Table2Analysis: Modeling and Recommendation of Common Analysis Patterns for Multi-Dimensional Data

Mengyu Zhou,[1] **Tao Wang,**[2*] **Pengxin Ji,**[2*] **Shi Han,**[1] **Dongmei Zhang**[1]
[1]Microsoft Research
[2]Beijing University of Posts and Telecommunications
Beijing, China
{mezho, shihan, dongmeiz}@microsoft.com, {wangt, jpx}@bupt.edu.cn

Abstract

Given a table of multi-dimensional data, what analyses would human create to extract information from it? From scientific exploration to business intelligence (BI), this is a key problem to solve towards automation of knowledge discovery and decision making. In this paper, we propose Table2Analysis to learn commonly conducted analysis patterns (denoted as Common Analysis) from large amount of (table, analysis) pairs, and recommend analyses for any given table even not seen before. Multi-dimensional data as input challenges existing model architectures and training techniques to fulfill the task. Based on deep Q-learning with heuristic search, Table2Analysis does table to sequence generation, with each sequence encoding an analysis. Table2Analysis has 0.78 recall at top-5 and 0.65 recall at top-1 in our evaluation against a large scale spreadsheet corpus on the PivotTable recommendation task.

1 Introduction

For data analysis and information sharing, people create PivotTables, charts (Figure 1), and other various types of visuals using analytics tools such as spreadsheets and other BI software. Non-trivial interactions and skills are often required to operate them, which could be difficult and time consuming.

Opportunities exist for enabling automatic recommendation of data analyses (Milo and Somech 2018; Ding et al. 2019) to mitigate this productivity issue. There are large amount of data analysis artifacts providing both source dataset and result visuals, such as Excel spreadsheets or Power BI reports. Such artifacts embed commonly conducted analysis patterns across users, denoted as **Common Analysis**, including patterns of typical combinations of data semantics (*e.g.*, "sum of sales by quarter" or "average price per month by region") and patterns about data characteristics (*e.g.*, without bucketing, a data-field with continuous float numbers in $[0, 1]$ is rarely used as breakdown dimension). These opportunities together motivate us to devise a technology for learning and recommending Common Analysis patterns.

First, an abstraction is needed to encode the essential analysis components that are invariant to interested visual types. For this purpose in §2 we design an *analysis language* that encodes analysis process into a sequence of action tokens. Each token is either a predefined analysis operation or a reference to a data-field from the given multi-dimensional dataset. (*E.g.*, "sum of sales by quarter" could be represented as the sequence `[ANA]` [*Sales*] `[SEP]` [*Quarter*] `[Sum]`.) The analysis language enables a structured representation of various analyses against which learning will be conducted.

Based on such an analysis language, each table is a set of data-field tokens and each analysis is a sequence of operation/reference tokens. We propose **Table2Analysis** (see §3) as a set-to-sequence framework. At learning stage, it learns from large amount of (table, analysis) pairs. At recommendation stage, it generates top-k sequences of analyses for a given table. The key is language modeling (§2.2) that can be transformed into an action value (Q-value) function that models the next-action score based on the current state of a given input set and already generated prefix. However, following facts challenge the learning of such a Q-value function:

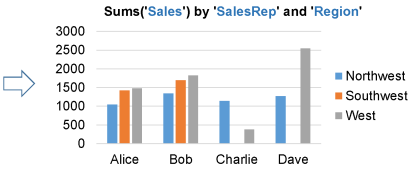
- It is an open-vocabulary problem while the tokens are not discrete nor enumerable. A reference token can refer to any data field of any given table, consisting of a combination of header text and data values. It challenges conventional techniques in typical NLP scenarios.
- The exposure bias (Ranzato et al. 2015) can not be mitigated by directly applying conventional techniques due to different success metric. At inference stage, when a prefix deviates from ground truth distribution, it implies the analysis is already broken. Instead of optimizing next- (and all subsequent) action generation to remedy some BLEU score-like target, the model training should optimize the chance of switching to a more promising search branch.

Therefore, in §3.2 we design an input layer of neural network models to embed semantics of unlimited data-fields from tables. In §3.4 we run multiple agents using the customized beam searching with the model itself to collect samples to bridge the gap between training and inference.

We verify the effectiveness of Table2Analysis on PivotTable recommendation. As shown in Figure 1b, PivotTable (see §4.1) is a widely used type of analysis. We collect over 121,000 tables from 74,000 real world Excel files. For recommendation, we achieve over 65% recall at top 1 and 78% at

*The contributions by Tao and Pengxin have been conducted and completed during their internships at Microsoft Research Asia.

Date	Region	SalesRep	Sales
10/21/13	Northwest	Alice	$154
10/21/13	Northwest	Bob	$205
10/21/13	Southwest	Bob	$895
10/20/13	West	Bob	$620
10/20/13	Northwest	Bob	$484
10/21/13	West	Alice	$1,254
…	…	…	…



(a) Raw Table and a Common Analysis.

Sum of Sales	Column Labels			
Row Labels	Northwest	Southwest	West	Grand Total
Alice	1041	1423	1476	3940
Bob	1340	1694	1821	4855
Charlie	1141		376	1517
Dave	1269		2542	3811
Grand Total	**4791**	**3117**	**6215**	**14123**

(b) Corresponding PivotTable.

Figure 1: Example of a Commonly Performed Analysis.

top 5 using ∼0.5 seconds for each table. Given the generality of our problem formulation, similar effectiveness on the PivotTable task can be expected against other analysis types.

This paper has the following major contributions:

- The novel Common Analysis task is proposed. It combines structured prediction and recommendation in data analytics domain and requires learning of common wisdom.
- We design a language-based Table2Analysis framework to learn and generate Common Analyses. To the best of our knowledge, for analysis recommendation, our approach is the first to automatically learn desirable semantic combinations by utilizing unstructured texts from tables.
- We collect a large scale dataset for the PivotTable recommendation task and conduct systematic evaluations to verify the effectiveness of the proposed techniques.

2 Problem

For a multi-dimensional dataset such as a spreadsheet table, its data fields fall into the following two categories.

- Dimension, a categorical attribute that can be used for grouping the records with the same value on this field;
- Measure, a numerical attribute that can be used for measuring a group of records by aggregation[1].

Given a dataset (Figure 1a left), a user typically constructs an analysis (Figure 1a right or the same PivotTable in Figure 1b) step-by-step: The measure "Sales" is first selected as the most cared; then "Sales" is grouped by a dimension "SalesRep" and further by "Region"; finally Sum is selected as the most proper aggregation function.

Among the various types of data analyses against multi-dimensional data, their shared essential analysis semantics can be abstracted as the combination in the following 3-tuple:

⟨dimension(s), measure(s), aggregation function(s)⟩

In this paper, we focus on building machine learning models that can mimic the human paradigm at a lower level and recommend Common Analyses at a higher level. The two levels of recommendation are as follows.

1. (Higher level) Complete analyses: A recommendation engine gives user a ranked list of data analysis candidates, each being a complete analysis as shown in Figure 1a.
2. (Lower level) Next-step analysis action: At each step of user constructing an analysis, a recommendation engine suggests next action candidates.

[1]The most commonly used aggregation functions in Excel PivotTables (see Figure 3) are Sum, Count, Average.

2.1 Analysis Language

In this section, we define the grammar of analysis language which captures the essential components of an analysis and the human process of constructing it.

Definition 2.1 (Source Field)**.** A n-dimensional dataset $\mathcal{D}$ consists of n source fields $\mathcal{F}_\mathcal{D} = (f_1^\mathcal{D}, ..., f_n^\mathcal{D})$. Each source field $f \in \mathcal{F}_\mathcal{D}$ is an attribute with its corresponding values of all data records. (*E.g.*, a column in the table of Figure 1a is a field.) It consists of two parts: 1) Field name, such as the header title describing the table column; 2) Data values, such as the values of records under the header title.

Definition 2.2 (Analysis Language)**.** An analysis token $a \in \mathcal{A}_\mathcal{D}$ corresponds to an action to take in the analysis construction process for dataset $\mathcal{D}$. There are three types of actions in $\mathcal{A}_\mathcal{D}$: 1) select a source field $f \in \mathcal{F}_\mathcal{D}$; 2) change from selecting one component to another, indicated by tokens from $\mathcal{E} = \{\texttt{[ANA]}, \texttt{[SEP]}\}$, where `[ANA]` means starting the analysis by selecting a measure and `[SEP]` means starting to select a series of dimension field(s) which leads to a breakdown hierarchy of groups; 3) apply an aggregation function from $\mathcal{G} = \{\texttt{[Sum]}, \texttt{[Count]}, \texttt{[Average]}, ...\}$ for the preceding measure field and breakdown dimensions. Then in the Backus-Naur form we propose the following analysis language for generating sequence of analysis tokens.

$$
\begin{aligned}
\langle \text{analysis} \rangle &\models \texttt{[ANA]}\langle \text{field} \rangle \texttt{[SEP]}\langle \text{dim(s)} \rangle \langle \text{func} \rangle \\
\langle \text{dim(s)} \rangle &\models \langle \text{field(s)} \rangle \mid \langle \text{field(s)} \rangle \texttt{[SEP]}\langle \text{dim(s)} \rangle \\
\langle \text{field(s)} \rangle &\models \lambda \mid \langle \text{field} \rangle \langle \text{field(s)} \rangle \\
\langle \text{field} \rangle &\models \textit{a field} \in \mathcal{F}_\mathcal{D} \\
\langle \text{func} \rangle &\models \textit{an aggregation function} \in \mathcal{G}
\end{aligned}
$$

The number of breakdown hierarchies (⟨field(s)⟩ separated by `[SEP]` in ⟨dim(s)⟩) varies for different analysis types. In this paper, we focus on basic analyses with only one measure, no filter and only aggregation calculations. But the proposed techniques are general when applied to analyses with multiple measures and other types of calculation operators.

Definition 2.3 (Complete, Partial and Target Sequences)**.** An analysis sequence $s \in \bigcup_{l=1}^{\infty} {\mathcal{A}_\mathcal{D}}^l$ is *complete* if it follows the analysis language grammar and there are no duplicate fields in its dimensions[2]. For a dataset $\mathcal{D}$, we denote the set of all the complete sequences as $\mathcal{C}_\mathcal{D}$, the set of all the prefixes of all the sequences in $\mathcal{C}_\mathcal{D}$ as $\mathcal{S}_\mathcal{D}^+$, and call the sequences in

[2]Although duplicated dimension fields do not violate the analysis grammar, in reality such kind of duplication is mostly useless and leads to redundancy or regression in results.

set $\mathcal{S}_\mathcal{D} = \mathcal{S}_\mathcal{D}^+ \setminus \mathcal{C}_\mathcal{D}$ as *partial* sequences. The user-created or adopted sequences $\mathcal{G}_\mathcal{D} \subseteq \mathcal{C}_\mathcal{D}$ are called *target* sequences. $\mathcal{T}_\mathcal{D}^+$ is the set of all the prefixes of all the targets in $\mathcal{G}_\mathcal{D}$. Then we call $\mathcal{T}_\mathcal{D} = \mathcal{T}_\mathcal{D}^+ \setminus \mathcal{G}_\mathcal{D}$ the set of *partial targets*.

It is worth noting that $|\mathcal{S}_\mathcal{D}^+|$ is exponentially larger than $|\mathcal{F}_\mathcal{D}|$ since the number of legal combinations explodes as the number of fields grows. This makes it very hard to correctly generate the rare targets $\mathcal{G}_\mathcal{D}$ from $\mathcal{S}_\mathcal{D}^+$.

Example 2.1 (Analysis Sequence Example)**.** The analysis sequence for both the chart in Figure 1a and the PivotTable in Figure 1b is: `[ANA]` [*Sales*] `[SEP]` [*SalesRep*] `[SEP]` [*Region*] `[Sum]`, where [*Sales*] is the measure, [*SalesRep*] is the first dimension and [*Region*] is the second dimension (separated in two breakdowns by `[SEP]` since they are the left and top headers (axes) in Figure 1b respectively). `[Sum]` is applied to [*Sales*] as the aggregation (sum of sales) for each sales representative and region.

2.2 Language Modelling Task

Based on the above definitions, the two recommendation tasks (discussed at the beginning of §2) can be transformed into the following language modelling tasks.

1. Complete analyses: Generate a top-k list of synthesized complete analysis sequences $(s_1, ..., s_k)$, ranked by $\mathcal{P}(s \in \mathcal{G}_\mathcal{D} \mid \mathcal{D})$ – the probability s being a Common Analysis for a given dataset $\mathcal{D}$.
2. Next-step action: Given a partial sequence $s \in \mathcal{S}_\mathcal{D}$ and next action $a \in \mathcal{A}_\mathcal{D}$, predict how likely sa is the prefix of a Common Analysis based on $\mathcal{P}(sa \in \mathcal{T}_\mathcal{D}^+ \mid s, \mathcal{D})$.

As long as the structure of interested analyses could be represented in some analysis language with sequential grammar such as the one in §2.1, the above language modelling tasks are applicable. With these tasks in mind, in the next section we propose the Table2Analysis framework to recommend complete analyses by learning an analysis language model from existing (and sometimes growing) analysis corpus.

3 Table2Analysis

The two recommendation tasks mentioned in §2.2 are highly correlated. We transform the complete analyses recommendation task (for a dataset $\mathcal{D}$) into a search (and rank) problem on the state space $\mathcal{S}_\mathcal{D}^+$ based on $\mathcal{P}(s \in \mathcal{G}_\mathcal{D} \mid \mathcal{D})$. The search process will be guided by a heuristic function that approximates the language model $\mathcal{P}(sa \in \mathcal{T}_\mathcal{D}^+ \mid s, \mathcal{D})$. Let's start by the Markov decision process (MDP) for token-by-token sequence generation, and show how its corresponding action-value function represents the analysis language model.

3.1 Token-by-Token Markov Decision Process

Definition 3.1 (Analysis MDP)**.** For an n-dimensional dataset $\mathcal{D}$, we adopt the definitions in §2.1 to describe the next-token analysis sequence generation MDP:

- State space is $\mathcal{S}_\mathcal{D}^+$, which can be viewed as a tree with `[ANA]` as its root node (initial state).
- Action space $\mathcal{A}_\mathcal{D}$: The legal actions for state s are $\mathcal{A}_\mathcal{D}(s) = \{a \mid sa \in \mathcal{S}_\mathcal{D}^+, \forall a \in \mathcal{A}_\mathcal{D}\}$.
- State transition is deterministic. The transition probability from s to s' by taking action $a \in \mathcal{A}_\mathcal{D}(s)$ is:

$$\mathcal{P}_\mathcal{D}(s, a, s') = \begin{cases} 1 & \text{if } s' = sa, \\ 0 & \text{otherwise.} \end{cases}$$

- Reward function $\mathcal{R}_\mathcal{D}$ is designed to reflect if a target sequence is successfully synthesized:

$$\mathcal{R}_\mathcal{D}(s, a, s') = \begin{cases} 1 & \text{if } s' = sa \text{ and } s' \in \mathcal{G}_\mathcal{D}, \\ 0 & \text{otherwise.} \end{cases}$$

- Discount rate $\gamma = 1$ so that the length of an analysis sequence has no impact on its rewards.

According to Bellman optimality equation, one can easily find the optimal action-value function[3]

$$q_*(s, a) = \mathcal{R}_\mathcal{D}(s, a, sa) + \gamma \max_{a' \in \mathcal{A}_\mathcal{D}(sa)} q_*(sa, a')$$
$$= \begin{cases} 1 & \text{if } s' = sa \text{ and } s' \in \mathcal{T}_\mathcal{D}^+, \\ 0 & \text{otherwise.} \end{cases}$$

In other words, $q_*(s, a)$ equals to 1 if and only if sa is **a prefix of a target sequence**. Now it is not difficult to realize the connection between the optimal action-value function and next-token language model: $q_*(s, a)$ is the learning target for $\mathcal{P}(sa \in \mathcal{T}_\mathcal{D}^+ \mid s, \mathcal{D})$.

Two more discussions: First, for simplicity, in the notation $q_*(s, a)$ there is no $\mathcal{D}$, but actually the action-value function should take $\mathcal{D}$ into account. Second, the expected return is determined since we are learning from a corpus with known simple rewards (of target analysis). More complex reward is possible but should be designed *w.r.t.* the strict structure defined by analysis language grammar, which is quite different from flexible NLP ones. To keep our framework directly applicable to languages of other potential analysis types, in this paper we adopt the above universal reward design.

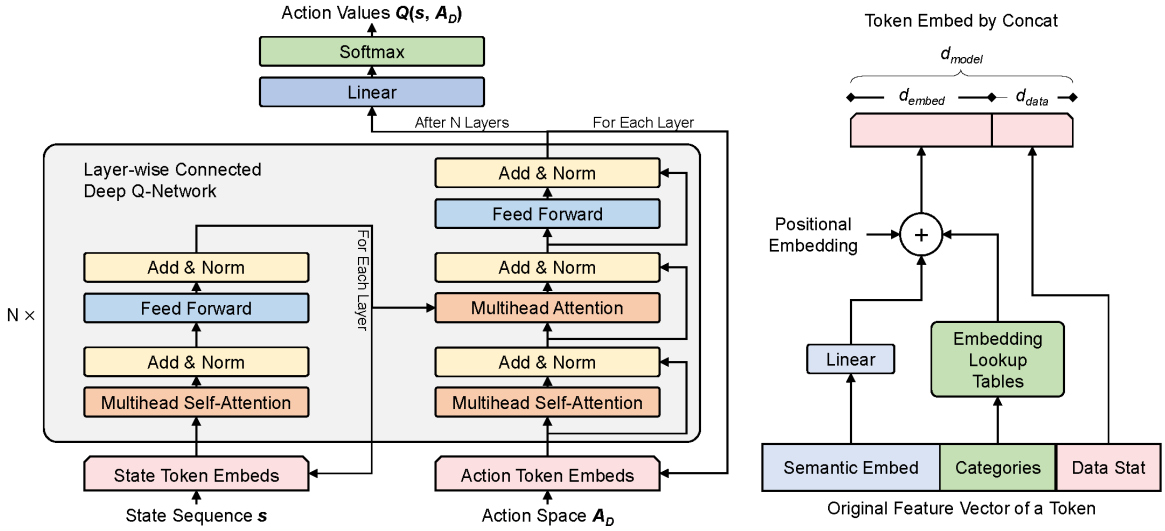
3.2 Action Value Approximator via DQN

For the following reasons, deep neural networks (DNNs) are chosen as $q_*(s, a)$ approximator. First, DNN is well-known for its ability to generalize from an observed corpus to unseen inputs. Second, the length of an analysis sequence is dynamic. Lots of DNN structures could handle varying input lengths. Third, to incorporate semantic information, one ideal approach is to adopt semantic embedding for the free texts in field name. Such embedding is also best suited for DNN.

As shown in Figure 2a, our DQN (deep Q-network) $Q(s, \mathcal{A}_\mathcal{D})$ takes a state s and the whole action space $\mathcal{A}_\mathcal{D}$ as its input, and calculates the estimated action values (ranging from 0 to 1) for all $a \in \mathcal{A}_\mathcal{D}$ at the same time (the output for invalid actions in $\mathcal{A}_\mathcal{D} \setminus \mathcal{A}_\mathcal{D}(s)$ are just ignored). The action space sequence[4] contains the source fields (in the order defined by the dataset $\mathcal{D}$, if any; also represents the whole context), `[SEP]`, and all the aggregation functions (in the order of usage frequency, if any).

[3] Optimal action-value function is defined by the expected discounted return for the optimal policy (Sutton and Barto 2018).

[4] `[ANA]` is the initial state, thus not included in the action space.



(a) Layer-wise Connected Transformer Model.

(b) Input Embedding Part.

Figure 2: $q_*(s, a)$ Approximator: DQN Model Architecture.

Figure 2a is a layer-wise connected variation (He et al. 2018) of Transformer encoder-decoder (Vaswani et al. 2017) architecture, modified by us for a different purpose than traditional NMT (neural machine translation). Keys and values of each encoder layer (left half of Figure 2a) are passed directly to the corresponding decoder layer (right half of Figure 2a). Unlike the typical NMT architecture which utilizes the decoder part to generate the next word one-by-one, the "generation" of an analysis sequence is based on $Q(s, \mathcal{A}_\mathcal{D})$ and leads to a growing state sequence. Meanwhile, the action space – the input to "decoder" part of the DQN – remains the same token set for the same table.

One major challenge in Table2Analysis is how to represent action tokens as feature vector inputs to the DQN. Different from the fixed vocabulary of tokens in typical NLP tasks, analysis language has infinite possibilities of field tokens. Fortunately $|\mathcal{F}_\mathcal{D}|$ (the number of fields in a table) is usually quite small, which means $|\mathcal{A}_\mathcal{D}|$ is also small. So instead of the typical NLP approach of representing each token as its global vocabulary index, we directly add an input embedding network (shown in Figure 2b) to convert the rich features of each analysis token into an embedding vector. This input embedding is shared across both state sequence and action space inputs. Three kinds of token features[5] are fused together:

- Semantic embedding: Any composition result such as the embedding of field name could be taken as an input. It will be resized by a linear layer to the length d_{embed}.
- Categories: Each categorical feature will also be mapped to a d_{embed}-size embedding through a trainable lookup table. Token type and aggregation function type are two categorical features defined by the analysis language. Corpus specific features such as data value types (*e.g.*, string, decimal, etc.) are also included.
- Data features: d_{data} features are directly appended to the token embedding by concatenation. They capture the statistic and distribution information of a field's data values.

Besides Transformer architecture, RNN is also widely used in seq2seq problems. However, plain seq2seq encoder-decoder does not work in Table2Analysis case because its output values have fixed length of the global vocabulary size, which is infinite in our case. One future work is to try complicated modifications on the existing RNN implementations, including varying-size vocabulary and reference mechanism such as pointing or copying (Gu et al. 2016).

3.3 Heuristic Beam Searching

To generate multiple sequences for analysis recommendation, Table2Analysis use the DQN as heuristic function for beam searching. In order to balance performance and efficiency, we focus on a class of drill-down beam searching algorithms which combines breadth and depth-first searching. They take the following steps to search a state space $\mathcal{S}^+_\mathcal{D}$:

1. Initially, the search frontier only contains the state `[ANA]`.
2. For each round, take at most $BeamSize$ top scored partial sequences out from the frontier:
   (a) For each state in the beam, greedily drill down (choose a with highest $Q(s, a)$ to append) until a complete sequence is generated. Each non-optimal state sa from each expansion (evaluation of $Q(s, \mathcal{A}_\mathcal{D})$ for all $a \in \mathcal{A}_\mathcal{D}(s)$) is put into the frontier with $Q(s, a)$ as its score.
   (b) No more rounds if #expansions exceeds $ExpandLimit$.

For the customized bream searching, the action value estimations could be directly used as scores for ranking reached states of different lengths. This is one reason why in §2.2 we did not adopt the classical maximum likelihood definitions which only give scores for local comparison.

[5] To provide position and order info, positional embedding same as the trigonometric one in (Vaswani et al. 2017) is added.

3.4 DQN Training

The DQN $Q(s, \mathcal{A}_\mathcal{D})$ could be trained by widely used teacher forcing (Williams and Zipser 1989) where only next action values of the target prefixes $\mathcal{T}_\mathcal{D}$ (of all $\mathcal{D}$ from corpus) are considered. Since much of the large state space $\mathcal{S}_\mathcal{D}^+$ remains untouched, teacher forcing could be done quickly.

However, teacher forcing would lead to the exposure bias issue (Ranzato et al. 2015) in sequence generation. During training a DQN is only exposed to the ground truth states (target prefixes) while at inference the DQN has only access to its own predictions. As a result, during generation the DQN can potentially deviate quite far from the actual sequence to be generated. Thus only seeing the partial targets would lead to a biased estimation. To train a better DQN, more explorations are needed on the rest of the state space $\mathcal{S}_\mathcal{D} \setminus \mathcal{T}_\mathcal{D}$.

In order to bridge the gap between training and inference, we generate samples using the customized beam searching with the DQN itself as heuristic function. By iterations of updating the DQN with samples generated from itself, the exposure bias will gradually eliminated by distribution shift. In this **Search Sampling** process, we adopt several techniques from reinforcement learning (Ranzato et al. 2015):

- Replay memory: During search we put encountered states into a replay memory after each expansion in step 2a; The estimator $Q(s, \mathcal{A}_\mathcal{D})$ is trained periodically by randomly generating a batch of sample states from the memory.
- OU noise: As the source of exploration randomness, diminishing[6] Ornstein-Uhlenbeck noises (Lillicrap et al. 2016) are added to the $Q(s, \mathcal{A}_\mathcal{D})$ estimation results.
- Multiple agents: Searching for multiple tables to accelerate sampling and fill the memory with diverse samples.
- Pretrain: Teacher forcing could help avoid the cold start problem by first train $Q(s, \mathcal{A}_\mathcal{D})$ under the supervision of the most informative part of the state space – $\mathcal{T}_\mathcal{D}$.

As discussed in §3.1, we already know the precise optimal action value $q_*(s, a)$ for each state $a \in \mathcal{A}_\mathcal{D}$. Thus Search Sampling is actually a supervised learning[7] process.

In §4.5 we will discuss in more details why previous techniques (Bengio et al. 2015; Ranzato et al. 2015) for exposure bias in NL sequence generation does not work in our scenario of analysis sequence generation.

4 Experiments

We build our PivotTable corpus for training and evaluating of Table2Analysis framework. Multiple experiments for tuning hyper-parameters and testing are run on the Azure Cloud using Standard NCv3 (24 CPUs, 448 GB memory, 4 NVIDIA Tesla V100 16G-memory GPUs) VM nodes.

[6]Intuitively, at an early training stage the action-value estimator is less accurate and less trustable, so there needs a larger chance to take exploring actions other than the model estimated "optimal" action; while when the model is well trained, the action-value estimator is more accurate and trustable, so there needs a larger chance to stick to exploiting actions suggested by the model.

[7]Temporal-difference methods and its improvements such as double DQN, soft update, *etc.* from RL are not used here since we already know $q_*(s, a)$ is 1 or 0 by whether sa is a target prefix.

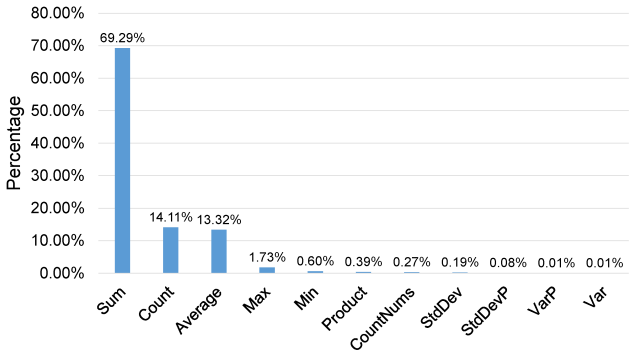


Figure 3: Usage Distribution of Aggregation Functions.

4.1 PivotTable Corpus

PivotTable (Alexander and Jelen 2001) represents a large class of analyses covered by the analysis language. It follows the analysis language definition (Definition 2.2), and adds one more constraint: There are exactly two dimension segments in the ⟨dim(s)⟩ part of ⟨analysis⟩. In other words, there is exactly one `[SEP]` in ⟨dim(s)⟩, which splits dimensions into two segments, each leads to one breakdown hierarchy. In the terminology in Excel PivotTables, one of the segment is called "row field(s)" and another is called "column field(s)". The measure field is also called "value field" in Excel.

Our PivotTable corpus contains $74,299$ unique English Excel spreadsheet files (with PivotTables) that are crawled from public Web. We utilize OpenXML to extract PivotTables and their corresponding source datasets. After excluding rare (less than 1%) datasets with extreme sizes (*i.e.*, > 128 fields) and incomplete PivotTables (*i.e.*, 0 measure), there remain $121,593$ datasets with $186,169$ PivotTables in the corpus. We further split the PivotTables that have multiple measures into multiple PivotTables, each having a single measure. The Excel files are allocated for training, validation, and testing in the ratio of $7:1:2$. Hyper-parameter experiments are based on the validation set, while overall accuracy and effectiveness comparisons against the baseline methods use the testing set.

4.2 Token Features

As mentioned in Figure 2b, each field of a dataset has three kinds of input features: semantic embedding, categorical features and data statistics. These features are concretely designed based on our domain knowledge of PivotTables.

Semantic embedding features are calculated from the name of a token (*e.g.*, header title string of a field or description text of a function) using the averaged output of all token embedding vectors (768d) from BERT model (Devlin et al. 2018). Besides the two default categorical features (token type and aggregation function type) in §3.2, field type and token segment type are also included. As shown in Figure 3, there are 11 predefined aggregation functions in Excel PivotTables.

Data statistic features are listed in Table 1. We designed 16 features trying to capture the important statistics that could be informational for common analysis recommendation. All features are calculated for the numeric fields while applicable ones are calculated for the string fields.

Feature	Meaning	Numeric	String
AggrPercentFormatted	Proportion of cells having percent format	✓	
Aggr01Ranged	Proportion of values ranged in 0-1	✓	
Aggr0100Ranged	Proportion of values ranged in 0-100	✓	
AggrIntegers	Proportion of integer values	✓	
AggrNegative	Proportion of negative values	✓	
CommonPrefix	Proportion of most common prefix digit / char	✓	✓
CommonSuffix	Proportion of most common suffix digit / char	✓	✓
KeyEntropy	Entropy by values	✓	✓
CharEntropy	Entropy by digits / chars	✓	✓
ChangeRate	Proportion of different adjacent values	✓	✓
PartialOrdered	Maximum proportion of increasing or decreasing adjacent values	✓	
Cardinality	Proportion of distinct values	✓	
Spread	Cardinality divided by range	✓	✓
Major	Proportion of the most frequent value	✓	✓
Benford	Distance of the first digit distribution to real-life average	✓	
OrderedConfidence	Indicator of sequentiality	✓	

Table 1: Data Statistic Features.

4.3 DQN Hyper-parameters

The first series of experiments run as the variations of the DQN pre-training (teacher forcing) discussed in §3.4. Since the range of the optimal action value $q_*(s, a)$ is $\{0, 1\}$, we choose DQN hyper-parameters by comparing the precision, recall and F1 scores of binary classification on each valid (s, a) pair for all $s \in \mathcal{T}_\mathcal{D}$. As shown in Table 2, the following important variations are controlled for comparisons:

- Three feature ablations: Using all features, without data statistic features, and without semantic embeddings.
- Two DQN size hyper-parameters[8]: The "small" one where $N = 4$, $h = 8$, $d_{ff} = 192$ and $d_{model} = 96$; The "large" one where $N = 6$, $h = 12$, $d_{ff} = 384$ and $d_{model} = 192$. They have ∼0.81M and ∼4.60M parameters respectively.
- Four settings of class weight of the negative log-likelihood loss function: $(1, 1)$, $(0.8, 1)$, $(0.2, 1)$ and $(0.08, 1)$ for the zero and one action value classes from $q_*(s, a)$.

Other hyper-parameters such as dropout rate ($= 0.1$) and epoch rounds ($= 30$) are derived from the above ones or fixed for all the pretrain experiments. From Table 2 we observe:

- Our input embedding layer (designed for the open vocabulary of analysis language in §3.2) works well even with feature ablation. Meanwhile, semantic embeddings have higher feature importance than data statistics.
- Class weight of the loss function balances precision and recall with trade-off. $(0.8, 1)$ is a sweet spot with highest F1 score. Thus we use the corresponding pretrain models for further experiments in upcoming sections.

[8]DQN size hyper-parameters include: The number of layers N, the number of attention heads h and the size of the feed forward layer d_{ff} as shown in Figure 2a; Hidden embedding size d_{model} (divisible by h) as shown in Figure 2b.

4.4 Search Sampling Hyper-parameters

The second series of experiments are the variations of the search and sampling process discussed in §3.4. Here the exploration of non-target prefixes is guided by both the search algorithm and its DQN heuristic function. The following configurations are considered for comparisons:

- Effectiveness of starting Search Sampling with a pre-trained model against randomly initialized model.
- Trade-off between computational cost (large and small model sizes) and accuracy (recall at top-k).
- The same loss function class weight options as the pre-training experiments in §4.3, except $(0.08, 1)$.
- The explore strategies during training affects sampling distribution. For the drill-down Step 2a in §3.4, the default choice is to add OU noise to $Q(s, \mathcal{A}_\mathcal{D})$, which will put into replay memory samples that are currently favored by Q during search. For comparison, another "blind explore" choice is to force $Q(s, a) = 1$ if $sa \in \mathcal{T}_\mathcal{D}^+$ and $rand(0, 1)$ otherwise. This will put all positive and random negative samples into the memory.

Other hyper-parameters for searching and sampling are determined by the above ones or fixed. *E.g.*, $(ExpandLimit, BeamSize)$ = (100, 4). For OU noises, we set $\theta = 0.15$, $\sigma = 0.2$, $\mu = \vec{0}$, and diminish the scale of noise as the epoch number grows (scaling factor starts at 0.9, and ends at 0.001 by multiplying 0.8 after each epoch).

The evaluation metric for the DQN trained after Search Sampling is recall at top k (R@k, $k = \{1, 3, 5\}$), which is widely used in evaluations of recommendation systems. DQN heuristic $q(s, \mathcal{A}_\mathcal{D})$ and the drill-down beam search is applied to each different table in the validation set to generate a list of analysis recommendations. R@k means the ratio over the tables that the list covers a target analysis sequence.

Ablation	All Features								No Data Stat.		No Semantic	
Weight	1.0		0.8		0.2		0.08		0.8		0.8	
Size	large	small	large	small	large	small	large	small	large	small	large	small
Precision	**0.9238**	0.9064	0.9099	0.8918	0.7825	0.7170	0.7016	0.5643	0.9057	0.8945	0.8804	0.8413
Recall	0.8639	0.7727	0.8785	0.7965	0.9411	0.8969	**0.9540**	0.9465	0.8713	0.7708	0.8278	0.7358
F1	0.8928	0.8342	**0.8939**	0.8415	0.8545	0.7970	0.8086	0.7070	0.8882	0.8280	0.8533	0.7850

Table 2: Validations of DQN Hyper-parameters.

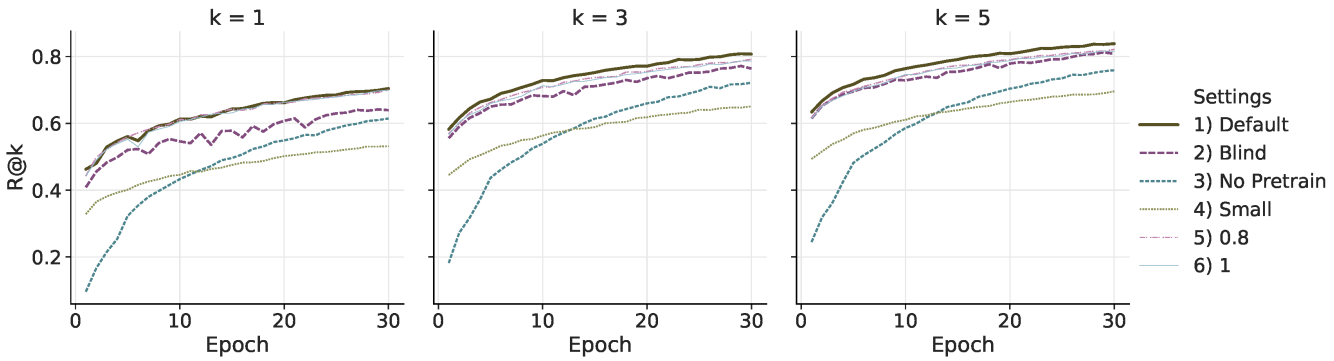


Figure 4: R@k on Validation Set over 30 Epochs of Search Sampling with Settings in §4.4.

Figure 4 shows R@k validations after each of the 30 epochs for 6 hyper-parameter settings. Each setting varies at one configuration from the "1) Default" setting that starts from pre-trained large model with weight $(0.8, 1)$ and explores with OU noise. We have the following observations:

- The clear increasing lines of R@k indicates Searching Sampling could relieve the exposure bias issue for analysis sequence generation we discussed in §3.4.
- The techniques we adopted in Searching Sampling improve the learning process. *E.g.*, OU noise is better than blind explore when comparing 1) and 2), which indicates that diminishing noise may better regularize against the distribution shift. Effectiveness of using pre-trained model is also verified in the comparison between 1) and 3).
- Large model remarkably outperforms small model (1 *vs.* 4). 0.2 is a sweet spot of class weight (1 *vs.* 5, 6).

4.5 Baseline Comparisons and Overall Accuracy

After comparing hyper-parameters in §4.3 and §4.4 on the validation set, in this section we evaluate baseline methods and different variations of Table2Analysis on the testing set.

Scheduled Sampling Sampling methods such as DAD (Venkatraman, Hebert, and Bagnell 2015) and MIXER (Ranzato et al. 2015) were proposed to tackle the exposure bias issue in natural language generation problems. At each time step and with a certain probability, DAD takes as input either the prediction from the model at the previous time step or the ground truth data. MIXER (Ranzato et al. 2015) borrows ideas from DAD and train with both token level loss and sequence loss. MIXER utilizes a annealing schedule to control the trade-off between the two levels of losses.

However, neither DAD nor MIXER can be directly applied in Table2Analysis due to different success metrics and analysis grammar restriction. Both DAD and MIXER use scheduled sampling to avoid yielding errors that can accumulate quickly along the generated sequence. Based on relaxed metrics such as BLEU score, they both assume that after a wrongly predicted token in the sequence, upcoming predictions should still match the remaining ground truth. This assumption is totally wrong in Table2Analysis. As discussed in §3.1, action value is 1 if and only if it leads to a prefix of target sequences, which means once a wrong prediction is taken, all upcoming action values are always 0. In addition, the analysis grammar restriction also conflicts with the assumption. For example, when the target PivotTable is `[ANA]` [*Sales*] `[SEP]` [*Region*] `[SEP]` `[Sum]`, if a wrong prefix `[ANA]` [*Sales*] `[SEP]` `[SEP]` is generated, then DAD and MIXER will assume `[SEP]` is the next correct token (5th token in the target), which violates the PivotTable grammar of exactly two dimension breakdowns. (As introduced in §4.1, the number of `[SEP]` in a PivotTable should be exactly two, representing two dimension breakdowns.)

We still try to modify DAD and MIXER and propose a baseline scheduled sampling algorithm for DQN training. With the pre-trained DQN as starting point, it further trains the DQN with an annealing schedule to sample either ground truth or model generated states. More Specifically, at each epoch, ground truth states are taken for the first l steps, and the model generated states (sampled with the estimated values as weights) are taken for the rest of steps. l is periodically decreased so that as the training progresses, more model generated states are sampled. This allows the model to be more aware of how it will be used during inference.

	1)	4)	Scheduled	Pretrain	Random
R@1	**0.6518**	0.5177	0.0302	0.3108	0.0016
R@3	**0.7471**	0.6195	0.0667	0.4258	0.0109
R@5	**0.7802**	0.6612	0.0817	0.4823	0.0121

Table 3: Test Highlights.

Comparisons on Test Set In Table 3, model 1) 4) from epoch 30 of Figure 4 are evaluated on the testing set (see §4.1), together with three baselines: the DQN with the baseline scheduled sampling[9], the pre-trained large DQN (teacher forcing with no Search Sampling), and a randomly initialized large DQN. The test is run on 1 node and on average takes $< 0.5s$ for a dataset ($\sim$1,500s for $\sim$23,700 testing datasets). The best model 1) achieves R@1$= 0.652$, R@3$= 0.747$ and R@5$= 0.780$, much higher than the pretrain-only model, which means it successfully relieve the exposure bias issue. Meanwhile, the baseline scheduled sampling leads to recall numbers lower than the pretrain model, which means traditional scheduled sampling techniques fails to tackle the exposure bias issue for Table2Analysis.

Empirical Study To facilitate more intuitive understanding of the recommendation results, below we list the top-5 recommendations by model 1) for the running example of the sales table in Figure 1a.

- `[ANA]` [*Sales*] `[SEP]` [*Regions*] `[SEP] [Sum]`
- `[ANA]` [*Sales*] `[SEP]` [*SalesRep*] `[SEP] [Sum]`
- `[ANA]` [*Sales*] `[SEP]` [*Data*] `[SEP]` [*Region*] `[Sum]`
- `[ANA]` [*Sales*] `[SEP]` [*SalesRep*] `[SEP]` [*Region*] `[Sum]`
- `[ANA]` [*Sales*] `[SEP]` [*Date*] `[SEP] [Sum]`

Impressively, Table 2 and 3 are also (partly) in the top recommendations by model 1) based on raw experimental records, which shows that Table2Analysis models recommend semantic meaningful analyses.

5 Related Work

Analysis Recommendation For data analysis and insight recommendation, collaborative filtering or statistical significance are usually used in existing systems.

Traditional collaborative filtering approaches (Marcel and Negre 2011; Milo and Somech 2018) are applicable to scenarios where recommendations are made based on user history on the exact same table or small number of tables with aligned schema. However, such scenarios are different from that in our paper where neither user profile / history data nor global schema is available. Therefore, those traditional approaches do not work for such tasks. From another perspective, Table2Analysis framework could be viewed as a new approach of collaborative filtering by learning the common wisdom and implicitly matching the common schema among large amount of different tables through neural nets.

[9]For scheduled sampling, we set $l = 15$ (which covers $> 99\%$ of the PivotTable sequences) and decrease it by 1 every 2 epochs.

Statistical significance methods have been verified as effective in (Tang et al. 2017; Ding et al. 2019). However, statistical significance should not be the only criterion for insights discovery and analysis recommendation. Semantic meaningfulness of analysis is quite important from human perspective. Existing work did not well use the semantic information presented in tables trough free text from humans. It is crucial to incorporate both unstructured free texts and structured values from a table to guide the structured learning and generation of data analysis. From this perspective, Table2Analysis technique aims to complement the existing techniques by adding the aspect of semantic meaningfulness.

Structured Learning and Prediction Learning and generating structures (BakIr et al. 2007) has many applications and lots of them are related to natural language processing. Some related examples are: Data to Text (Generating different kinds of text and documents from structured data (Liu et al. 2018; Wiseman, Shieber, and Rush 2017).), Text to SQL (Generating queries from natural language (Zhong, Xiong, and Socher 2017; Yu et al. 2018).) and Program Synthesis (Generating programs to meet the clear rules and example input/output pairs. Such as DeepCoder (Balog et al. 2017) and NGDS (Kalyan et al. 2018).).

Behind these work are some common model architectures widely adopted, such as the attention based models (Vaswani et al. 2017; Devlin et al. 2018; He et al. 2018; Yang et al. 2019) which greatly inspired our DQN model design. Unfortunately, most existing implementations of seq2seq encoder-decoder architectures are based on a fixed vocabulary assumption, which is natural for NLP but does not fit the input and output requirements in Table2Analysis framework. As discussed in §3.2, we have to design and implement specific model architectures for the table input.

The exposure bias of sequence generation is discussed in (Bengio et al. 2015; Ranzato et al. 2015; Lamb et al. 2016). As we have discussed in §3.4 and §4.5, the existing techniques were designed for natural language generation and does not apply to analysis sequence generation. Thus we design our Search Sampling method for Table2Analysis.

Deep Reinforcement learning Some terms (such as action value, exploration *vs.* exploitation) and techniques (such as replay memory, OU noise) we use in Table2Analysis comes from the reinforcement learning literature, especially the value-based model-free deep RL which was first successfully applied in gaming environments (Mnih et al. 2015). Our Search Sampling process is greatly inspired by RL work such as (Mnih et al. 2015) and (Lillicrap et al. 2016).

6 Conclusion

In this paper, we propose the Table2Analysis framework to learn and recommend Common Analysis patterns. The proposed techniques can generate multiple Common Analysis sequences for a given dataset, thus greatly boost the productivity of data analysis tasks. It also enables future improvements on downstream tasks such as insights recommendation and natural language query generation.

References

Alexander, M., and Jelen, B. 2001. *Pivot Table Data Crunching*. Pearson Education.

BakIr, G.; Hofmann, T.; Schölkopf, B.; Smola, A. J.; and Taskar, B. 2007. *Predicting Structured Data*. MIT press.

Balog, M.; Gaunt, A.; Brockschmidt, M.; Nowozin, S.; and Tarlow, D. 2017. Deepcoder: Learning to write programs. In *5th International Conference on Learning Representations*, ICLR '17 Poster.

Bengio, S.; Vinyals, O.; Jaitly, N.; and Shazeer, N. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In Cortes, C.; Lawrence, N. D.; Lee, D. D.; Sugiyama, M.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 28*, NIPS '15. Curran Associates, Inc. 1171–1179.

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR* abs/1810.04805.

Ding, J.; Han, S.; Xu, Y.; Zhang, H.; and Zhang, D. 2019. Quickinsights: Quick and automatic discovery of insights from multi-dimensional data. In *Proceedings of the 2019 ACM International Conference on Management of Data*, SIGMOD '19.

Gu, J.; Lu, Z.; Li, H.; and Li, V. O. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1631–1640.

He, T.; Tan, X.; Xia, Y.; He, D.; Qin, T.; Chen, Z.; and Liu, T.-Y. 2018. Layer-wise coordination between encoder and decoder for neural machine translation. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS '18, 7955–7965. USA: Curran Associates Inc.

Kalyan, A.; Mohta, A.; Polozov, A.; Batra, D.; Jain, P.; and Gulwani, S. 2018. Neural-guided deductive search for real-time program synthesis from examples. In *6th International Conference on Learning Representations*, ICLR '18 Poster.

Lamb, A. M.; ALIAS PARTH GOYAL, A. G.; Zhang, Y.; Zhang, S.; Courville, A. C.; and Bengio, Y. 2016. Professor forcing: A new algorithm for training recurrent networks. In Lee, D. D.; Sugiyama, M.; Luxburg, U. V.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 29*, NIPS '16. Curran Associates, Inc. 4601–4609.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2016. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, ICLR '16 Poster.

Liu, T.; Wang, K.; Sha, L.; Chang, B.; and Sui, Z. 2018. Table-to-text generation by structure-aware seq2seq learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, AAAI '18.

Marcel, P., and Negre, E. 2011. A survey of query recommendation techniques for data warehouse exploration. In *Actes des 7èmes journées francophones sur les Entrepôtsde Données etl'Analyseen ligne, Clermont-Ferrand, France,EDA 2011, Juin 2011*, EDA '11, 119–134.

Milo, T., and Somech, A. 2018. Next-step suggestions for modern interactive data analysis platforms. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, 576–585. New York, NY, USA: ACM.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

Ranzato, M.; Chopra, S.; Auli, M.; and Zaremba, W. 2015. Sequence level training with recurrent neural networks.

Sutton, R. S., and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Tang, B.; Han, S.; Yiu, M. L.; Ding, R.; and Zhang, D. 2017. Extracting top-k insights from multi-dimensional data. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, SIGMOD '17, 1509–1524. ACM.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, NIPS '17, 5998–6008.

Venkatraman, A.; Hebert, M.; and Bagnell, J. A. 2015. Improving multi-step prediction of learned time series models. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI '2015.

Williams, R. J., and Zipser, D. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation* 1(2):270–280.

Wiseman, S.; Shieber, S. M.; and Rush, A. M. 2017. Challenges in data-to-document generation. *arXiv preprint arXiv:1707.08052*.

Yang, Z.; Dai, Z.; Yang, Y.; Carbonell, J.; Salakhutdinov, R.; and Le, Q. V. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.

Yu, T.; Li, Z.; Zhang, Z.; Zhang, R.; and Radev, D. 2018. Typesql: Knowledge-based type-aware neural text-to-sql generation. *arXiv preprint arXiv:1804.09769*.

Zhong, V.; Xiong, C.; and Socher, R. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.