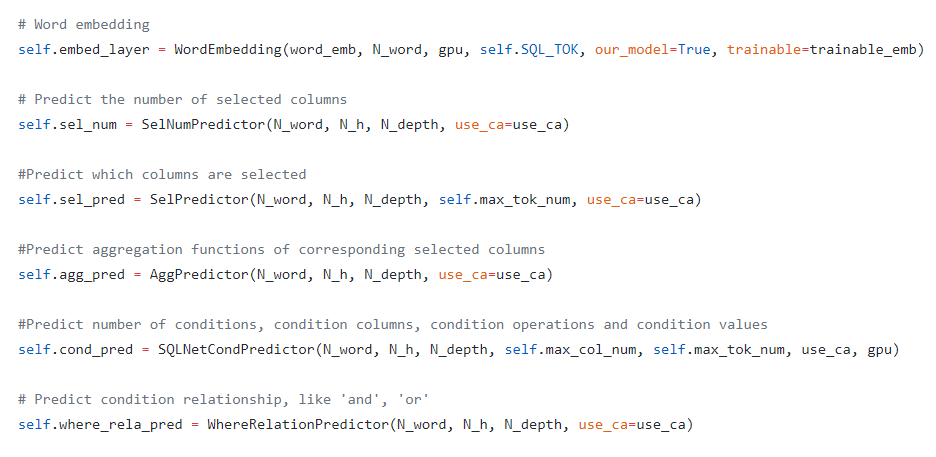


**Baseline模型类(SQLNet())**

SQLNet() 类中实例化方法如下：



SQLNet类中实例化了词嵌入、各子任务网络实现的类，如下：



由图可知，sel\_num, sel\_pred为判断选择列(select-column)任务的两个子任务的logits；agg\_pred为判断选择聚合（select-aggregation）任务的对象；cond\_pred为判断条件列（condition-column）任务的对象；where\_rela\_pred为判定条件编号（condition-number）任务的对象；

定义了前向传播方法，forward(),如下：



定义了loss计算的方法，如下：



定义了计算准确率的方法，如下：



定义了生成查询sql的方法，如下：



**Baseline项目目标**

该项目是将自然语言处理转化为mysql语句，采用的模型为SQLNet，该方法也是WikiSQL的baseline模型，原论文：<https://arxiv.org/pdf/1711.04436.pdf>

**Baseline任务解析**

SQLNet不同于seq2seq的架构，这里采用的是类似完形填空的思想：通读一句话，在指定位置添上对应属性名。

基本思想：将任务分成两部分来预测：

1. 这里采用的是类似完形填空的思想：通读一句话，在指定位置添上对应属性名。

2. 将整个任务分成两大部分来预测：1.预测Where子句 2. 预测Select子句

3. 在上述两大部分下面又存在着几个小模块（子任务）用于处理子句中各个位置的填充任务：

由于SQL语言由诸多关键字构成， 因此分解为多个子任务，包括选择编号（select-number），选择列(select-column)，选择聚合（select-aggregation），条件编号（condition-number），条件列（condition-column）。

4. 最后将这些模块的loss，综合起来，就是整句话的loss

**Baseline目标（SQL结构构成）：**

select $agg {0:"", 1:"AVG", 2:"MAX", 3:"MIN", 4:"COUNT", 5:"SUM"}

$column

where

$column $op{0:">", 1:"<", 2:"", 3:"!="}

$conn\_sql\_dict{0:"", 1:"and", 2:"or"}

$column $op{0:">", 1:"<", 2:"", 3:"!="}

我将SQL结构做了一些规范化，关键字部分都没有缩进，涉及到字段和条件内容的是有缩进的，方便理解SQL结构。

可以将其对应到以下子任务中：

1. 判断选择聚合（select-aggregation），即上模板中$agg，判断聚合函数类型，如果没有聚合，就是“”空。

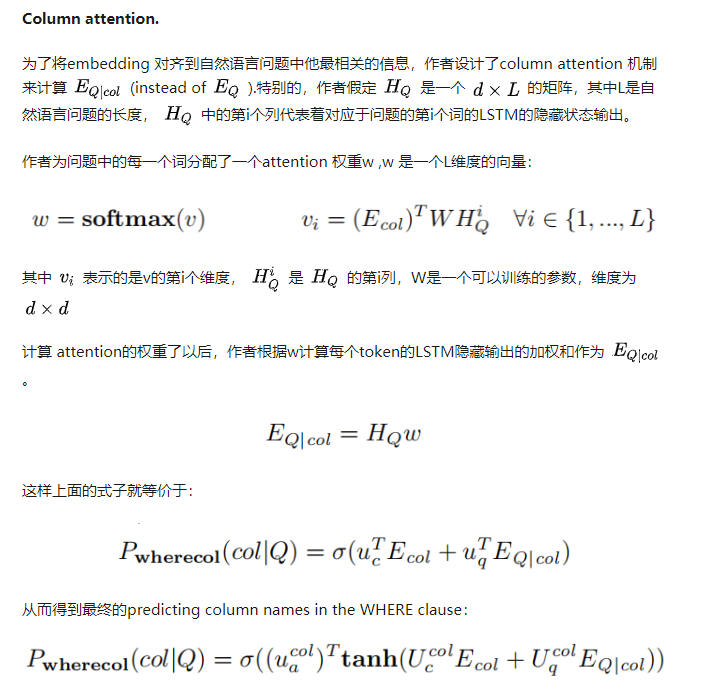
2. 判断选择列(select-column)，对应上面模板中绿色标注的$column, 该任务又可以分为列个数sel\_num和列值sel\_pred

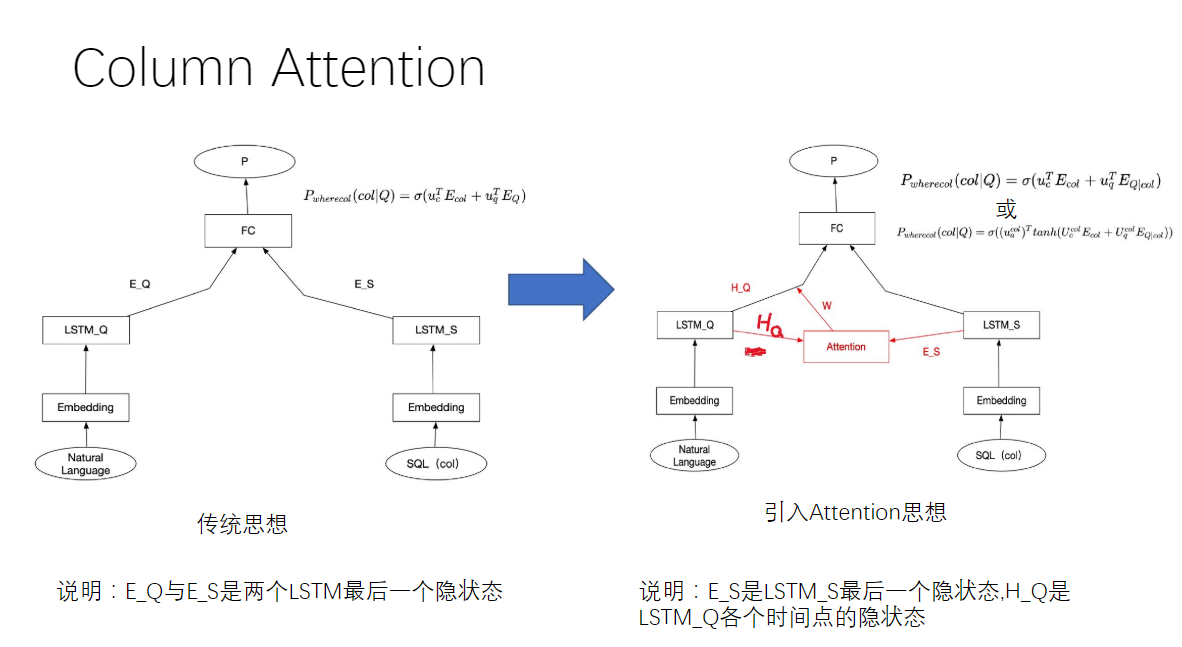
3. 判断条件列（condition-column），对应上面模板中黄色标注的$column, 也就是where后面条件语句的词；判断条件操作符，即对应$op。

4. 判定条件连接符（where-column），也就是where后面条件的组合关系，and或者or或者“”

**SQLNet网络结构解析：**

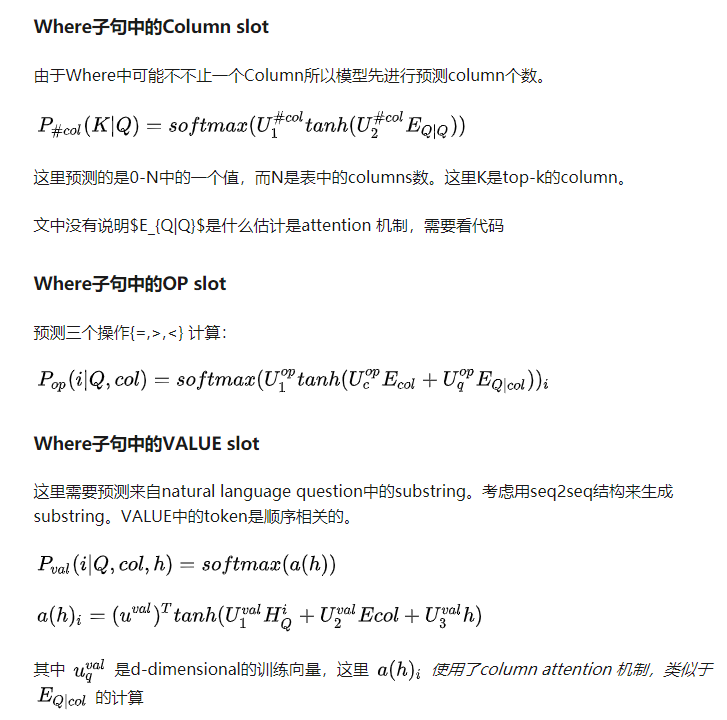
1.Column attention：



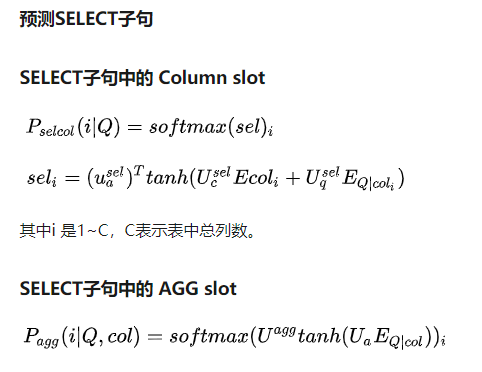


2.完整模型及训练细节

1. 预测Where子句



2. 预测Select子句

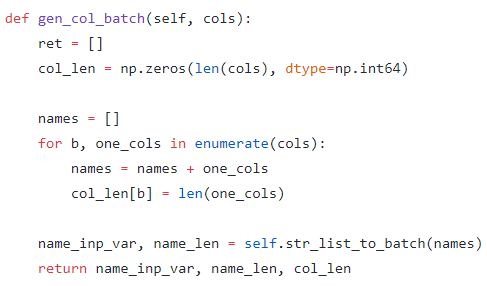


**功能接口：**

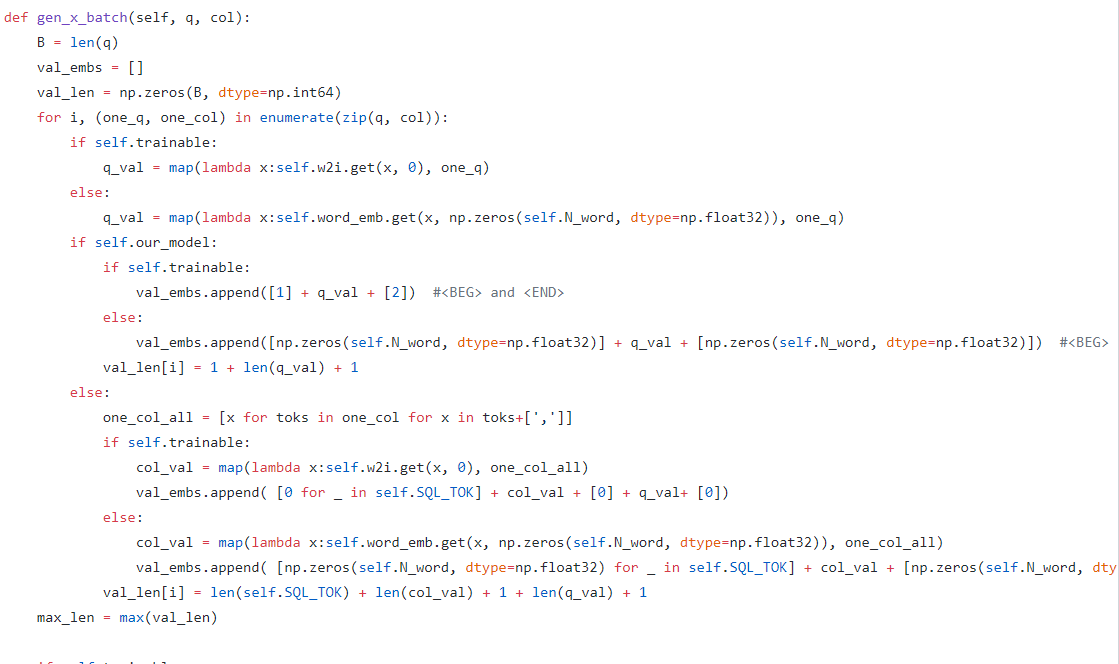
1. 词嵌入层（word embedding layer）: 该Baseline用的character embedding，训练方法不过多介绍。

Word embedding layer接口：

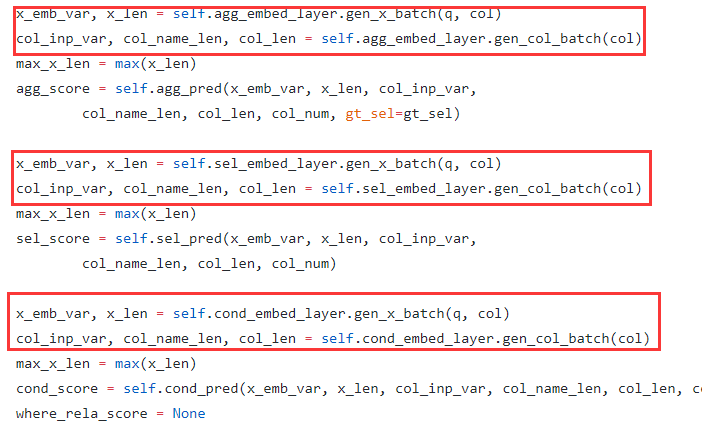
1. gen\_col\_batch: 切分col batch的方法



1. Gen\_x\_batch：切分训练数据batch的方法



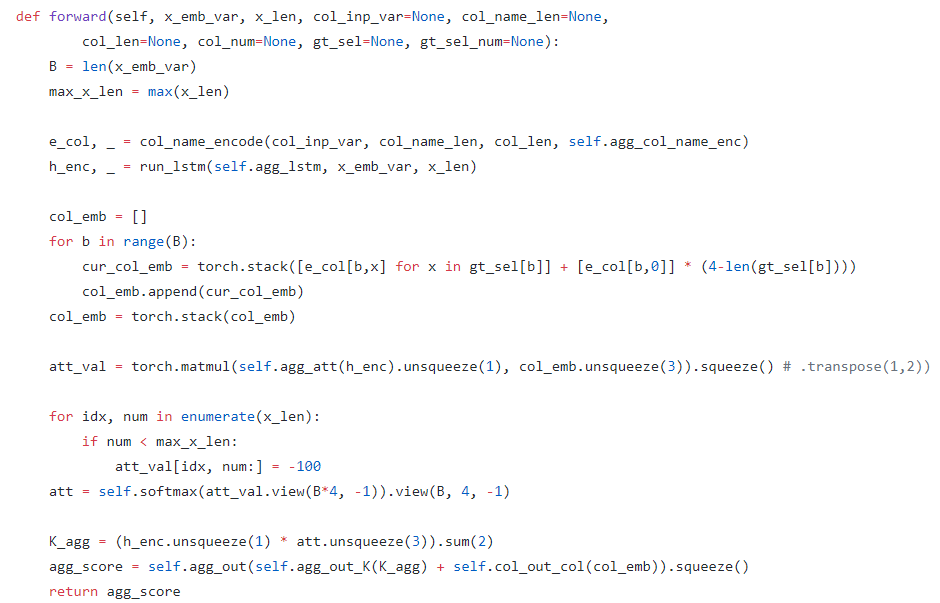
方法执行的顺序一般是：



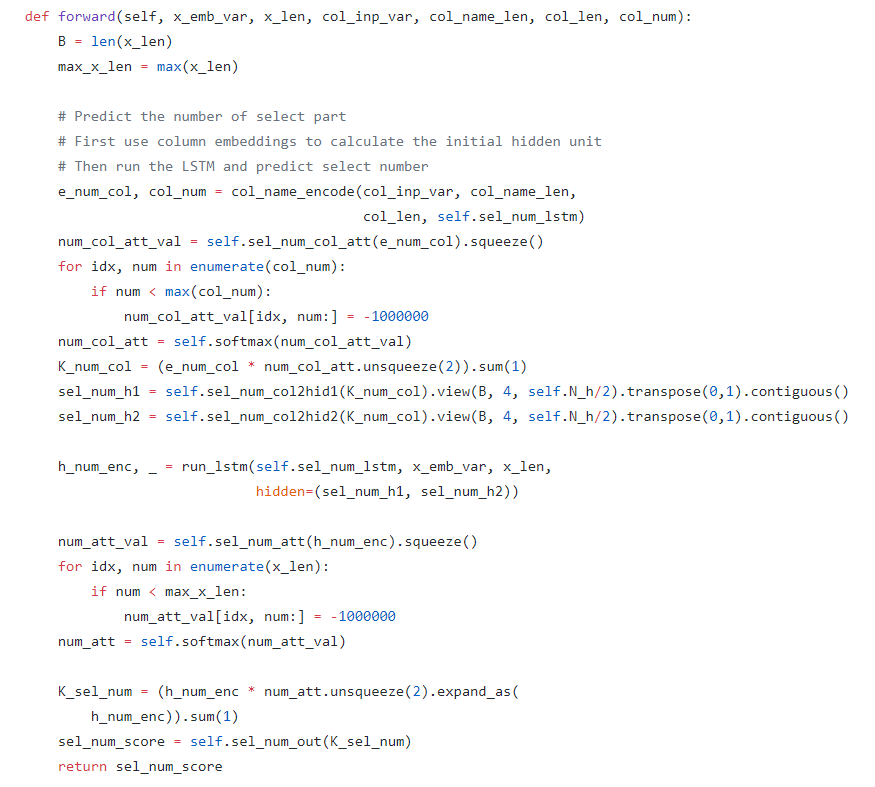
1. 前向传播层（forward layer）：SQLNet是一个joint model，将上述子问题整体进行优化，对于forward部分来说，整个网络的前向得分score分为：sel\_num\_score, sel\_score, agg\_score, cond\_score, where\_rela\_score。对于这些score，SQLNet使用不同的loss来进行组合，最后通过AdamOptimizer进行优化

**最细粒度子任务技术实现：**

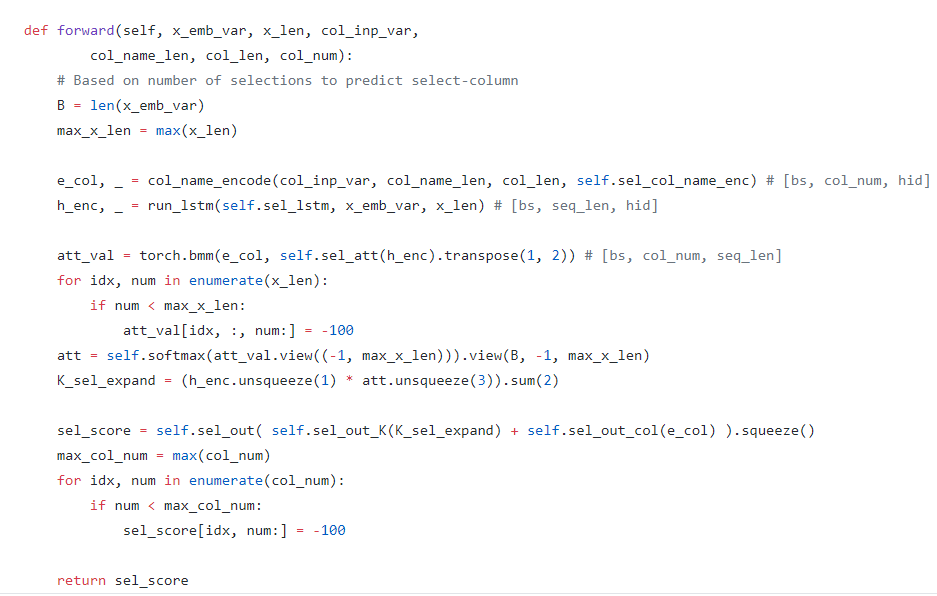
1. aggregator\_predict：判断选择聚合（select-aggregation）



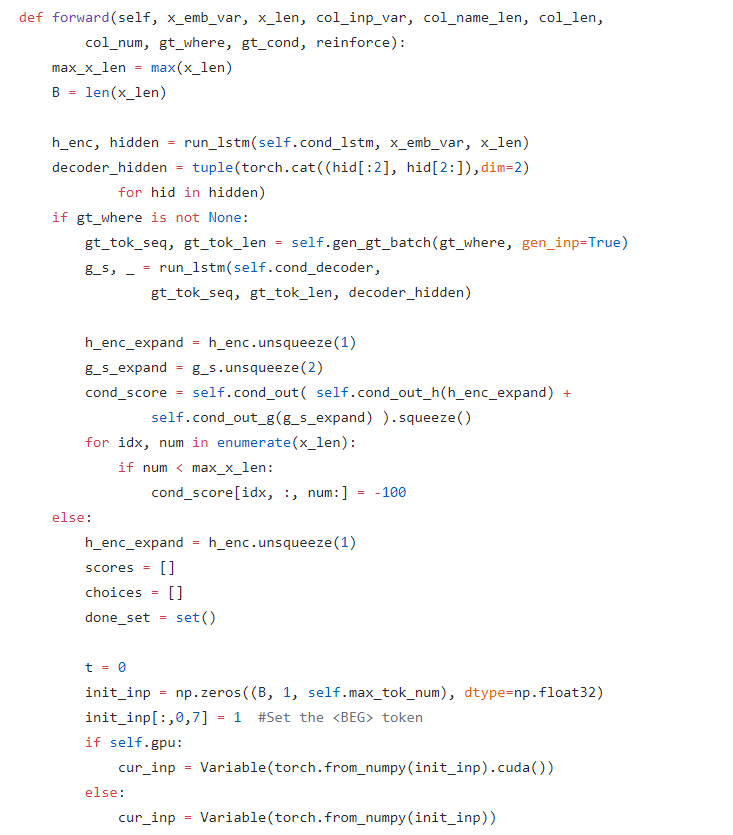
1. select\_number：判断选择列(select-column)中的sel\_num

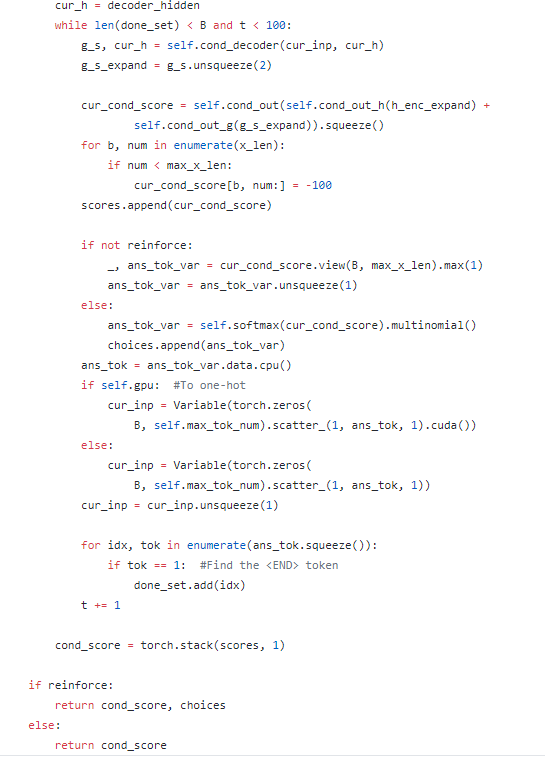


1. selection\_predict：判断选择列(select-column)中的sel\_pred

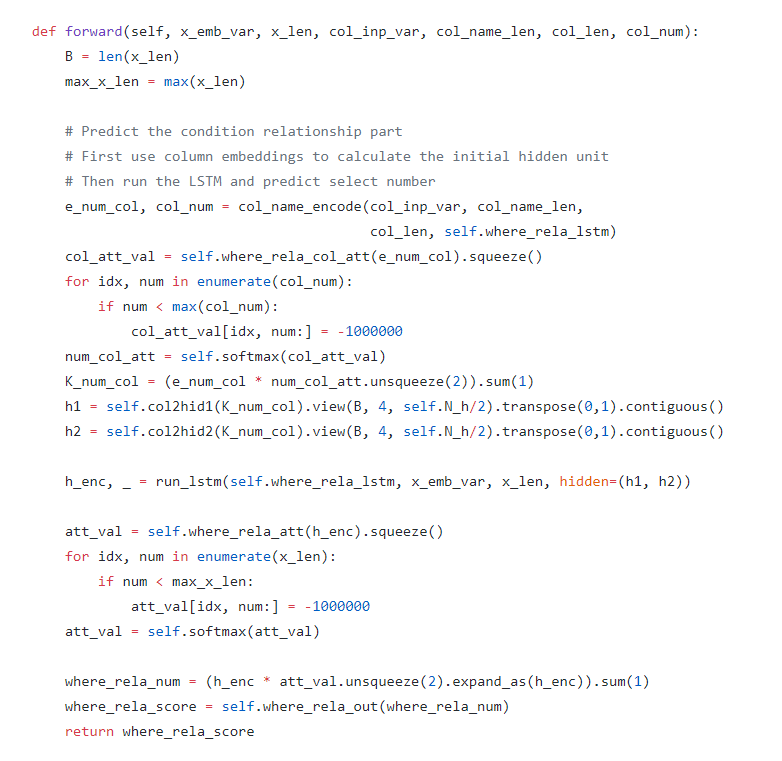


1. seq2sql\_condition\_predict：判断条件列（condition-column）





1. where\_relation：判定条件连接符（where-column）

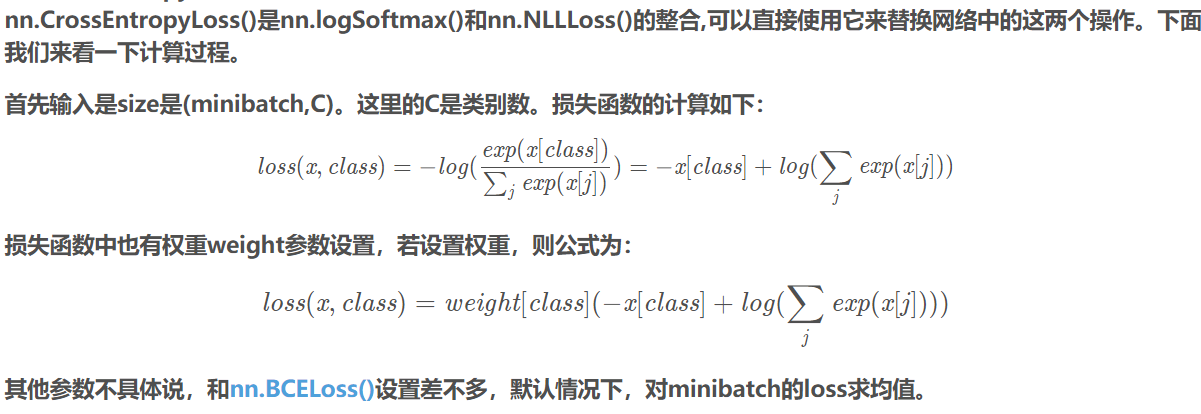


3.损失层(loss layer)：

**SQLNet 损失函数**：

**SQLNet中用到的loss function**：

1. self.CE = nn.CrossEntropyLoss()：



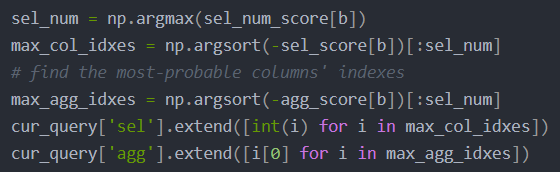
1. nn.BCEWithLogitsLoss()：



**SQLNet中loss计算方法**：

针对各个score，score之间的计算关系是：

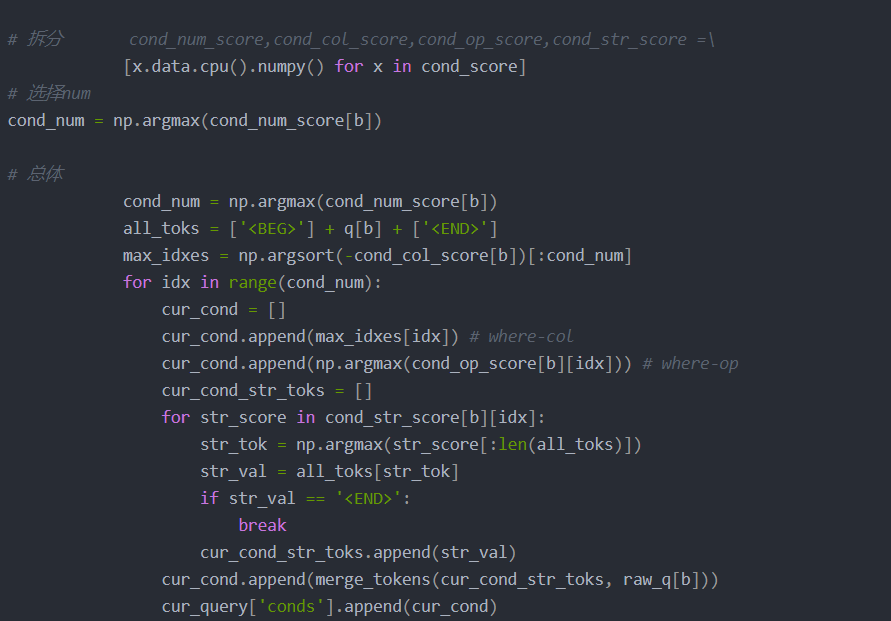
1. 根据sel\_num来选定select后面columns的列和agg：



1. where\_rela\_score自力更生：



1. cond也就是where后面的数量num，列col，op，value其实也是自力更生。



Loss计算部分代码：

1. loss += self.CE(sel\_num\_score, sel\_num\_truth)
2. bce\_loss=-torch.mean(3\*(cond\_col\_truth\_var\*torch.log(cond\_col\_prob+1e-10))+(1-cond\_col\_truth\_var)\*torch.log(1-cond\_col\_prob+1e-10) )

loss+= bce\_loss

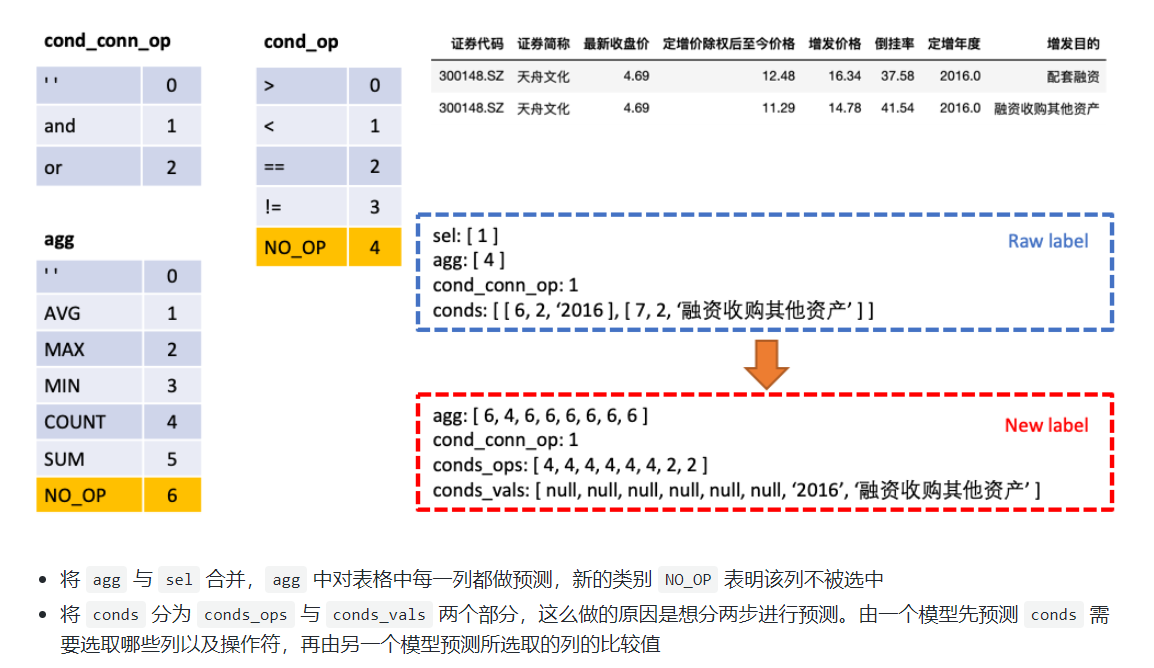
1. loss += (self.CE(cond\_op\_pred, cond\_op\_truth\_var) / len(truth\_num))
2. loss += (self.CE(cond\_str\_pred, cond\_str\_truth\_var) / (len(gt\_where) \* len(gt\_where[b])))
3. loss += self.CE(where\_rela\_score, where\_rela\_truth)

**训练方法细节**

1.不同slots之间不共享权重，但要共享embedding

2.允许embedding更新，实验证明效果更好

# 比赛第三名solution



**Model 1**

Model 1 将 Question 与 Header 顺序连接，在每个 Column 之前添加一个特殊标记，TEXT 或 REAL，这两个特殊的 Token 可以从 BERT 预留的未训练过的 Token 中任选两个来代替。

**Model 2**

Model 2 则负责 cond\_val 的预测。我们的思路是根据 Model 1 选择的 cond\_col，枚举 cond\_op 与 cond\_val，生成一系列的候选组合，将这些组合当成多个二分类问题来做。

由于作者已将详细思路及代码放在了github上，这里不再做解析，第三名solution较有参考意义。

github地址：<https://github.com/beader/tianchi_nl2sql>

# 比赛第一名solution

第一名solution并没有开源出代码，所以需要按照ppt内容自行复现，但是第三名的解决方案及代码已经比较完整了，第一名solution显得没有那么有价值，不过可以参考PPT，感兴趣可以复现

github链接：

<https://github.com/nudtnlp/tianchi-nl2sql-top1/blob/master/%E5%A4%A9%E6%B1%A0NL2SQL%E5%86%A0%E5%86%9B%E6%96%B9%E6%A1%88.pdf>

References：

1. <https://zhuanlan.zhihu.com/p/70408854>
2. <https://zhuanlan.zhihu.com/p/71955744>
3. <https://github.com/ZhuiyiTechnology/nl2sql_baseline>