Adversarial TableQA: Attention Supervision
for Question Answering on Tables

Minseok Cho   WHATJR@YONSEI.AC.KR
Yonsei University, Seoul, South Korea

Reinald Kim Amplayo[*]   REINALD.KIM@ED.AC.UK
University of Edinburgh, UK

Seung-won Hwang[†]   SEUNGWONH@YONSEI.AC.KR
Yonsei University, Seoul, South Korea

Jonghyuck Park   JHPARK0615@NCSOFT.COM
Knowledge Lab. NCSOFT, South Korea

Abstract

The task of answering a question given a text passage has shown great developments on model performance thanks to community efforts in building useful datasets. Recently, there have been doubts whether such rapid progress has been based on truly understanding language. The same question has not been asked in the table question answering (TableQA) task, where we are tasked to answer a query given a table. We show that existing efforts, of using "answers" for both evaluation and supervision for TableQA, show deteriorating performances in adversarial settings of perturbations that do not affect the answer. This insight naturally motivates to develop new models that understand question and table more precisely. For this goal, we propose NEURAL OPERATOR (NEOP), a multi-layer sequential network with attention supervision to answer the query given a table. NEOP uses multiple Selective Recurrent Units (SelRUs) to further help the interpretability of the answers of the model. Experiments show that the use of operand information to train the model significantly improves the performance and interpretability of TableQA models. NEOP outperforms all the previous models by a big margin.

Keywords: Question Answering

1. Introduction

Thanks to the advancements of deep learning for natural language understanding, the task of question answering (QA) has grown since its first occurrence. Specially, QA on structured data has gained more attention from people who want to seek effective methods for accessing these data, as the amount of the structured form of data grows. There are two types of QA tasks on structured data: (1) knowledge base (KB) such as Freebase which is one of the representative the structured data (i.e. KB-QA) (Zettlemoyer and Collins, 2005; Berant et al., 2013; Cui et al., 2017), and (2) database such as table which starts to get the spotlight recently (i.e. TableQA) (Pasupat and Liang, 2015). In this paper, we are particularly interested in the TableQA task.

Meanwhile, in the context of TextQA which is the task of answering a question given a text passage, existing systems (Min et al., 2017; Liu et al., 2017b) have been shown recently, to be

---

[*] Work done while in Yonsei University supported by Yonsei University Research Fund of 2017-22-0082
[†] corresponding author

sensitive to adversarially-chosen inputs, such as adding a distracting sentence that does not affect the answer (Jia and Liang, 2017). That is, a new evaluation system, which motivates more robust models that truly understand language, is being considered to be crucial for QA tasks.

The goal of this paper is thus to study an adversarial evaluation system for TableQA, and QA models that are robust to such attacks. Current state-of-the-art models in TableQA (Pasupat and Liang, 2015; Neelakantan et al., 2016; Yin et al., 2016) are trained upon pairs of natural language questions and answer, such as (a) and (b) shown in Figure 1. However, such evaluation may count "*spurious*" programs that accidentally lead to the correct answers even by using the wrong cells or operations, as correct answers as also observed in (Goldman et al., 2017). For example, in Figure 1, the correct program (i.e. (f) in the figure) leads to the correct answer, while spurious programs (i.e. (c), (d) and (e) in the figure) also derive the correct answer using wrong reasons.

Based on this observation, we argue that using only answer annotation is too weak for both evaluation and training purposes. That is, only the query and the final answer are given in the existing datasets, such that no information is given to distinguish the right answer derived from spurious programs. This leads to confusion in the model on which of the possible ways should it choose to arrive at the final answer. For example, wrong cells can be in the same column (i.e. (d) in the figure), or they can be in a similar column using the same operation (i.e. (e) in the figure).

Figure 1: Example data where current TableQA systems may fail. Abbreviation glossary: 3B - triple, SB - stolen base, CS - caught stealing.

Recently, SQL statements were proposed as annotations to improve the performance of models (Zhong et al., 2017). They can uniquely define the answer in Figure 1, and hence not spurious, but with the following overheads: First, the kind of queries that SQL statements can express is very limited; for example, aggregating values from multiple cells in multiple columns would require a very complicated SQL query that is hard to learn and not supported in the current literature (Zhong et al., 2017). Second, labeling SQL statements is hard, especially as the text query becomes more complicated, and only few SQL enthusiasts would qualify as annotators, which limits the crowdsourcing of training resources from normal non-technical users.

Because of these reasons, SQL annotations that reflect domain-specific query needs can be obtained, only by a highly limited set of annotators that are both SQL and domain experts. To relax the former condition, the most widely adopted SQL annotated dataset was created by automatically generating SQL annotations first, such that SQL answers can be translated into corresponding natural language questions by non-technical domain experts (which we call **annotate-then-query** method). This method dictates domain experts to write questions according to arbitrary generated SQL queries, instead of writing those reflecting their information needs.

In this paper, we propose *attention supervision* that does not require SQL expertise. Instead, for attention supervision, we propose to use "operand information", the set of correct cells to be selected and operated that provides much richer information about the final answer. In our running example, aside from feeding the final answer 14 as supervision, we also provide the location of the cells 5, 2, and 7 (i.e. cells of (f) in the Figure 1). Operand information is advantageous over both answer and

SQL annotations, because of two reasons. First, all SQL queries (and beyond) can be expressed and annotated using operand information. In the example where multiple cells from multiple columns are aggregated, one can easily mark the cells as operand information. Second, given the text query, labeling is easy because one only needs to mark the cells to be selected and operated. This makes it easy for any person who understands the query to do annotations directly, guided solely by their curiosity, generating more realistic question workloads than **annotate-then-query** datasets, while achieving the scale as effectively.

To this end, our contributions can be divided into three parts. First, we study the limitation of answer and SQL annotations of existing datasets and present WIKIOPS dataset which annotates operand information. Second, we present NEURAL OPERATOR (NEOP), a neural network that learns the correct operands to achieve the final answer. NEOP improves the robustness by (A) supervising attention weights through the operand information and (B) using three-layer Selective Recurrent Units (SelRUs) to select the following at each timestep: column, pivot, and parameter. Finally, we compare the performance of NEOP and other baseline models, to show the robustness over adversarial examples.

2. Related work

2.1. Semantic parsing models

The TableQA task is solvable using semantic parsers which are trained on the question-answer pairs (Kwiatkowski et al., 2013; Krishnamurthy and Kollar, 2013). This method is effective because it does not need expensively annotated pairs of question and program (Zettlemoyer and Collins, 2005). However, semantic parsing requires the engineering of hand-crafted and domain-specific grammar or pruning strategies (Wang et al., 2015). The performance of the model is then heavily dependent on the effectiveness of the utilized grammar or pruning strategy (Pasupat and Liang, 2015).

2.2. Neural models

Recent methods eliminate the need to use traditional semantic parsing techniques by using neural networks. NEURAL ENQUIRER (NEEN) (Yin et al., 2016) is a fully neuralized method to execute a natural language query and return an answer from a table. However, NEEN can only return a single cell as an answer and cannot output an answer from an operation (e.g. sum and average), thus is not able to answer the query in Figure 1. NEURAL PROGRAMMER (NEPR) (Neelakantan et al., 2016) is a neural network model that allows operation on multiple cells, but it only supports three operations: max, min and count.

2.3. Attention supervision

Attention supervision has been studied in the areas of visual question answering (VQA) and neural machine translation (NMT) (Das et al., 2017; Liu et al., 2017a), where attention that is close to human perception is observed to be more effective for end-task. This observation naturally motivates to explicitly supervise the attention (Yu et al., 2017; Mi et al., 2016) to follow the human's attention. For QA tasks, the following work can be viewed as a form of attention supervision: (Rajpurkar et al., 2016) introduced spans and (Yang et al., 2015) introduced sentence-type answers for models to easily learn the answer. Recently, (Zhong et al., 2017) proposed to use SQL statements

Dataset	Size	Real Query?	Multi-cell Answer?	Real Data?	Attention Supervision?
WIKITABLEQUESTIONS* (Pasupat and Liang, 2015)	22K queries 2K tables	✓	✓	✓	✗
NEPR data (Neelakantan et al., 2015)	Unlimited	✗	✓	✗	✗
NEEN data (Yin et al., 2016)	Unlimited	✗	✗	✗	✗
WIKISQL (Zhong et al., 2017)	81K SQL queries 24K tables	✗	✓	✓	SQL Statements
WIKIOPS* (ours)	**81K operands 24K tables**	✗	✓	✓	**Operand Information**
MLB* (ours)	**36K operands 2K tables**	✗	✓	✓	**Operand Information**

Table 1: A survey and comparison of previous TableQA datasets and our proposed WIKIOPS dataset. Datasets with asterisk (*) are publicly shared.

as additional supervision knowledgeable at SQL. In a more general context of NLP tasks, leveraging supervisions beyond the raw text has been effective in text classification (Zhao and Mao, 2017; Amplayo et al., 2018a,b) and summarization (Amplayo et al., 2018c).

2.4. Our improvements

This paper presents *operands* as a unit for both evaluation and training purposes. First, we build a dataset with operand annotation, from the manually-generated WIKISQL dataset (Zhong et al., 2017) (more discussion in the next section). Second, our model allows operation on multiple cells and is extensible to new operations (as we will discuss in Section 4.5). It also accepts operand information as additional supervision on attention weights.

3. Our dataset

We survey recent TableQA datasets and compare the characteristics of the data used in Table 1. Current public datasets can be categorized into real and synthetic queries. Only dataset that collects real queries from users is WikiTableQuestions (Pasupat and Liang, 2015), which has the advantage of reflecting real query needs. However, such manual annotation limits the query size small in size, thus neural models cannot be trained. On the other hand, automatically generated queries can be arbitrarily large, but cannot reflect real-world distributions of queries. Some datasets generate data tables synthetically as well, enjoying the unlimited size of query and data size, with the expense of them being less realistic. Recently, a dataset called WIKISQL (Zhong et al., 2017), which contains real-life tables, but not realistic questions, because, as discussed earlier, automatically generated SQL queries are annotated first, dictating to annotate matching questions (or, **annotate-then-query**).

We perform preliminary analysis based on the current datasets, with the following insight on the gap of real-life and synthetic queries.

1. More than half of the real queries in WikiTableQuestions (Pasupat and Liang, 2015) are only solvable using multiple cells, to sum all home runs of players with a certain position. All previous models assuming a single cell assumption, such as NEEN, thus cannot support real-life query needs.

Type	Operation	Query example	Logical form	#Query
Print	`all`	- What positions does the college/junior/club team, molot perm have?	`all(Pos|Team= Molot Perm)`	58K
Extremity	`min`	- What's the 2001 census of the region of Abruzzo where the 1981 census is bigger than 51092.0?	`min(2001 Census|Reg= Abruzzo,1981 Census >51092.0)`	5K
	`max`	- The record of 7-3 had the largest attendance of what?	`max(Attendance |Record=7-3)`	5K
Totality	`count`	- What is the total number of music genre/style in which the lyrics are a detective?	`count(Genre|Theme =Detective Story)`	7K
	`sum`	- What is the total number of points team Alfa Romeo 184T won?	`sum(Points|Team= Alfa Romeo 184T)`	3K
Centrality	`avg`	- Tell me the average attendance for week of 11	`avg(Attend|Week=11)`	3K
Total Queries				81K

Table 2: WIKIOPS dataset statistics per operation grouped into types with corresponding sample queries

2. WikiTableQuestions queries may include unconventional aggregations, such as getting the ranges, top 3 values, or difference. Desirable models should rapidly adapt to domain-specific operation, unstudied in all existing models.

3. Although providing additional supervision helps, SQL statements provide limitation on the kind of expressions used as query. As an example, the query "sum the total second bases, third bases, and homeruns of player A" would need to calculate the sum cells from multiple columns. This is difficult to express as an SQL statement. Apart from that, labeling instances with SQL statements is very hard for non-technical users.

To compare and contrast SQL with its operand annotations, we present WIKIOPS[1], an altered version of the original WIKISQL dataset (Zhong et al., 2017), by transforming the SQL statements into operand information. This can be done as follows. For the print operation, we execute the original SQL statements. For other operations, we modify the SELECT statement by removing the aggregation function (e.g. SELECT SUM(a) to SELECT a). Sample data instances are shown in Table 2. We note that this dataset is skewed towards having many `all` operations and the total number of operations are smaller.

Meanwhile, to motivate against the practice of **annotate-then-query** of WIKIOPS, we also crowdsourced 21 natural language question templates from baseball experts and annotate operands to answer such queries, over subsets of player stats tables gathered from the MLB website. Collected expert queries include complex patterns (e.g. "give me the smallest", "what is the minimum", etc.). We call this dataset, MLB dataset, paraphrased into 36k questions. We note that this dataset, being generated from expert questions, is expected to reflect more realistic question patterns of the given domain.

---

1. We share WIKIOPS dataset here: https://github.com/MinseokCho/NeuralOperator

4. Our model: NEURAL OPERATOR

In this section, we present our model NEURAL OPERATOR (NEOP). NEOP is a multi-layer sequential network that accepts as input the query and the given table, and learns both the operands and the operation to calculate the final answer. In Figure 2, we take an example to explain how to answer the query in our model. When the table is given, any query can be divided to a sequence as follows:

1. Selection: select a column in the table and words in the query as pivot and parameter[2], then output the corresponding rows. (Section 4.2 and 4.3)

2. Projection: project the selected column to the selected rows, then output the qualifying cells as operand information. (Section 4.4)

3. Aggregation: execute an aggregate operation through the projected cells from step (2) above. (Section 4.5)

As shown in the Figure 2, our model can interpret the answer through showing the highlighted words and column at each sequence. This process is very intuitive for users who want to know how the answer is generated. The full architecture of the model is shown in Figure 4.

4.1. Encoding

Word and cell encoding We use a single word embedding matrix $W_e = \{w\}$ for the initial vectors of both space-separated strings in the query and cells in the table. A word vector w_i is initially encoded using the concatenation of two encoding methods: (1) an embedding w_{i_x} initialized using Xavier initialization (Glorot and Bengio, 2010), and (2) a binary encoding w_{i_b} of the equivalent numerical value of the word, i.e. $w_i = [w_{i_x}; w_{i_b}]$. If the word does not have an equivalent numerical value (i.e. a plain string), then the binary encoding is a zero vector. If the word is a numerical value in the form $int.dec$ where int is the integer part and dec is the decimal part of the number, the binary encoding w_{i_b} is a concatenation of the binarized value of the integer part and the decimal part, i.e. $w_{i_b} = [int_2; dec_2]$. In experiments, we use 300 dimensions each for the word encoding and binary encoding and 15 dimensions for decimal part of the number. Our intention behind choosing 15 is to represent most numbers using 16 bits for decimal part. Our initial experiments showed that the inclusion of the binary encodings increase the performance of the model.

Figure 2: An example query ran through NEOP. 'number of walks', 'slugging percentage' and 'position' corresponds to the column 'BB', 'SLG' and 'Pos' in the baseball, respectively.

2. Here on, we define *pivot* as the conditional operator (e.g. greater) and *param* as the conditional operand (e.g. 0.547).
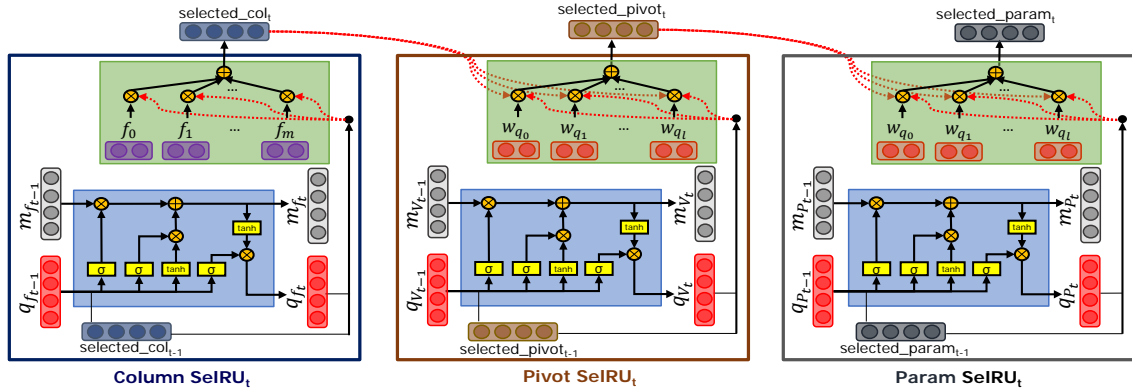
Figure 3: At timestep t, Column, Pivot, and Param SelRUs and their sub-components. All SelRUs have both LSTM and the Attentive Pooling sub-components. Pivot and Param SelRU have one more parameter than Column SelRU, received from the previous SelRU layer.

Query encoder A query q is a natural language text composed of multiple words $q_1, q_2, ..., q_l$ where l is the length of the query, and these words are represented by $w_{q_1}, w_{q_2}, ..., w_{q_l}$ from word embedding matrix W_e. As used in practice (Chung et al., 2014), we encode the query using bidirectional GRU (Bahdanau et al., 2014), i.e. $\overrightarrow{h}_i, \overleftarrow{h}_i = GRU(w_{q_i}, \overrightarrow{h}_{i-1}), GRU(w_{q_i}, \overleftarrow{h}_{i+1})$. The forward and backward final states are concatenated to create the query vector, i.e. $q = [\overrightarrow{h}_l; \overleftarrow{h}_1]$.

Table encoder A table T consists of n rows and m columns and a total of $n \times m$. $T[j, k]$ is a cell value belonging to the jth row and the kth column and it is initialized as $w_{T[j,k]}$ from the word embedding matrix W_e. Moreover, each column header is represented by a field embedding f_k representing each column. The field embedding f_k is also obtained from the word embedding matrix W_e. Finally, we follow the method of updating the cell vectors $c_{j,k}$ of (Yin et al., 2016), where $w_{T[j,k]}$ is updated to include the information about the field it is currently in, i.e. $c_{j,k} = tanh(W_t[w_{T[j,k]}; f_k] + b)$.

4.2. Selective Recurrent Units

Previous neural-based TableQA models used coarse-granular representations for cells to operate, which lessens the interpretative power of the model. For example, NEURAL ENQUIRER (Yin et al., 2016) only used a single column vector, thus it is hard to know which condition and operation are selected at each timestep. NEURAL PROGRAMMER (Neelakantan et al., 2015) improves this by using separate vectors, however these vectors are *simultaneously* created.

We argue that the selection of three different elements in a condition, i.e. columns, pivots and parameters, should be *cascaded*. For example, in Figure 2, we first select the column SLG. Then, we use the column information SLG as context to select the pivot greater. Finally, we use the pivot information greater to select the parameter 0.547. These vectors are leveraged to select the correct rows at each timestep. Since the model explicitly separates the selection of the parts of the condition, interpreting the model becomes clearer, as we show in Section 5.6.

We propose Selective Recurrent Units (SelRUs), shown in Figure 3. SelRU is composed of two sub-components: (1) an LSTM layer for processing the query vector to contain the semantic information of the query as well as the previous selection information, and (2) an attentive pooling

component to focus on the current information. We use three SelRUs in turn: Column SelRU, Pivot SelRU, and Param SelRU to select the column, pivot, and parameter, respectively, at each timestep. In example of Figure 2, for column, previous selection information is `SLG` and `Pos` at timestep 3.

Column, Pivot and Param SelRUs output the selected column vector $\widetilde{f}_t$, selected pivot vector $\widetilde{v}_t$ and selected param vector $\widetilde{p}_t$ respectively for selecting each element at timestep t. To get each selected instance vector, 1) LSTM layer encodes query vector to contain the previous selection information, 2) attentive pooling component yields selected instance vector from newly encoded query vector and other inputs.

First, we can compute latent representations of query and memory:

$$q_{f_t}, m_{f_t} = LSTM(\widetilde{f}_{t-1}, q_{f_{t-1}}, m_{f_{t-1}}) \tag{1}$$

Here q_{f_t} is query vector carrying semantic information of the query and previous selection information of column, and m_{f_t} is memory vector which save previous selection information. Because we have newly encoded the query to contain the previous selection information, we can use a different query vector for each timestep. This helps to select the correct condition and row. At timestep 0, q_{f_0} is GRU-encoded vector q and m_{f_0} is initialized to zero vector.

Second, selected column vector $\widetilde{f}_t$ is obtained by attentively pooling the column vectors f_k, and selected pivot vector $\widetilde{v}_t$ and selected param vector $\widetilde{r}_t$ are calculated in similar manners as we will describe later. At timestep t, the selected column vector $\widetilde{f}_t$ is calculated as follows:

$$e_f(f_k, \widetilde{f}_{t-1}, q_{f_t}) = u_f^\top tanh(W_f[f_k; \widetilde{f}_{t-1}; q_{f_t}] + b_f) \tag{2}$$

$$a_f(f_k)_t = \frac{\exp(e_f(f_k, \widetilde{f}_{t-1}, q_{f_t}))}{\sum_{k'} \exp(e_f(f_{k'}, \widetilde{f}_{t-1}, q_{f_t}))} \tag{3}$$

$$\widetilde{f}_t = \sum_k a_f(f_k)_t \times f_k \tag{4}$$

Here $a_f(f_k)_t$ is the attention weight of the kth column at timestep t using two contexts: the selected column vector $\widetilde{f}_{t-1}$ at timestep $t-1$ and the query vector q_{f_t} containing previous selection information from LSTM layer (Eq. 1). This decision is cascaded to Pivot and Param SelRUs, in calculating the selected pivot vector and param vector $\widetilde{v}_t$ and $\widetilde{r}_t$ respectively. Each module uses the same network as column selection, only cascading the column and pivot just decided as additional contexts, as Figure 4 shows.

4.3. Row RNN

Once we obtain the selected column, pivot, and parameter vectors $\widetilde{f}_t$, $\widetilde{v}_t$, and $\widetilde{p}_t$ at timestep t, we use these vectors as the conditions to select the correct rows. To do this selection process, we use a simple RNN with multiple contexts called Row RNN to represent n row vectors at timestep t, i.e. r_{j_t}. The row vectors r_{j_t} represent the information whether to select the row or not. For each row j, Row RNN accepts six inputs: (1-3) the current selected column, pivot, and parameter vectors $\widetilde{f}_t$, $\widetilde{v}_t$, and $\widetilde{p}_t$, (4) the current selected cell vector at row r_j, i.e. $\widetilde{c}_{j_t}$, (5) the previous row vector $r_{j_{t-1}}$, and (6) the previous table vector x_{t-1}, and returns the row vector r_{j_t} as output.

The selected cell vector $\widetilde{c}_{j_t}$ contains information of the row j biased selected column at timestep t. This is obtained by performing a weighted sum of the cell vectors $c_{j,k}$ using the column attention scores from Column SelRU (i.e. $a(f_k)$ in Eq. 3), i.e. $\widetilde{c}_{j_t} = \sum_k a(f_k) \times c_{j,k}$. The table vector x_t
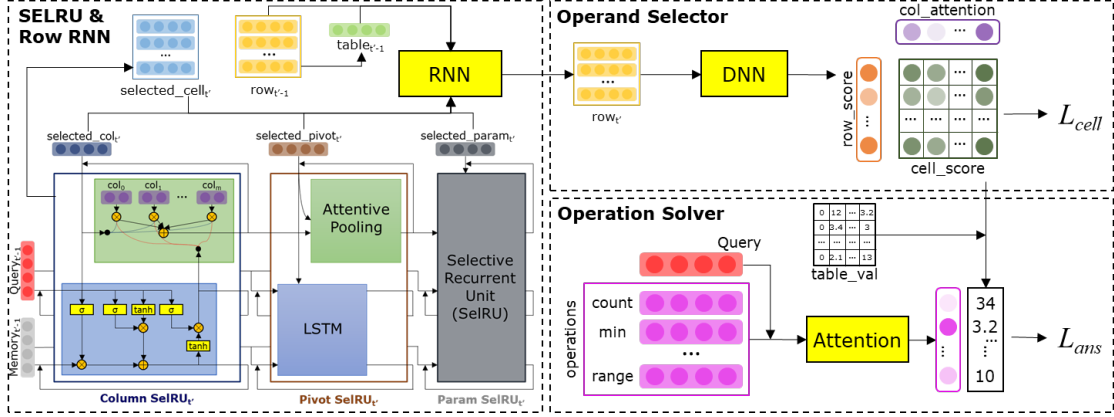
Figure 4: Full architecture of NEURAL OPERATOR at the final timestep t' of the SelRU and Row RNN.

contains the global information regarding result of operation. This is obtained by performing a max pool on all the row vectors r_{j_t}, i.e. $x_t = max\_pool(r_{1_t}, ..., r_{n_t})$, where $j = 1, ..., n$.

Finally, the row vector r_{j_t} is obtained through a non-linear activation function using the six inputs above, as shown in Eq. 5.

$$r_{j_t} = tanh(W_r[\widetilde{f}_t; \widetilde{v}_t; \widetilde{p}_t; \widetilde{c}_{j_t}; r_{j_{t-1}}; x_{t-1}] + b_r) \tag{5}$$

4.4. Operand Selector

After the final timestep t' of Row RNN, we obtain the final row vectors $r_{j_{t'}}$. The operand selector translates these vectors into n scalar row scores $p(r)_{t'}$ with values corresponding to the probability of selecting a row. This is obtained by transforming the concatenated version of the row vectors, using a sigmoid activation function, as shown in Eq. 6.

$$p(r)_{t'} = \sigma(W_a[r_{1_{t'}}; ...; r_{n_{t'}}] + b) \tag{6}$$

Moreover, the operand selector uses the product of both the row scores $p(r_j)_{t'}$ and the column attention scores from Column SelRU $a_f(f_k)_{t'}$ (Eq. 3) to calculate the cell scores $C(j, k)$, which represents the probability of selecting the operands used when calculating the final answer, as shown in Eq. 7.

$$C(j, k) = p(r_j)_{t'} \times a_f(f_k)_{t'} \tag{7}$$

At test time, we filter the cells using a threshold γ, where cells with scores $C(j, k) > \gamma$ are the selected operands.

4.5. Operation Solver

Finally, we use the cell scores $C(j, k)$ to solve all the operations available in the model. In the experiments, we use the seven operations[3], but our model can easily be extended to use other operations. An operation consists of an embedding o and a function op_o that returns an aggregated value y_o as output. The operation solver accepts several operations, compares them to the query q, selects the appropriate operation, and solves the final answer using the operation function.

---

3. We use the operations: argmax, argmin, average, count, print, sum and range

Operation selection is done by computing the relative probabilities of the operations given the query and selecting the operation with the highest probability. The relative probabilities $a_o(o)$ are obtained using attention mechanism (Bahdanau et al., 2014), as shown in Eq. 8.

$$e_o(o, q) = u_o^\top tanh(W_o[o; q] + b_o)$$
$$a_o(o) = \frac{\exp(e_o(o, q))}{\sum_{o'} \exp(e_o(o', q))} \tag{8}$$

At training time, to ensure that the procedure of predicting answer is differentiable, we adopt soft approximations for each of the operations using the operation functions op_o defined in Table 3. All functions accept as inputs all the cell scores $C(j, k)$ and the original cell values $T[j, k]$. Our model predicts the final answer y', as shown in Eq. 9. There are three main challenges here and we solve them through the following: (1) to handle non-numerical cell values in $T[j, k]$, we set them to zero; (2) since `all` operation is a non-numerical operation, we set its function to $-\infty$; and (3) we define reversed

Function	Definition
$all(\cdot)$	$-\infty$
$min(\cdot)$	$max(C(j, k) * T'[j, k])$
$max(\cdot)$	$max(C(j, k) * T[j, k])$
$count(\cdot)$	$\sum_{j,k} C(j, k)$
$sum(\cdot)$	$\sum_{j,k} C(j, k) * T[j, k]$
$mean(\cdot)$	$sum(\cdot)/count(\cdot)$
$range(\cdot)$	$max(\cdot) - min(\cdot)$

Table 3: Operation functions and definitions. The dot $\cdot$ corresponds to the cell values $T[j, k]$ and the cell scores $C(j, k)$ parameters.

cell values as $T'[j, k] = max(T[j, k]) - T[j, k] + \epsilon$ to be used when the operation requires an extremity opposite to the extremity of the cell scores $C(j, k)$, such as the `min` operation. The bias parameter ϵ can be any positive number; we set ours to one. At test time, we use the original functions of the operations to get the final answer.

$$y' = \sum_{o'} a_o(o') \times op_{o'}(\cdot) \tag{9}$$

4.6. Training objective

To train the model, we use two loss functions: L_{cell} for selecting the operands and L_{ans} for solving the final answer. We use the log-loss function over all the cells to calculate L_{cell}, as shown in Eq. 10, where $I[j, k]$ is the labeled operand information. For L_{ans}, we use L2 loss between the predicted and the actual answer y. We notice that L_{ans}, as it is, is significantly larger than L_{cell}. Thus, we use natural logarithm on top of L2 loss to balance the weight intensity of two loss, as shown in Eq. 11. The final loss is the sum of both loss functions, as shown in Eq. 12.

$$L_{cell} = \sum_{j,k} \Big(I[j, k] \log C(j, k) + (1 - I[j, k]) \log \big(1 - C(j, k)\big) \Big) \tag{10}$$

$$L_{ans} = \log \Big(\sum_{i} (y_i' - y_i)^2 \Big) \tag{11}$$

$$L = L_{cell} + L_{ans} \tag{12}$$

	$SoftOpP$	$SoftOpR$	$HardOpA$	$FinalAcc$
SEMPRE	51.3	60.1	58.4	64.4
NEPR	22.8	29.6	22.9	32.4
NEEN	62.7	6.4	8.3	13.3
NEOP (ours)	**92.9**	**94.1**	**72.3**	**80.3**

Table 4: Results of competing models on MLB dataset. Top scores are **bold-faced**.

5. Experiments

5.1. Experimental settings

We use a 600-dimension embedding matrix for our word embedding: 300 dimensions are used for the Xavier-initialized embedding and the other 300 dimensions are used for the binary encoding. For the GRU query encoder, we set the state size to 300 and the timestep to 4, following (Neelakantan et al., 2016). We use dropout (Srivastava et al., 2014) on all non-linear connections with a dropout rate of 0.2. We set the batch size to 50. Training is done via stochastic gradient descent with the Adadelta update rule (Zeiler, 2012), with l_2 constraint (Hinton et al., 2012) of 3. We tune the operand threshold γ using a separate development set and find that anywhere between 0.4 and 0.6 provides similar performance. We set $\gamma = 0.5$ during test phase.

5.2. MLB dataset

Competing models and evaluation We compare our models with three competing TableQA models: one semantic parsing-based model, SEMPRE (Pasupat and Liang, 2015), and two neural-based models, Neural Enquirer NEEN (Yin et al., 2016) and Neural Programmer NEPR (Neelakantan et al., 2016). We use the MLB dataset and it consists of 16k train set, 10k development set and 10k test set. We evaluate the models using four metrics: (1-2) soft operand precision and range when selecting a single cell as an operand, i.e. $SoftOpP = \frac{\#CorrectSelected}{\#TotalSelected}$ and $SoftOpR = \frac{\#CorrectSelected}{\#TotalOperands}$; (3) hard operand accuracy is the percentage of correct prediction of operand *sets*, i.e. $HardOpA = \frac{\#CorrectSets}{\#TotalSets}$; and (4) final answer accuracy is the percentage of correct final answers, i.e. $FinalAcc = \frac{\#CorrectAnswers}{\#TotalAnswers}$.

Results We report the results in Table 4. NEOP performs the best on all metrics, outperforming the second best model, SEMPRE, by at least 26%. NEEN performs the worst in the final accuracy metric, even though it obtains good soft operand precision. This is because NEEN is not able to answer questions that need multiple cells.

Another interesting observation is that NEPR achieves a final accuracy much higher than the hard operand accuracy. This means that even though the models selected the wrong cells, they are magically able to get the answer correctly, a similar anomaly example in Figure 1. This hurts the interpretability of the models and the robustness of their performance. We study these issues more closely in Section 5.5. On the other hand, the final accuracy and the hard operand accuracy of NEOP have minimal difference, thus making the results reliable.

5.3. WIKIOPS dataset

Competing models and evaluation We compare our model with five competing TableQA models: two neural-based models[5], Neural Enquirer NEEN (Yin et al., 2016) and Neural Programmer NEPR (Neelakantan et al., 2016), and three models from the WIK-ISQL dataset: Sequence-to-Tree with Attention SEQ2TREE (Dong and Lapata, 2016), the Augmented Pointer Network AUGPTR (Zhong et al., 2017) and the Sequence-to-SQL SEQ2SQL (Zhong et al., 2017). We use the

Dataset	Model	$FinalAcc$
WIKIOPS	**NeOp (ours)**	**59.5**
	NEPR	52.0
	NEEN	3.4
WIKISQL	SEQ2SQL	59.4[4]
	AUGPTR	53.3
	SEQ2SEQ	35.9

Table 5: Results of models on WIKIOPS and WIK-ISQL. Top scores are **bold-faced**

same train/development/test split of the original WIKISQL dataset. Since some models do not have SQL statement or operand information supervision, we only compare the models using the final answer accuracy, i.e. $FinalAcc = \frac{\#CorrectAnswers}{\#TotalAnswers}$.

Results The results are reported in Table 5. Using the WIKIOPS dataset, NEOP outperforms all other models. The NEPR model performs relatively good compared to its performance in the MLB dataset. We argue this is because of the dataset distribution of WIKIOPS dataset that is skewed towards a lot of `all` operations. Nevertheless, our model still performs better than NEOP by 7.5%. Finally, NEEN performs the worst again in this dataset, since this dataset also contains queries that need multiple cells.

When compared with models using the WIKISQL dataset, NEOP outperforms two baseline models SEQ2TREE and AUGPTR, and performs comparably with the SEQ2SQL model, despite unfair disadvantage of competitors being supervised by SQL statements that are more informative.

5.4. Analysis on operand loss

We argue that one of the reasons why NEOP performs well is that we use the operand information for solving the queries. We perform further analysis on the use of operand information by adding/removing the operand loss on two models: NEEN and NEOP. We add the operand loss to the objective function of NEEN to create an improved version of the model. Furthermore, we remove the operand loss to the objective function of NEOP and obtain a reduced version of the model. We report the results in Table 6 on the MLB dataset. Both models enjoy increases in performance on all metrics. This is especially noticeable in the hard operand accuracy, where NEEN and NEOP receive a 139.8% and 107.8% increase, respectively, and in the final accuracy, where NEEN and NEOP receive a 67.7% and 47.1% increase, respectively.

5.5. Robustness to adversarial examples

From the results in Table 4, we could observe that the models predict the correct answer with inaccurate cell selection. To examine how such "lucky guesses" hurt the robustness and interpretability of

---

4. SEQ2SQL is being outdated with newer results, but our intention of reporting this number is not to compete, considering their unfair advantages of using SQL statements as more informative supervisions. Considering its major limitations as explained in Section 1, we report this rather as an oracle result.
5. We are not able to run SEMPRE due to about 7x higher training complexity than neural-based model, which makes the training infeasible for large sets.

Model	$SoftOpP$		$SoftOpR$	
	w/o L_{cell}	w/ L_{cell}	w/o L_{cell}	w/ L_{cell}
NEEN	62.7	**94.5 (+50.7%)**	6.4	**9.6 (+50.0%)**
NEOP	83.4	**92.9 (+11.4%)**	80.0	**94.1 (+17.6%)**
Model	$HardOpA$		$FinalAcc$	
	w/o L_{cell}	w/ L_{cell}	w/o L_{cell}	w/ L_{cell}
NEEN	8.3	**19.9 (+139.8%)**	13.3	**22.3 (+67.7%)**
NEOP	34.8	**72.3 (+107.8%)**	54.6	**80.3 (+47.1%)**

Table 6: Increase in performance of NEEN and NEOP when L_{cell} is introduced.

Model	$FinalAcc$	V-P	O-P
SEMPRE	64.4	**62.6 (-2.80%)**	**64.2 (-0.36%)**
NEPR	32.4	**31.2 (-3.70%)**	**32.0 (-1.23%)**
NEOP	80.3	**80.1 (-0.25%)**	**80.1 (-0.25%)**

Table 7: Results of models on adversarial examples MLB dataset.

the models, we manually created adversarial examples and evaluate the models with such examples. Specifically, we study two classes of lucky guess scenarios, with data and operation perturbations:

- Value Perturbation (V-P) (perturbing 12% of test set): From the results in Table 4, we find many answers coincide as 0, such that many lucky guesses generated the right answer, merely based on frequency. We change the values that do not affect the answers to reduce its frequency.

- Operation Perturbation (O-P) (perturbing 70% of test set): Some operation, such as count, can be answered correctly, regardless of the columns selected. We change the operation (e.g. replace `argmax` to `sum`) and answer accordingly.

Results Using the perturbed dataset, we compare our model NEOP with semantic parsing-based model, SEMPRE (Pasupat and Liang, 2015) and neural-based models, NEPR (Neelakantan et al., 2016). For V-P, NEOP stays robust, only with a slight degradation (0.25%) from its original accuracy. However, SEMPRE and NEOP show more significant decreases of 2.80% and 3.70% from their original accuracy, respectively. For O-P, NEPR is still the most sensitive to adversarial examples with the decrease of 1.23%. NEOP and SEMPRE responds better to adversarial operation changes.

5.6. Model interpretability

Through the Column, Pivot, Param SelRUs, and the Operation Solver of NEOP, it is easy to interpret the model in a step-by-step manner. First, at each timestep, we obtain the attention weights of the Attentive Pooling component of each SelRU (e.g. Eq. 3 of Column SelRU). The one with the highest weight is the selected field/word. Second, after the final timestep, we obtain the attention weights of the operation solver (i.e. $a_o(o)$ in Eq. 8). The one with the highest weight is the selected operation.

In Figure 5, we show an example query answered by our model. At the first timestep, the selected column PB, the selected pivot is the word "*equal*", and the selected parameter is the word
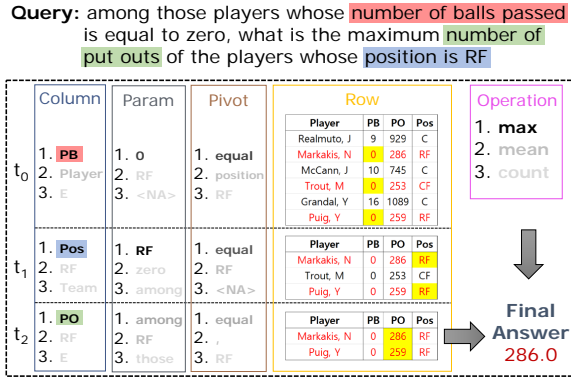
Figure 5: An example query ran through NEOP. The columns, pivots, parameters, and the operation are selected using the weights from our model. We show three instances with the highest weights at each timestep. The last timestep is omitted. The rows in the right side are manually interpreted. The logical form of the natural language query shown is `max(PO | PB=0, Pos='RF')`. Abbreviation glossary: PO - put outs, PB - passed balls, Pos - position.

"*0*". These elements correspond to the first condition of the query, i.e. `PB=0`. At the second timestep, a similar step-by-step selection of column, pivot, and parameter is carried out to create the second condition of the query, i.e. `Pos='RF'`. Finally, at the last timestep, although the words "*equal*" and "*among*" are selected as pivot and parameter, the weights are not high enough. Hence, this signals the end of the iteration of the SelRUs. A possible interpretation is shown in the yellow box at the right of the same figure, where selected rows are colored red and conditioned cells are highlighted in yellow. Finally, the operation solver selects `max` operation, thus the final answer is `max(286, 259) = 286`.

6. Conclusion

In this paper, we propose the usage of operand information to improve the performance of TableQA models. To do this, we create two new datasets called MLB dataset and WIKIOPS dataset, both of which improves on previous datasets in terms of attention supervision. Furthermore, we develop NEURAL OPERATOR, a neural-based TableQA model which has improved interpretability thanks to the its layer-wise architecture and the use of operand information. We test our hypothesis on both proposed datasets, and our model significantly outperforms previous models. Finally, we show multiple analysis on how the operand information increases the performance, reliability, and interpretability of models.

References

Reinald Kim Amplayo, Jihyeok Kim, Sua Sung, and Seung won Hwang. Cold-start aware user and product attention for sentiment classification. In *ACL*, 2018a.

Reinald Kim Amplayo, Kyungjae Lee, Jinyeong Yeo, and Seung won Hwang. Translations as additional contexts for sentence classification. In *IJCAI*, 2018b.

Reinald Kim Amplayo, Seonjae Lim, and Seung won Hwang. Entity commonsense representation for neural abstractive summarization. In *NAACL*, 2018c.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, 2013.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Wanyun Cui, Yanghua Xiao, Haixun Wang, Yangqiu Song, Seung-won Hwang, and Wei Wang. Kbqa: learning question answering over qa corpora and knowledge bases. *Proceedings of the VLDB Endowment*, 10(5):565–576, 2017.

Abhishek Das, Harsh Agrawal, Larry Zitnick, Devi Parikh, and Dhruv Batra. Human attention in visual question answering: Do humans and deep networks look at the same regions? *Computer Vision and Image Understanding*, 163:90–100, 2017.

Li Dong and Mirella Lapata. Language to logical form with neural attention. *arXiv preprint arXiv:1601.01280*, 2016.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

Omer Goldman, Veronica Latcinnik, Udi Naveh, Amir Globerson, and Jonathan Berant. Weakly-supervised semantic parsing with abstract examples. *arXiv preprint arXiv:1711.05240*, 2017.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328*, 2017.

Jayant Krishnamurthy and Thomas Kollar. Jointly learning to parse and perceive: Connecting natural language to the physical world. *Transactions of the Association for Computational Linguistics*, 1:193–206, 2013.

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. Scaling semantic parsers with on-the-fly ontology matching. In *2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013*. Association for Computational Linguistics (ACL), 2013.

Chenxi Liu, Junhua Mao, Fei Sha, and Alan L Yuille. Attention correctness in neural image captioning. In *AAAI*, pages 4176–4182, 2017a.

Xiaodong Liu, Yelong Shen, Kevin Duh, and Jianfeng Gao. Stochastic answer networks for machine reading comprehension. *arXiv preprint arXiv:1712.03556*, 2017b.

Haitao Mi, Zhiguo Wang, and Abe Ittycheriah. Supervised attentions for neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2283–2288, 2016.

Sewon Min, Minjoon Seo, and Hannaneh Hajishirzi. Question answering through transfer learning from large fine-grained supervision data. *arXiv preprint arXiv:1702.02171*, 2017.

Arvind Neelakantan, Quoc V Le, and Ilya Sutskever. Neural programmer: Inducing latent programs with gradient descent. *arXiv preprint arXiv:1511.04834*, 2015.

Arvind Neelakantan, Quoc V Le, Martin Abadi, Andrew McCallum, and Dario Amodei. Learning a natural language interface with neural programmer. *arXiv preprint arXiv:1611.08945*, 2016.

Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*, 2015.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Yushi Wang, Jonathan Berant, Percy Liang, et al. Building a semantic parser overnight. In *ACL (1)*, pages 1332–1342, 2015.

Yi Yang, Wen-tau Yih, and Christopher Meek. Wikiqa: A challenge dataset for open-domain question answering. In *EMNLP*, pages 2013–2018, 2015.

Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. Neural enquirer: learning to query tables in natural language. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 2308–2314. AAAI Press, 2016.

Youngjae Yu, Jongwook Choi, Yeonhwa Kim, Kyung Yoo, Sang-Hun Lee, and Gunhee Kim. Supervising neural attention models for video captioning by human gaze data. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017). Honolulu, Hawaii*, pages 2680–29, 2017.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Luke S Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *In Proceedings of the 21st Conference on Uncertainty in AI*. Citeseer, 2005.

Rui Zhao and Kezhi Mao. Topic-aware deep compositional models for sentence classification. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(2):248–260, 2017.

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