Text-to-SQLにおけるSQL構文に着目したデータ拡張手法

谷口　泰史[†]　中山　光樹[†]　久保　隆宏[†]　鈴木　潤[‡]
TIS株式会社[†]，東北大学 大学院情報科学研究科[‡]
[†]{taniguchi.yasufumi，nakayama.hiroki，kubo.takahiro}@tis.co.jp，
[‡]jun.suzuki@ecei.tohoku.ac.jp

1　はじめに

意味解析 (Semantic parsing) は，計算機による文章の解析/理解に向けた自然言語処理研究の重要な研究領域の一つである [10, 5]．本稿では，意味解析タスクの中でも Text-to-SQL タスク [1] について議論する．ここで，Text-to-SQL タスクの例を図 1 に示す．この例からもわかるように，Text-to-SQL は，自然言語により記述された質問文を入力として受け取り，それと同じ意味になる SQL（論理形式）に変換するタスクである．現在，Text-to-SQL タスクでは，他の自然言語処理研究同様，深層ニューラルネットワーク（以下 DNN と略記）を用いた方法論が広く用いられている [4]．しかし例えば，Yu ら [9] が提案した Spider と呼ばれる最新の Text-to-SQL 用データセットのリーダーボード[1]で現在一位を獲得しているシステムは，最新の DNN 技術を駆使した方法論ではあるが，それでも F_1 値で 61.9 程度である．このことから，実用に耐えうる性能に到達するまでにはまだ多くの改善が必要な状況と言える．

Text-to-SQL タスクの性能の大幅向上に向けた取り組みは様々な方向性が考えられるが，本研究では，現状の Text-to-SQL タスクの大きな課題と考えられる，DNN の学習に利用できるデータ量に着目する．Spider などの Text-to-SQL タスクのデータは，基本的には SQL の専門的な知識を持った人によるアノテーションを必要とする．また，web 上などに質問文と SQL 文の対応が取れたデータが自然発生的に蓄積されることは極めて稀である．そのため，自然言語処理分野で成功を収めつつ，かつ，Text-to-SQL タスクに類似点が多い機械翻訳と比較して，利用可能なデータ量が極めて少ないという状況を容易には改善できない．

一般論として，近年広く用いられている DNN 技術は，データ量が最終的なモデルの性能に大きく影響することが知られている．そこで本研究では，Text-to-SQL タスクに向けた DNN の性能を大幅に向上させるために学習用データの自動拡張手法を考案する．その第一歩目として，本稿では，Spider を用いた実験において，データ拡張の有用性を検証する．また，SQL 文の構造に着目したデータ拡張法を検討し，一部性能向上に寄与することを確かめる．

[1]https://yale-lily.github.io/spider

自然言語	少なくとも 3 つの車メーカーがある ヨーロッパ国はどこですか.
論理形式 (SQL)	SELECT T1.country_name FROM countries AS T1 JOIN continents AS T2 ON T1.continent = T2.cont_id JOIN car_makers AS T3 ON T1.country_id = T3.country WHERE T2.continent = 'Europe' GROUP BY T1.country_name HAVING COUNT(*) >= 3;

図 1: Text-to-SQL タスクの例．Yu ら [9] が提案した Spider と呼ばれるデータセットに採用されているサンプルから抜粋．

2　関連研究

Text-to-SQL データセット． Text-to-SQL で用いられるデータセットとして，Zhong ら [11] の提案した WikiSQL や，Finegan-Dollak ら [2] の提案した Advising などが挙げられる．Zhong ら [11] の WikiSQL は 80,654 件からなるデータセットであり，Text-to-SQL では最大規模のデータセットである．しかし，Finegan-Dollak ら [2] が指摘するように，未知の質問文に対して正しく SQL 文を生成させることを目的としたデータセットであるため，学習データとテストデータで全く同じような SQL 文が含まれることがあった．Finegan-Dollak ら [2] は上述の課題に対応するため，学習データとテストデータで全く同じ SQL 文が含まれないデータセットとして，Advising を提案している．Advising は WikiSQL のように，学習データとテストデータの被りは存在しないが，採用さている SQL 文に含まれる SQL 句の種類が少なく，ドメインも 1 つに絞られていた．Yu ら [9] の Spider は，これらの課題を解決したデータセットである．Spider は上記のこれまで提案されてきたデータセットの中で，高難度の Text-to-SQL のデータセットの一つである．よって，本研究では，Spider のデータを対象として議論を行う．

Text-to-SQL タスクでのデータ拡張． Text-to-SQL のためのデータ拡張に関しては，Daya ら [3] が注意機構付き encoder-decoder にコピー機構を追加したモデルを用いた研究を行っている．しかし Daya ら [3] の手法は WikiSQL に対してのみ行われており，Spider[9] での検証は行われていない．Yu ら [8] は，Spider に対

してテンプレートを用いたデータ拡張手法を提案し，性能が向上することを報告している．しかし，Yu ら [9] の方法論は，テンプレートを用いて，Spider 以外のデータベースから，Spider に出現するような質問文と SQL 文を自動生成する方法論であり，データセットに出現しない SQL 句の組み合わせ方を持つデータの生成は行っていない．

3 Text-to-SQL ベースモデル

Text-to-SQL の研究においては，現状，デファクトスタンダードと呼べるベースラインモデルは確立されていない．本研究では，現状トップレベルの性能を示しており，かつ，実験用のコードが公開されているという理由で，IRNet[4] をベースラインのモデルとして利用する．IRNet は Text-to-SQL を 3 段階に分けて行うモデルである．1 段階目では，データベーススキーマを用いて，質問文に含まれるテーブル名やカラム名の予測を行う．続いて，質問文と予測した情報から，文法ベースニューラルモデルを用いて，中間表現である SemQL 文の生成を行う．最後に，SemQL 文を SQL 文に変換する．

以下に IRNet の各処理に関して簡単に説明する．

3.1 スキーマリンキング

スキーマリンキングでは，質問文の単語 n-gram を *column*，*table*，*value*，*none* の 4 つの $type_1$ に分類する．*column* はカラム名，*table* はテーブル名とする．*value* は *column* の実際の値である．前述 3 つの $type_1$ に含まれない場合は *none* とする．さらにカラムを Exact Match，Partial Match，Value Exact Match，Value Partial Match の 4 つの $type_2$ 分類する．

質問文の単語 n-gram の分類では，単語 n-gram を 1-gram から 6-gram まで列挙し，6-gram から順に分類を行う．カラム名もしくはテーブル名と完全一致もしくは部分一致した場合，その n-gram を *column* もしくは *table* として分類する．n-gram がカラム名とテーブル名の両方にマッチした場合は，*column* を優先する．n-gram の 1 単語目と最後の単語が両方ともシングルクオートの場合 *value* として分類する． n-gram が *column*，*table*，*value* のうち何れかとして分類された場合，列挙した n-gram 列から，分類された n-gram と被りがあるものをすべて取り除く．最終的に *column*，*table*, *value* の何れかに分類された n-gram と 3 つの何れにも分類されなかった 1-gram となる．分類されたかった 1-gram はすべて *none* に分類される．以上の操作で，質問文の単語 n-gram の分類が完了する．

カラム名の分類では質問文中に，カラム名が出現した場合 Exact Match，カラム名の部分文字列が出現した場合 Partial Match とする．さらに，質問文の単語が ConceptNet[2]において，カラム名と '*is a type of*' もしくは，'*related terms*' の関係を持つ場合 Value Exact Match，カラム名の部分文字列と上記の関係を持つ場合 Value Partial Match とする．

[2]http://conceptnet.io/

3.2 SemQL 文

Guo ら [4] が定義した，質問文と SQL 文の中間表現である SemQL 文について説明する．SemQL 文は文脈自由文法により定義され，木構造を持っている．SemQL 文では，SQL 文の各 SQL 句に対応するノードが定義されている．各 SQL 句の詳細は，対応するノードの子孫ノードとして定義されていく．ただし，SemQL 文のノードは，複数の SQL 句に対応するノードも存在している．例えば，`ORDERBY` 句と `LIMIT` 句が同時に使われている場合には，*Superlative* というノードが対応する．SemQL 文は SQL 文の自己結合を表現できないなどの制約があるため，SemSQL 文から完全な SQL 文は復元できない．

3.3 モデル

IRNet[4] で利用されるモデルは，質問文のエンコーダ，スキーマのエンコーダ，そして SemQL 文を生成するデコーダの 3 つからなる．各コンポーネントを順に説明する．

質問文のエンコーダ.
3.1 節で述べた処理後の質問文 $x = [(x_1, \tau_1), \cdots, (x_L, \tau_L)]$ を双方向 LSTM に入力し，ベクトル表現 $\boldsymbol{H}_x$ を出力する．ここで x_i は質問文の i 番目の n-gram，τ_i は 3.1 の処理で割り当てた $type_1$ の one-hot 表現とする．双方向 LSTM に入力する際は単語埋め込みベクトルによってベクトル化した．

スキーマのエンコーダ.
データベーススキーマを $s = (c, t)$ とする．ただし，$c = \{(c_1, \phi_1), \cdots, (c_n, \phi_n)\}$ とし，c_i はカラム名，ϕ_i は 3.1 の $type_2$ の one-hot 表現とする．$t = t_1, \cdots, t_m$ はテーブル名の集合である．スキーマのエンコーダは s を入力にとり，カラムとテーブルそれぞれをベクトル表現 $\boldsymbol{E}_c$, $\boldsymbol{E}_t$ に変換する．c_i と t_j はそれぞれ質問文のエンコーダと同じ単語埋め込みベクトルを用いてベクトル化し，ϕ_i は $type_2$ 埋め込みベクトルを使って $\boldsymbol{\tau}_i$ とする．カラムについては以下のように変換を行う:

$$
\begin{aligned}
g_k^i &= \frac{\left(\hat{c}_c^i\right)^\top e_x^k}{\|\hat{e}_c^i\| \, \|e_x^k\|}, \\
c_c^i &= \sum_{k=1}^{L} g_k^i e_x^k, \\
e_c^i &= \hat{e}_c^i + c_c^i + \varphi_i.
\end{aligned}
\tag{1}
$$

SemQL 文生成用デコーダ. 質問文およびスキーマのエンコーダの二つの出力を受け取り，SemQL の生成を行う．生成には，Yin と Neubig[7] が提案した文法ベースデコーダを用いる．

3.4 予測手順

IRNet による質問文からの SQL 文生成について説明する．質問文に対して，3.1 を用いて，カラム名やテーブル名などの情報を付与する．SQL 文は，3.2 により事前に SemQL 文に変換を行う．質問文，SQL 文をそれぞれ変換後，3.3 のモデルの学習を行う．

学習したモデルから SemQL 文を得たのち，対応する SQL 文の推測を行うことで，SQL 文を生成する．SemQL 文から SQL 文への推測は，事前に定義されたルールによって行われる．

4 提案手法：データ拡張

本研究では，Text-to-SQL に向けたデータ拡張がニューラルネットワークを用いた方法論に対して有効であることの検証を行う．データ拡張手法は低資源環境 (low-resource setting) の機械翻訳でその効果が確認されている [6]．Text-to-SQL も機械翻訳と同様の系列変換タスクであり，データ拡張手法が有効であると考えられる．

本研究で用いたデータ拡張手法の詳細について述べる．データ拡張のためには SQL 文と質問文のペアを用意する必要がある．そのための提案法は，SQL 文の生成と質問文の生成の 2 ステップから構成される．SQL 文の生成では SQL の構造を利用し，元の SQL 文を変更することで新たな SQL 文を生成する．ルールの構築が容易な `ORDERBY` に着目した手法を提案する．質問文の生成では SQL 文の変更と整合性が取れるように元の質問文を変更して生成する．

SQL 文の生成. SQL 文の `ORDERBY` 句において，`DESC` とするとデータは降順に得られ，`ASC` とすると昇順に得られることになる．次のような SQL 文を考える:

```
SELECT id
FROM trip
ORDER BY duration DESC
LIMIT 1;
```

`ORDERBY` 句の `DESC` を `ASC` に置き換えることで，取得されるデータの順序を入れ替えることができる．この SQL 文は `LIMIT` 句が 1 になっているので，`duration` の値が最小のデータを取得しているが，`DESC` を `ASC` に置き換えると，`duration` の値が最大のデータを取得することになる．

質問文の生成. 上述の SQL 文に対応する質問文は以下となる:

*What is the id of the trip that has the **shortest** duration?*

`ORDERBY` 句の `DESC` を `ASC` を変えたときは，以下のように変化すると考えることができる:

*What is the id of the trip that has the **longest** duration?*

そこで本研究では，`ORDERBY` 句を含む SQL 文が降順なら昇順に，昇順なら降順に変換した．さらに SQL 文に対応する質問文が，比較級または最上級の形容詞を含む場合に，形容詞を反対語に変換した．上記 2 点を満たすデータを追加してモデルの学習を行った．

また，Spider に含まれる `ORDERBY` 句を含む SQL 文の `ORDERBY` 句部分を，`ORDERBY` 句を含まない SQL 文に付与する拡張も行った．対応する質問文は，既存の質問文に *in descending order of* (カラム名) など予め定義した文節を付与することで生成を行った．

表 1: 学習データと開発データにおける難易度ごとのデータ件数

難易度	学習データ	開発データ
easy	1,989	252
medium	3,875	469
hard	1,467	153
extra hard	1,328	160
all (合計)	8,659	1,034

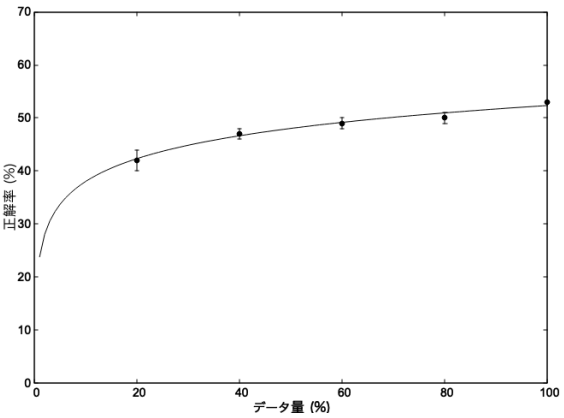

図 2: 学習データ量と性能の関係

5 実験

5.1 実験設定

実験には，Yu ら [9] 提案した Spider データセット 10,181 件を利用した．Yu ら [9] の定義に従い，8,659 件を学習データ，1,034 件を開発データ，2,147 件をテストデータとした．ただしテストデータは公開されていないので，分析は開発データを用いて行った．データ拡張の効果を擬似的に検証するため，学習データのうち 20%, 40%, 60%, 80%をランダムサンプルし，それらの少量データによる性能評価も行った．シード値を変えて，5 回データをサンプルし，結果には各結果の平均値を用いた．

ベースラインモデルは PyTorch[3]による実装を利用した．単語埋め込みベクトルは GloVe[4]で初期化を行い，学習中は固定した．パラメータ最適化には Adam を利用した．使用したパラメータの詳細は 4 に示した．

[3]https://github.com/microsoft/IRNet
[4]https://nlp.stanford.edu/projects/glove/

表 2: SQL 句ごとの SQL 句単位評価と完全一致評価

難易度	完全一致	SELECT	SELECT (no AGG)	WHERE	WHERE (no OP)	GROUP (no Having)	GROUP	ORDER	AND/OR	IUEN	keywords
easy	0.696	0.868	0.900	0.721	0.766	0.773	0.727	0.735	0.988	-	0.868
medium	0.548	0.800	0.823	0.669	0.674	0.725	0.679	0.599	0.976	-	0.877
hard	0.466	0.925	0.943	0.508	0.608	0.759	0.759	0.821	0.973	0.357	0.754
extra hard	0.306	0.748	0.754	0.363	0.484	0.671	0.671	0.755	0.938	0.257	0.714
all	0.530	0.829	0.850	0.591	0.646	0.718	0.692	0.719	0.972	0.291	0.819

表 3: ORDERBY 句を拡張した際のモデルの性能 (F_1 値)

	完全一致	SELECT	SELECT (no AGG)	WHERE	WHERE (no OP)	GROUP (no Having)	GROUP	ORDER	AND/OR	IUEN	keywords
IRNet	0.530	0.829	0.850	0.591	0.646	0.718	0.692	0.719	0.972	0.291	0.819
+データ拡張	0.509	0.815	0.836	0.560	0.598	0.735	0.708	0.726	0.969	0.311	0.811

表 4: モデルの各層のベクトル次元数

単語埋め込みベクトル	300
$type_2$ 埋め込みベクトル	300
エンコーダ・デコーダの隠れ層のベクトル	100

記載のないパラメータについては Guo ら [4] の論文と同じパラメータを使用した．

性能評価には，従来用いられている SQL 全体の完全一致による正解率を用いた．また，モデルの詳細な挙動を分析する目的で，SQL 句ごとの評価を行った．ここでは以下「SQL 句単位評価」とよぶ．SQL 句単位評価では F 値により評価を行った．Yu ら [9] の提供する評価スクリプト[5]を用いて評価を行った．さらに Spider[9] では，各 SQL 文ごとに難易度が設定されているので，各難易度ごとの評価も行った．表 1 に，学習データと開発データにおける難易度の統計を示す．

5.2 実験結果

データ拡張の有用性の検証． 学習データ量を変化させたときのモデルの性能の平均値を図 2 に示した．図 2 から学習データ量を増やすことで，性能は向上し，データ量を増やすことで，さらに性能向上を見込めることが確認できた．

提案手法の検証． 全学習データを用いて学習したときの SQL 句ごとの SQL 句単位評価結果を表 2 示した．表 1 と表 2 から，モデルの性能に最も寄与するのは medium 難易度であり，medium 難易度で完全一致率が最も低いのは，`ORDERBY` 句であることがわかった．簡易的な実験として，`ORDERBY` 句を含むデータを拡張し，実験を行った．拡張したデータを用いたときの，結果を表 3 に示す．他の SQL 句の性能は劣化したものの，`ORDERBY` 句の性能は向上した．本稿では，未知の SQL 文とそれに対応する質問文が単純なルールベースにより獲得できる `ORDERBY` 句のみの拡張を行った．他の SQL 句においても未知の SQL 文は容易に得ることができると考えられる．よって，質問文を獲得できれば，データ拡張により他の SQL 句についても性能向上が期待できる．

[5]https://github.com/taoyds/spider

6 おわりに

本研究では，意味解析タスクの 1 つである Text-to-SQL のデータ拡張手法を提案した．学習に利用するデータ量を変化されることで，データ拡張が性能向上に寄与することを確認した．また SQL 文の構造に着目したデータ拡張手法の提案を行った．全体としてのモデルの性能は向上しなかったが，句単位では性能向上することを確認した．

参考文献

[1] Ion Androutsopoulos, Graeme D. Ritchie, and Peter Thanisch. Natural language interfaces to databases - an introduction. *CoRR*, Vol. cmp-lg/9503016, , 1995.

[2] Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. Improving text-to-SQL evaluation methodology. In *Proc. of ACL'18*, pp. 351–360, 2018.

[3] Daya Guo, Yibo Sun, Duyu Tang, Nan Duan, Jian Yin, Hong Chi, James Cao, Peng Chen, and Ming Zhou. Question generation from SQL queries improves neural semantic parsing. In *Proc. of EMNLP'18*, pp. 1597–1607, 2018.

[4] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Towards complex text-to-SQL in cross-domain database with intermediate representation. In *Proc. of ACL'19*, pp. 4524–4535, 2019.

[5] Percy Liang, Michael I. Jordan, and Dan Klein. Learning dependency-based compositional semantics. *Computational Linguistics*, Vol. 39, No. 2, pp. 389–446, 2013.

[6] Mengzhou Xia, Xiang Kong, Antonios Anastasopoulos, and Graham Neubig. Generalized data augmentation for low-resource translation. In *Proc. of ACL'19*, pp. 5786–5796, 2019.

[7] Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. In *Proc. of ACL'17*, pp. 440–450, 2017.

[8] Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. SyntaxSQLNet: Syntax tree networks for complex and cross-domain text-to-SQL task. In *Proc. of EMNLP'18*, pp. 1653–1663, 2018.

[9] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proc. of EMNLP'18*, pp. 3911–3921, 2018.

[10] Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proc. of UAI'05*, pp. 658–666, 2005.

[11] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, Vol. abs/1709.00103, , 2017.