CoSQL 测试报告

蒋仕彪　　李广林
3170102587@zju.edu.cn　　3170104648@zju.edu.cn

日期：2020 年 6 月 28 日

摘　　要

本文是 2020 年春夏学期《机器学习》课程的期末论文。我们小组聚焦到 Text2SQL 领域的 CoSQL 数据集上，对当下的两个经典算法进行了复现。我们在强化 SOTA 时遇到了复杂的显存问题，并得到了一个折衷的解决方法。我们还应用了数据增强的手段使结果进一步提高。

关键词： Text2SQL　CoSQL　深度学习　EditSQL　复现　显存溢出　数据增强

1　CoSQL 数据集简介

CoSQL 全称 **Co**nversational text-to-**SQL**，是耶鲁大学在 EMNLP2019 提出的 NLP 领域的数据集[1]。与经典的 text-to-SQL 任务（如 ATIS 或 Spider）相比，CoSQL 的难度增加了不少：

1. 为了模拟现实场景，用户的询问可能有多轮，要求系统有整合信息的能力。
2. 系统生成 SQL 语句并得到查询结果后，要用自然语言反馈给用户。
3. 用户与系统的多轮对话之间，可能需要 clarify ambiguous questions（如下图 $Q3$ 和 $R3$）。

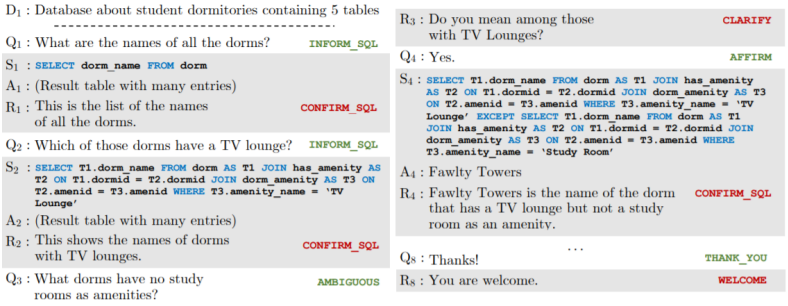

Figure 1: A dialog from the CoSQL dataset. Gray boxes separate the user inputs (Q_i) querying the database (D_i) from the SQL queries (S_i), returned answers (A_i), and expert responses (R_i). Users send an input to the expert, who writes the corresponding SQL query (only seen by the expert) if possible and sends an answer and response description back. Dialogue acts are on the right-hand side (e.g., Q_3 is "ambiguous" and R_3 is "clarify").

图 1: 一整轮 CoSQL 对话的实例

CoSQL 包含到 3000+ 组对话（2164 Train，292 Dev，551 Test）。共计 10000+ 条标注过的 SQL 问答。值得注意的是，CoSQL 的内容横跨 200 个数据库，而且不同组数据所用到的数据库**没有交集**（140 Train，20 Dev，40 Test），以考察模型的鲁棒性。

CoSQL 一共有以下三个任务：

1. **SQL-grounded dialogue state tracking**：给出 interaction history，求每个问题对应的 SQL 答案。可能会包含表述模糊的问题，此时会同时给出确认信息。以上图为例，可能给出 $(Q_1, Q_2, Q_3, Q_4, R_3)$ 求 S_4。

[1]官方网站：https://yale-lily.github.io/cosql

2. **natural language response generation**：以 SQL 语句和返回结果为基础生成自然语言回答。
3. **user dialogue act prediction**：对每一个用户的提问，判断属于以下哪个 DB user 标签。

Groups	Dialog acts
DB user	`inform_sql`, `infer_sql`, `ambiguous`, `affirm`, `negate`, `not_related`, `cannot_understand`, `cannot_answer`, `greeting`, `goodbye`, `thank_you`
DB expert	`conform_sql`, `clarify`, `reject`, `request_more`, `greeting`, `sorry`, `welcome`, `goodbye`

我们组主要关注任务一（**DST**），原因如下：

1. 第一个任务对标传统的 Text2SQL 任务，并在难度上做了拔高，适合我们挑战。
   - **Spider → CoSQL → DST** 的难度递增。Spider 在以往的数据集的基础上，提出了 Cross-Domain 的要求。而 SparC 和 DST 进一步要求解决多轮对话的场景。
   - 那 **DST** 与 **SParC** 有什么区别呢？多了更多长句和嵌套句。作者在论文中提到，SparC 数据集中代表性的做法 CD-Seq2Seq 和 SyntaxSQL-con 在 DST 任务下正确率均下降，说明 CoSQL 比 SparC 任务更琐碎更难。
2. 第二个任务与 Test2SQL 的关系不大，倒是更接近翻译任务。
3. 第三个任务其实就是做一个 11− 分类器。
   - 部分标签很容易预测，baseline 算法已经有 60% ~ 80% 的准确率。
   - 关键标签的预测还是要靠理解 Text 并转化为 SQL，而 CoSQL 的难度摆在那里，准确率提高到上面应该会有个瓶颈。
   - 由以上两条，我们认为该任务改进空间不大，难以大幅提高准确率。

DST 任务主要有两个**指标**（这也是我们在接下来测试时要用到的指标）：

- **Question Match**：把一个 dialog 里不同提问视为不同的 Question，对 SQL 做 Exact Match。
- **Interaction Match**：一个 dialog 里的全部 Question 正确才算正确。

由于 CoSQL 的测试集不公开，我们将在验证集上观察和比较实验结果。

2 测试算法简介

因为篇幅有限，这里只对用到的两个算法做简单的介绍。详细介绍可以见我们整理的网站。

2.1 CDSeq-to-Seq

CDSeq-to-Seq 模型在最简单的 Seq-to-Seq 的模型的基础上加入了对话中的历史交互信息 h_i^I，在 **Hidden state** 的与 **Attention vector** 的预测中，结合了上一轮对话历史交互信息的 h_i^I。因此在本轮 token 概率的预测中，利用 Hidden state 与 Attention vector，可以得到结合了在历史交互信息中的 token 预测分布。

但是只有这些信息可能是不够的，因为历史信息在多次数的迭代后很难保留下来。所以为了获得更多的对结果有影响的历史信息并且提高预测效率，我们可以将之前对话的一些**预测片段 (segment)** 加入到这次的预测中，在此基础上进行本轮对话的预测，这样做大大加快了后几轮对话的预测速度，相对提高了它们的准确率。

2.2 Edit-SQL

EditSQL 是耶鲁大学在 EMNLP2019 提出的基于多轮对话的 Text2SQL 解决方案。它最初是针对 SParC 任务设计的[2]。它目前是 SParC 和 CoSQL 的 SOTA，在两个数据集的 Question Match 上均达到了 **40+%** 的成绩。本次实验我们主要基于 EditSQL 进行一些修改和调优。

EditSQL 并没有 Fancy 的 Backbone，而是由许多个基于 Attention 的 LSTM 搭成。特点如下：

- **本质还是 Encoder-Decoder 模型**。EditSQL 在编码-解码的基础上加入了大量的 Attention 和上下文联系，包括 Utterance-Table Encoder，Interaction Encoder，Table-aware Decoder 等。
- **对数据进行了一些有用的微调**。移除了 SQL 语句里的 `from` 关键词，并在每一个列名前都附上它所属的表。这样不仅让结构更加清晰，还为大量的上下文 Attention 操作提供依据。
- **提出了适配多轮对话的 edit 机制**。作者通过调查数据发现，Ground Truth 的 SQL 回答里，超过半数的 token 都在上一个 SQL 回答里出现过。所以在解码新的 SQL 回答时，我们可以先构建一个概率来判断当前 token 的属性是 **Copy** 还是 **Insert**。如果是 Copy 的话，可以套一层和前一个回答所有 token 相连的全连接网络来进一步确认 token。
- **使用 Bert 能大幅提高准确率**。由于套用了大量 LSTM 结构，如果我们用 BERT 词向量去表示的话，LSTM 和 Attention 的表达能力将进一步增强。

3 测试和分析

3.1 Baseline: CDSeq2Seq

首先我们先测试 CDSeq2Seq 在 CoSQL 数据集上的正确率，作为本次实验的 Baseline。下图展示了它在在 Tesla K20c 显卡上训练一天后的结果（显存占用大概 1 G）。

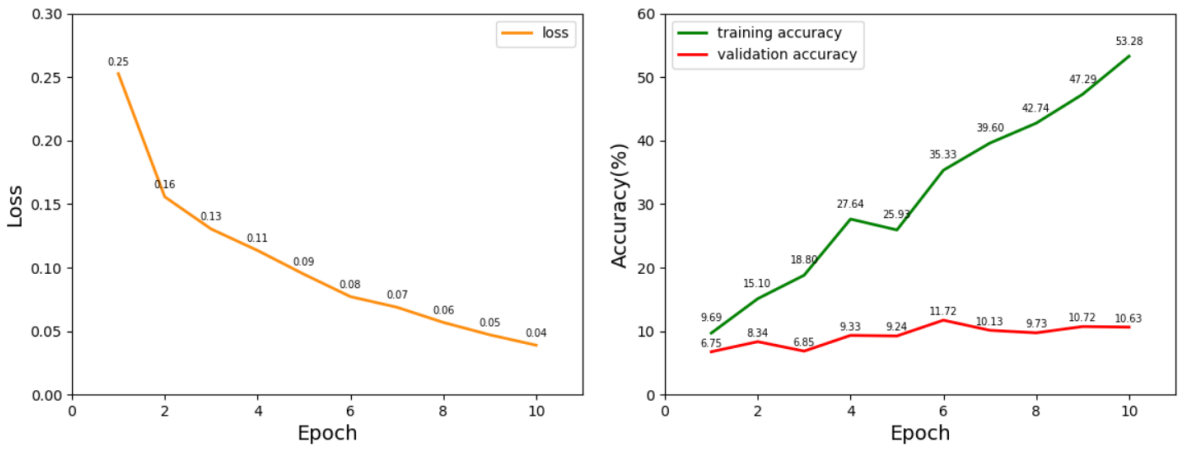

图 2: CDSeq2Seq 的训练 Loss 和 Question Match 正确率随着 Epoch 的折线图

采用官方脚本测试后，Question Match 准确率 12.6%, Interaction Match 准确率 2.4%。

我们会发现，CDSeq2Seq 的有效训练轮数是 5，继续训练会导致**严重的过拟合**。

[2]CoSQL 也是发表在 EMNLP2019 上，所以论文并没有测在 CoSQL 上的正确率

3.2 全版本的 EditSQL 以及显存问题

即使我们把全版本的 EditSQL 部署在有 **Quadro P5000 16G** 显卡的服务器上，并把 *batch_size* 设为 1，依然会遇到 **CUDA Out of Memory** 的情况，使训练无法进行下去。

修改训练参数并不能有效降低 GPU 资源，于是 5 月 11 日我们在作者的 Github Repository 上提出了如下问题，可惜至今还未得到回复（从留言可以看出，其他人也遇到了这个困扰）。

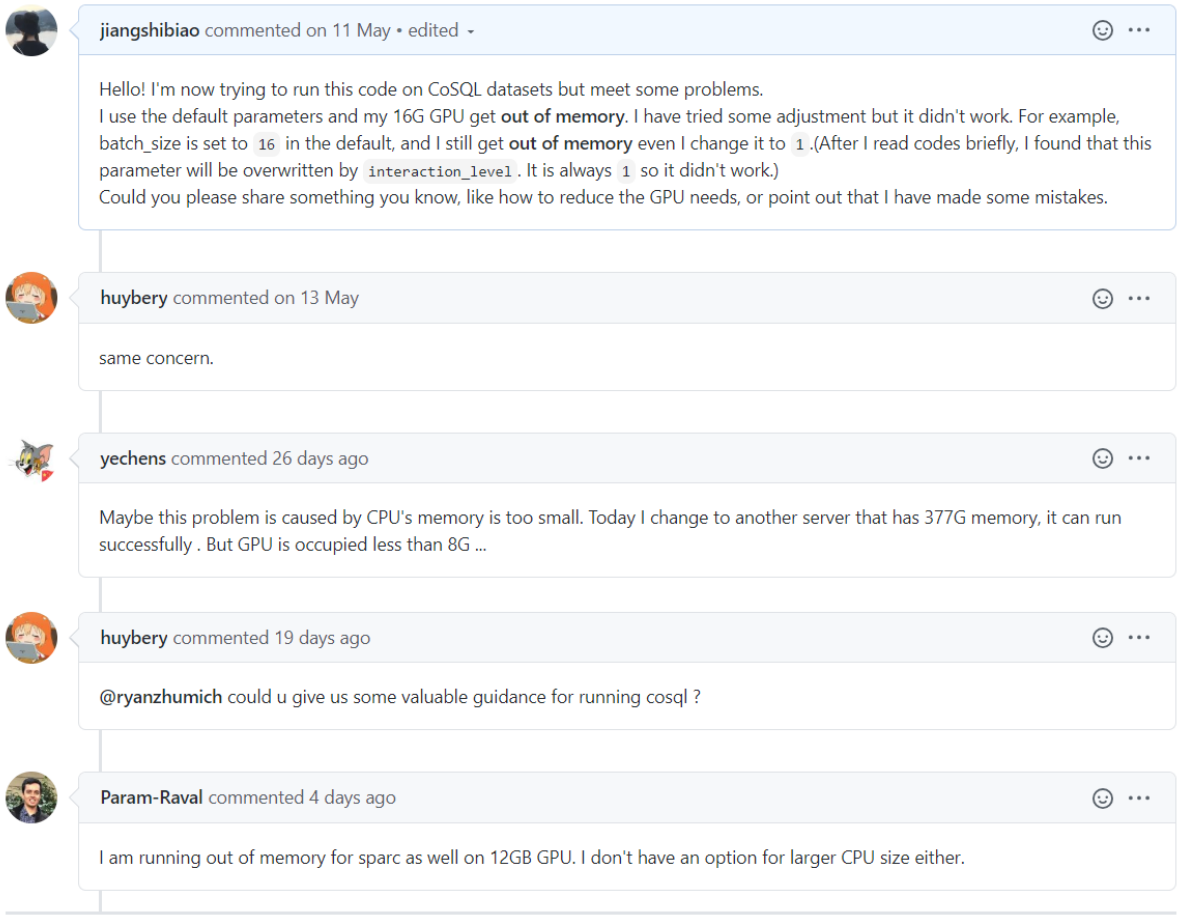

图 3: 我们在作者的 Github Repository 上提的 Issue

3.3 解决方法: CPU 训练 & 去掉 Bert

一个最直观的解决思路是：**改用 CPU 去训练**。

我们对源代码做了一些修改，使其能正确地在 CPU 环境下部署并运行。

为了缓解 CPU 训练速度慢的情况，我们在 Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz 的服务器上开十核并行（CPU 占用率稳定在 1000% 左右）。尽管如此，完全跑完一次 EditSQL+BERT 花费了两周的时间，共计 38 个 Epoch。下图展示了前 25 个 Epoch 的情况。

采用官方脚本测试后，Question Match 准确率 **43.5%**, Interaction Match 准确率 **13.3%**。

分析该折线图我们会发现，过拟合现象**依然十分严重**。Loss 和 Training Accuracy 随着 Epoch 的增大不断减少/增大，但是 validation accuracy 很快就趋于稳定。不过最优点的位置倒是比较靠后（**Epoch=28** 取到模型最优解），这说明 EditSQL 对 Epoch 的需求比 CDSeq2Seq 要高。

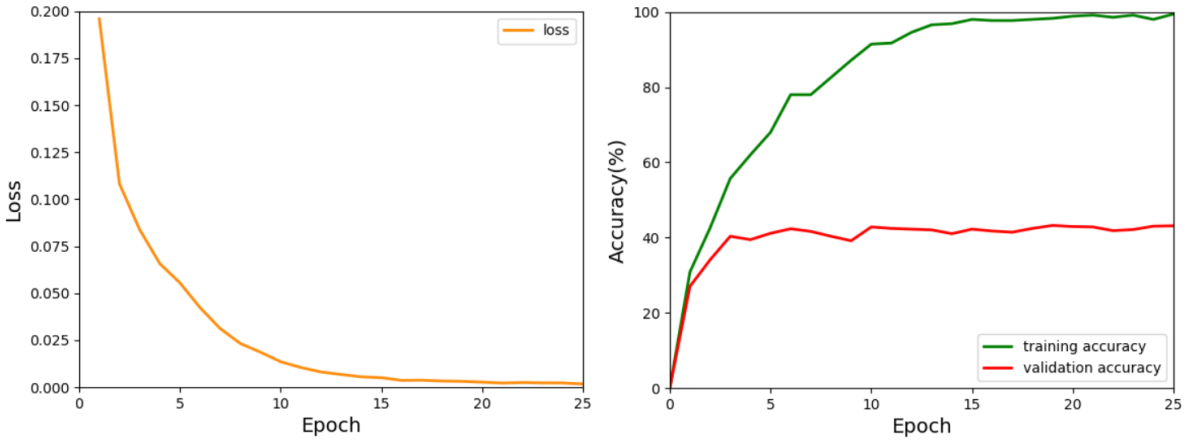

图 4: 用 CPU 训练 EditSQL+Bert 两周后的结果（取前 25 个 Epoch）

但是 CPU 策略并不是万能的，过慢的训练速度让我们很难进一步优化。我们对 EditSQL 的代码结构做了详细的分析，发现主要显存占用是 BERT。即使我们用最小的 **glove.840B.300d.txt** 表示词向量，依然有 30522×768 的大小。此外，EditSQL 会利用 BERT 做大量 fune tuning，所以也不好对其结构或者对 BERT 的调用做修改。

最后我们决定**在去掉 BERT 的数据集上实验和调参**。逻辑上来说，BERT 只是辅助 EditSQL 的一个工具，同一组参数在“是否保留 BERT”上的相对结果应该是一定的。

3.4 数据增强

注意到，EditSQL 已经最大限度地使用 LSTM 和 Attention 结构了，所以我们很难通过继续加网络来优化；此外，LSTM 和 Attention 本身已经很成熟了，魔改里面的结构显然不现实。

受到 SyntaxSQLNet 论文的启发，我们想对 EditSQL 做**数据增强**。有两个主要依据：

1. EditSQL **模型复杂，参数众多**，但是用到的训练数据只有 2157 组。
2. EditSQL **过拟合现象十分严重**，增加了数据显然后能有效地缓解。

在 SyntaxSQLNet 论文里，作者设置了很宽松的数据迁移条件，最终获得了巨量的生成数据（10000+ 条）。但是我们不能一味地套这个做法，因为 CoSQL 数据集语句复杂，又有多轮对话，随意的数据生成会反而会影响最终结果。

```
Try to transfer from [ship_1] to [railway]
let me know the name of the youngest captain
['select', 'captain.name', 'order_by', captain.age, 'limit_value']
let me know the name of the youngest manager
['select', 'manager.name', 'order_by', manager.age, 'limit_value']
How young is he ?
['select', 'captain.age', 'order_by', 'captain.age', 'limit_value']
How young is he ?
['select', 'manager.age', 'order_by', 'manager.age', 'limit_value']
Now who is the oldest captain ?
['select', 'captain.name', 'order_by', 'captain.age', 'desc', 'limit_value']
Now who is the oldest manager ?
['select', 'manager.name', 'order_by', 'manager.age', 'desc', 'limit_value']
{'captain': 'manager'}
```

图 5: 数据增强后的一则简单的例子 by 蒋仕彪

最终我们**讨论出了一种谨慎但是高效的数据增强做法**。经过 EditSQL 本身的数据预处理后，`from` 语句会被去掉，所有的列名 B 会变成 $A.B$ 的形式，其中 A 是 B 对应的表名。我们试图建立一个从数据库 C 到数据库 D 的映射，使得对于所有的 $C.B$，都存在对应的 $D.B$（保证 C 的所有列名都在 D 里出现）。然后我们枚举所有训练集，对于每一个 Intersection 的 Ground Truth，尽可能地用它找到可映射的数据库，并增加这则用了新数据库的 Intersection。下面举了一些细节：

- 找到了映射后，我们要把 SQL 的 GT 里的每一个 $C.B$ 都换成 $D.B$。
- 除了修改 SQL 的回答，我们还需要同时**修改询问**。如果询问语句里的某个 Token 是 C，我们要将其变成 D[3]。举个例子，假设我们做一个从 `Teacher` 到 `Student` 的映射，不仅需要把 SQL 回答从 `Teacher.name` 改成 `Student.name`，还要把询问里的 *List all the names of students* 改成 *List all the names of teachers.*

```
Try to transfer from [chinook_1] to [cinema]
List album titles for albums containing Reggae genre tracks . | Did you mean both contain 'Reggae ' and 'Rock ' genre tracks ? | yes
['select', 'album.title', 'where', 'genre.name', '=', 'value', 'intersect', 'select', 'album.title', 'where', 'genre.name', '=', 'value']
List film titles for film containing Reggae cinema tracks . | Did you mean both contain 'Reggae ' and 'Rock ' cinema tracks ? | yes
['select', 'film.title', 'where', 'cinema.name', '=', 'value', 'intersect', 'select', 'film.title', 'where', 'cinema.name', '=', 'value']
List album titles for albums containing Rock genre tracks
['select', 'album.title', 'where', 'genre.name', '=', 'value', 'intersect', 'select', 'album.title', 'where', 'genre.name', '=', 'value']
List film titles for film containing Rock cinema tracks
['select', 'film.title', 'where', 'cinema.name', '=', 'value', 'intersect', 'select', 'film.title', 'where', 'cinema.name', '=', 'value']
List album titles for albums containing Metal genre tracks
['select', 'distinct', 'album.title', 'where', 'genre.name', '=', 'value']
List film titles for film containing Metal cinema tracks
['select', 'distinct', 'film.title', 'where', 'cinema.name', '=', 'value']
How many albums contain 'Metal ' genre tracks ?
['select', 'count', '(', 'distinct', 'album.title', ')', 'where', 'genre.name', '=', 'value']
How many film contain 'Metal ' cinema tracks ?
['select', 'count', '(', 'distinct', 'film.title', ')', 'where', 'cinema.name', '=', 'value']
```

图 6: 数据增强后的一则复杂的例子 by 蒋仕彪

为了增大成功的概率，我们各自按以上思路去实现数据增强，并独立在服务器上测试。由于细节众多，我们经过很多轮数据的核验、提纯才统一了格式。最终蒋仕彪获得了 2157+1119 = 3276 组数据，李广林获得了 2157 + 3300 = 5457 组数据。为了能更快地获得实验结果，我们重新在**天河二号**上部署了训练环境并训练。部分测试结果展示在下面的表格里，每一组大概训练一天，GPU 消耗 2 G 左右。容易发现，蒋仕彪写的数据增强有良好的效果，在 Question Accuracy 和 Interactionn Accuracy 中都取得了**最优秀的准确率**，这和我们的预期是一致的

表 1: 天河服务器上使用不同数据的运行结果汇总

Algorithm Description	Data Size	Length Limitation	Question Accuracy	Interaction Accuracy
原方法（默认参数）[4]	2157	200	19.2%	4.7%
原方法（换长度限制）	2157	250	18.6%	4.7%
数据增强 By JSB	3276	250	19.0%	4.1%
数据增强 By JSB	3276	200	**20.1%**	**5.5%**
数据增强 By LGL	5457	200	18.2%	3.8%
数据增强 By LGL	5457	150	17.3%	3.8%

[3]具体实现的时候更加复杂，询问里的 Token 可能是经过变换的（比如从单数变成复数）。我们会计算两两字符串的交集，如果交集 > 90% 就认为是同一组单词。

[4]为了保证稳定性，该方法的结果取的是三次实验（一次在原来的服务器上跑，两次在天河服务器上跑）的平均值。