(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
22 November 2018 (22.11.2018)

WIPO | PCT

(10) International Publication Number
WO 2018/213530 A2

(51) International Patent Classification:
Not classified

(21) International Application Number:
PCT/US2018/033099

(22) International Filing Date:
17 May 2018 (17.05.2018)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/508,367 18 May 2017 (18.05.2017) US
15/885,613 31 January 2018 (31.01.2018) US

(71) Applicant: **SALESFORCE.COM, INC** [US/US]; The Landmark @ One Market Street, Suite 300, San Francisco, CA 94105 (US).

(72) Inventors; and

(71) Applicants: **ZHONG, Victor** [—/US]; salesforce.com, Inc., The Landmark @ One Market Street, Suite 300, San Francisco, CA 94105 (US). **XIONG, Caiming** [—/US]; salesfroce.com, Inc., The Landmark @ One Market Street, Suite 300, San Francis, CA 94105 (US). **SOCHER, Richard** [—/US]; salesforce.com, Inc., The Landmark @ One Market Street, Suite 300, San Francisco, CA 94105 (US).

(74) Agent: **PANWAR, Rajendra .** et al.; Fenwick West LLP, 801 California Street, Mountain View, CA 94041 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

(54) Title: NEURAL NETWORK BASED TRANSLATION OF NATURAL LANGUAGE QUERIES TO DATABASE QUERIES

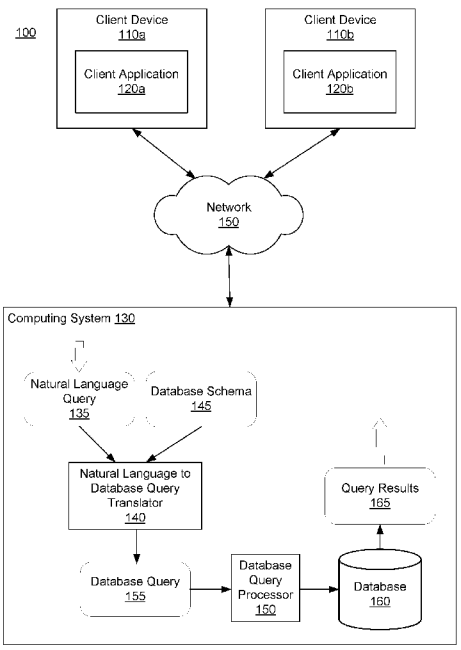

FIG. 1

(57) Abstract: A computing system uses neural networks to translate natural language queries to database queries. The computing system uses a plurality of machine learning based models, each machine learning model for generating a portion of the database query. The machine learning models use an input representation generated based on terms of the input natural language query, a set of columns of the database schema, and the vocabulary of a database query language, for example, structured query language SQL. The plurality of machine learning based models may include an aggregation classifier model for determining an aggregation operator in the database query, a result column predictor model for determining the result columns of the database query, and a condition clause predictor model for determining the condition clause of the database query. The condition clause predictor is based on reinforcement learning.

WO 2018/213530 A2

[Continued on next page]

Published:

— *without international search report and to be republished upon receipt of that report (Rule 48.2(g))*

NEURAL NETWORK BASED TRANSLATION OF NATURAL LANGUAGE QUERIES TO DATABASE QUERIES

BACKGROUND

FIELD OF ART

[0001] The disclosure relates in general to automatic generation of database queries, and more specifically to neural network based models for translating natural language queries to database queries.

DESCRIPTION OF THE RELATED ART

[0002] A significant amount of data available in the world is stored in relational databases. Relational databases provide the foundation of applications such as medical records, financial markets, customer relations management, and so on. However, accessing information in relational databases requires an understanding of database query languages such as the structured query language (SQL). Although database query languages such as SQL are powerful in terms of allowing a user to specify requests for data from a relational database, they are difficult to learn. To be able to write database queries effectively using database query languages requires expertise in databases and strong technical knowledge.

[0003] Some systems support natural language for accessing data stored in the system. Natural language queries provide ease of expression since people do not require training to use natural language. However, these systems do not provide the expressive power of the database query languages such as SQL. For example, a natural language query may be interpreted in multiple ways and the corresponding execution of the natural language query to access data stored in a relational database may be inefficient and may not retrieve the exact information that was requested. Accordingly, conventional techniques for accessing data stored in relational databases using either natural language queries or database queries have drawbacks since they either provide ease of expression or the power of expression, but not both.

BRIEF DESCRIPTION OF DRAWINGS

[0004] The disclosed embodiments have other advantages and features which will be more readily apparent from the detailed description, the appended claims, and the accompanying figures (or drawings). A brief introduction of the figures is below.

[0005] Figure (FIG.) 1 is a high-level block diagram illustrating the overall system environment for translating natural language queries to database queries, in accordance with an embodiment.

[0006] FIG. 2 illustrates the system architecture of the computing system for translating natural language queries to database queries, in accordance with an embodiment.

[0007] FIG. 3 illustrates the details of the processing performed by the natural language to database query translator, according to an embodiment.

[0008] FIG. 4 illustrates the overall process for translating natural language queries to database queries, according to an embodiment

[0009] FIG. 5 illustrates the process of the aggregation classifier for determining the aggregation operator of the output database query based on a natural language query, according to an embodiment.

[0010] FIG. 6 illustrates the process of the result column predictor for determining the columns of the SELECT clause of the output database query based on a natural language query, according to an embodiment.

[0011] FIG. 7 illustrates the process of training the condition clause predictor for determining the condition clause of the output database query, according to an embodiment.

[0012] FIG. 8 is a high-level block diagram illustrating an example computer for implementing the client device and/or the computing system of FIG. 1.

[0013] The Figures (FIGS.) and the following description describe certain embodiments by way of illustration only. One skilled in the art will readily recognize from the following description that alternative embodiments of the structures and methods illustrated herein may be employed without departing from the principles described herein. Reference will now be made in detail to several embodiments, examples of which are illustrated in the accompanying figures.

DETAILED DESCRIPTION

[0014] A computing system uses deep neural networks for translating natural language queries to corresponding database queries, for example, queries specified using structured query language (SQL). Embodiments use the structure of SQL queries to greatly reduce the output space of generated queries. The computing system uses deep neural networks to translate the natural language query to a database query.

[0015] In an embodiment, the computing system uses a plurality of machine learning based models, for example, neural network based models to generate different portions of the output database query. For example, the computing system may use an aggregation classifier model for determining an aggregation operator in the database query, a result column predictor model for determining the result columns of the database query, and a condition clause predictor model for determining the condition clause of the database query. In an embodiment, the aggregation classifier model and result column predictor model comprise multi-layer perceptrons. The condition clause predictor model uses policy-based reinforcement learning (RL) to generate the condition clause of the database query. This is so because the condition clause is unordered in nature and multiple representations of the condition clause may provide the same output result for the database query. Therefore the condition clause unsuitable for optimization using cross entropy loss. The deep neural network is trained using a mixed objective that combines cross entropy losses and RL rewards.

[0016] As an example, a database may store a table CFLDraft with columns Pick_number, CFL_Team, Player, Position, and College. The table may store following example rows.

Pick_number	CFL_Team	Player	Position	College
27	Hamilton Tiger-Cats	Connor Healy	DB	Wilfrid Laurier
28	Calgary Stampeders	Anthony Forgone	OL	York
29	Ottawa Renegades	L. P. Ladouceur	DT	California
30	Toronto Argonauts	Frank Hoffman	DL	York
…	…	…	…	…

[0017] The system receives a natural language query, for example, “How many CFL teams are from York College?” The system processes the received natural language query in connection with the database schema comprising the table CFLDraft to generate a database query using SQL language “SELECT COUNT(CFL_Team) FROM CFLDraft WHERE

College = "York"". The system executes the database query using the database schema. Two rows of the table CLFDraft match the WHERE clause of the database query since they have the college "York". As a result the system returns the result 2.

OVERALL SYSTEM ENVIRONMENT

[0018] FIG. 1 is a high-level block diagram illustrating the overall system environment for translating natural language queries to database queries, in accordance with an embodiment. The system environment 100 includes one or more client devices 110 connected by a network 150 to a computing system 130. The computing system 130 may be an online system but may also work offline, for example, by performing batch processing for translating each of a set of natural language queries to database queries.

[0019] Here only two client devices 110a, 110b are illustrated but there may be multiple instances of each of these entities. For example, there may be several computing systems 130 and dozens or hundreds of client devices 110 in communication with each computing system 130. The figures use like reference numerals to identify like elements. A letter after a reference numeral, such as "110a," indicates that the text refers specifically to the element having that particular reference numeral. A reference numeral in the text without a following letter, such as "110," refers to any or all of the elements in the figures bearing that reference numeral.

[0020] The client devices 110 are computing devices such as smartphones with an operating system such as ANDROID® or APPLE® IOS®, tablet computers, laptop computers, desktop computers, electronic stereos in automobiles or other vehicles, or any other type of network-enabled device on which digital content may be listened to or otherwise experienced. Typical client devices 110 include the hardware and software needed to connect to the network 150 (e.g., via Wifi and/or 4G or other wireless telecommunication standards).

[0021] The client device 110 includes a client application 120 that allows a user of the client device 110 to interact with the computing system 130. For example, the client application 120 may be a user interface that allows users to input natural language queries that are sent to the computing system 130. The client application 120 receives results from the computing system 130 and presents them to the user via the user interface. In an embodiment, the client application 120 is a browser that allows users of client devices 110 to interact with a web server executing on the computing system 130.

[0022] The computing system 130 includes software for performing a group of coordinated functions or tasks. The software may allow users of the computing system 130 to perform certain tasks or activities of interest, or may include system software (e.g., operating systems) that provide certain functionalities and services to other software. The computing system 130 receives requests from client devices 110 and executes computer programs associated with the received requests. As an example, the computing system 130 may execute computer programs responsive to a request from a client device 110 to translate natural language queries to database queries. Software executing on the computing system 130 can include a complex collection of computer programs, libraries, and related data that are written in a collaborative manner, in which multiple parties or teams are responsible for managing different components of the software.

[0023] In an embodiment, the computing system 130 receives a natural language query 135 from a client device 110. The natural language query 130 may be provided by a user via the client application 120 executing on the computing system 130. The computing system 130 stores a database schema 145 that defines the structure of data stored in a database. For example, the database schema 145 may identify various tables stored in the database, the columns of each table, the relations between tables such as foreign key relations, any constraints associated with the tables, and so on.

[0024] The natural language to database query translator 140 receives the natural language query 135 and the database schema 145 as input and generates a database query 155 that is equivalent to the input natural language query 135. The generated database query 155 conforms to the database schema 145. The generated database query 155 is received by a database query processor 150 that processes the database query 155 using the data stored in the database 160. The database query processor 150 generates the query results 165 by processing the database query 155. The computing system 130 provides the generated query results 165 to the client application 120 running on the client device 110 that sent the natural language query 135.

[0025] In an embodiment, the natural language to database query translator 140 performs a sequence to sequence translation. Conventional neural network based sequence to sequence translator search in a very large space. In contrast, embodiments exploit the structure inherent in a database query language to reduce the search space. In particular, the system limits the output space of the generated sequence based on the union of the table schema, the input question, and SQL key words. In one embodiment, the natural language to database

query translator 140 uses a deep neural network that is a pointer network with augmented inputs.

[0026] The network 150 provides a communication infrastructure between the client devices 110 and the record management system 130. The network 150 is typically the Internet, but may be any network, including but not limited to a Local Area Network (LAN), a Metropolitan Area Network (MAN), a Wide Area Network (WAN), a mobile wired or wireless network, a private network, or a virtual private network. Portions of the network 150 may be provided by links using communications technologies including WiFi based on the IEEE 802.11 standard, the BLUETOOTH short range standard, and the Wireless Universal Serial Bus (USB) standard.

SYSTEM ARCHITECTURE

[0027] FIG. 2 illustrates the system architecture of the computing system for translating natural language queries to database queries, in accordance with an embodiment. The computing system 130 comprises an input encoding module 210, a training module 240, a natural language to database query translator 140, a query synthesis module 220, a query execution engine 230, a training data store 215, and a database 160. Conventional components such as network interfaces, security functions, load balancers, failover servers, management and network operation consoles, and the like are not shown so as to not obscure the details of the system architecture.

[0028] The input preprocessing module 210 preprocesses the input data for providing as input to the natural language to database query translator 140. In an embodiment, the input preprocessing module 210 generates 420 a sequence of tokens by concatenating column names from the database schema, the input natural language query, and the vocabulary of the database query language, for example, SQL. The input preprocessing module 210 generates one or more input representations for providing to the various models that generate the various parts of the output database query.

[0029] The natural language to database query translator 140 processes an input natural language query for generating the database query corresponding to the natural language query. In an embodiment, the natural language to database query translator 140 includes other components, for example, an aggregation classifier 260, a result column predictor 270, and a condition clause predictor 280, further described herein, in connection with FIG. 3.

[0030] The natural language to database query translator 140 generates different components of the database query using different neural networks. In an embodiment, the natural language to database query translator 140 uses a different neural network to generate the components of a database query including the select columns, an aggregation operator, and a where clause.

[0031] The training module 240 uses historical data stored in training data store 215 to train the neural networks in the natural language to database query translator 140. In an embodiment, the training module 240 trains the aggregation classifier 260 and the result column predictor 270 using cross entropy loss, but trains the condition clause predictor 280 using policy gradient reinforcement learning in order to address the unordered nature of query conditions. Utilizing the structure of a SQL query allows the natural language to database query translator 140 to reduce the output space of database queries. This leads to a significantly higher performance compared to other techniques that do not exploit the query structure.

[0032] The query synthesis module 220 receives various components of the database query as generated by the natural language to database query translator 140 and combines them to obtain a database query. The query execution module 230 executes the database query provided by the query synthesis module 220 using the data stored in the database 160. The computing system 130 returns the result of execution of the query to the requestor of the result, for example, a client application 120 executing on a client device 110.

[0033] FIG. 3 illustrates the details of the processing performed by the natural language to database query translator 140, according to an embodiment. As shown in FIG. 3. The inputs to the natural language to database query translator 140 include the natural language query 320 and the database schema 320. In the example illustrated above based on CFLDraft table, the natural language query 320 is "How many CFL teams are from York College?" and the database schema 320 comprises the various columns including columns Pick_number, CFL_Team, Player, Position, and College. The example output database query is "SELECT COUNT(CFL_Team) FROM CFLDraft WHERE College = "York"".

[0034] The input preprocessing module 210 generates one or more input representations and provides an input representation to each component of the natural language to database query translator 140 including the aggregation classifier 260, the result column predictor 270, and the condition clause predictor 280. Each of the aggregation classifier 260, the result

column predictor 270, and the condition clause predictor 280 generates a part of the output database query.

[0035] The result column predictor 270 generates the result columns, for example, the columns specified in the SELECT clause 310 of the output database query expressed using SQL. An example of a result column is the column CFL_Team in the example output database query. In an embodiment, the result column predictor 270 is a pointer network that receives an encoding of a sequence of columns as input and points to a column in the sequence of columns corresponding to a SELECT column.

[0036] The condition clause predictor 280 generates the WHERE clause 320 of the output database query that specifies the condition used to filter the output rows of the output database query. In the above example, the WHERE clause "College = "York"" is the condition clause in the output database query.

[0037] The aggregation classifier 260 generates an aggregation operator 330 in the output database query if any, for example, the COUNT operator in the example output database query. The aggregation operators produce a summary of the rows selected by the SQL. Examples of aggregation operators that may be generated by the aggregation classifier 260 include maximum (MAX), minimum (MIN), average (AVG), sum (SUM), and so on. The aggregation classifier 260 may generate a NULL aggregation operator if there is no aggregation operator in the output query.

[0038] The various components of the output database query including the SELECT clause 310, the WHERE clause 320, and the aggregation operator 330 are provided as input to the query synthesis module 270. The query synthesis module 270 combines the individual components of the output database query to generates the complete output database query 340.

OVERALL PROCESS

[0039] FIGs. 4-7 illustrate various process for translating natural language queries to database queries. Those of skill in the art will recognize that other embodiments can perform the steps of FIGs. 4-7 in different orders than those shown in the flowcharts. Moreover, other embodiments can include different and/or additional steps than the ones described herein. Steps indicated as being performed by certain modules may be performed by other modules.

[0040] FIG. 4 illustrates the overall process for translating natural language queries to database queries, according to an embodiment. The natural language to database query

translator 140 receives 410 an input natural language query. The input preprocessing module 210 generates 420 a sequence of tokens by concatenating column names from the database schema, the input natural language query, and the vocabulary of the database query language, for example, various keywords of the SQL language such as SELECT, FROM, WHERE, and so on. For example, equation (1) shows the sequence of tokens comprising the columns names x_i^c , the terms x^s representing the SQL vocabulary, and the terms x^q representing the input natural language query.

$$x = [< \text{col} >; x_1^c; x_2^c; \ldots; x_N^c; < \text{sql} >; x^s; < \text{question} >; x^q] \quad (1)$$

[0041] In equation (1), concatenation between the sequences a and b is represented as [a; b]. Furthermore, the combined sequence x includes sentinel tokens between neighboring sequences to demarcate the boundaries. For example, token <col> identifies columns names, token <sql> identifies terms representing SQL vocabulary, and token <question> identifies terms of the input natural language query.

[0042] The input preprocessing module 210 generates 430 an input representation of the sequence of tokens. In an embodiment, the input preprocessing module 210 generates multiple input representations, one for each of the plurality of models.

[0043] The natural language to database query translator 140 accesses a plurality of neural machine learning models, each model configured to generate a portion of the output database query. In an embodiment, the natural language to database query translator 140 loads the plurality of trained neural network based models from a storage device to memory. The natural language to database query translator 140 provides 450 an input representation to each of the plurality of machine learning based models. Each of the plurality of machine learning based models generates a portion of the database query.

[0044] In some embodiments, the input preprocessing module 210 generates multiple input representations, the natural language to database query translator 140 may provide a different input representation to each machine learning based model. Each machine learning based model generates a portion of the database query and provides it to the query synthesis module 270. The query synthesis module 270 combines 460 the plurality of portions of the database query to generate the full database query. The query execution engine 230 executes 470 the database query to generate a results set.

AGGREGATION CLASSIFIER

[0045] FIG. 5 illustrates the process of the aggregation classifier for determining the aggregation operator of the output database query based on a natural language query, according to an embodiment. The aggregation classifier 260 determines the aggregation operator of the output database query based on the type of question specified in the input natural language query. For example, the aggregation classifier 260 may map an input question comprising the string "how many" to the aggregation operator COUNT, the aggregation classifier 260 may map an input question comprising "what is the highest" to the aggregation operator maximum, the aggregation classifier 260 may map an input question comprising "what is the smallest" to the aggregation operator minimum, and so on.

[0046] The aggregation classifier 260 determines 510 an input representation of the input sequence of tokens. The aggregation classifier 260 computes a scalar attention score $\alpha_t^{inp} = W^{inp} * h_t^{enc}$ for each t^{th} token in the input sequence. Accordingly, the aggregation classifier 260 generates a vector of scores $\alpha^{inp} = [\alpha_1^{inp}, \alpha_2^{inp}, \ldots]$. The aggregation classifier 260 normalizes the vector of scores α^{inp}, to produce a distribution over the input encodings by applying the softmax function to the α^{inp} vector to determine $\beta^{inp} = \text{softmax}(\alpha^{inp})$. The aggregation classifier 260 produces a distribution over the input encodings. The aggregation classifier 260 determines 510 the input representation κ^{agg} as the sum over the input encodings h^{enc} weighted by the normalized scores β^{inp} as shown by the following equation.

$$\kappa^{agg} = \sum_t \beta_t^{inp} h_t^{enc} \quad (2)$$

[0047] The aggregation classifier 260 comprises a multi-layer perceptron applied to the generated input representation κ^{agg} to generate scores α^{agg} corresponding to various aggregation operations, for example, COUNT, MIN, MAX, the NULL operator indicating no aggregation, and so on. The aggregation classifier 260 identifies 530 the aggregation operation for the database query based on the generated scores.

[0048] In an embodiment, the aggregation classifier 260 determines α^{agg} using the following equation.

$$\alpha^{agg} = W^{agg} \tanh\left(V^{agg}\kappa^{agg} + b^{agg}\right) + c^{agg} \quad (3)$$

[0049] The terms W^{agg}, V^{agg}, b^{agg}, and c^{agg} denote weights corresponding to the multi-layer perceptron. The aggregation classifier 260 applies the softmax function to obtain the

distribution over the set of possible aggregation operations $\beta^{agg} = softmax(\alpha^{agg})$. The aggregation classifier is trained based on the cross entropy loss L^{agg}.

RESULT COLUMN PREDICTOR

[0050] The SELECT clause is also referred to as the selection columns or the result columns. The result column predictor 270 determines the selection columns based on the table columns in the database schema as well as the natural language query. For example, given a natural language query "How many CFL teams ..." the result column predictor 270 determines that the selection columns include CFL_Teams column from the CFLDraft table. Accordingly, the result column predictor 270 solves the problem of SELECT column prediction as a matching problem. In an embodiment, the result column predictor 270 uses a pointer to identify a SELECT column. Given the list of column representations and a representation of the natural language query, the result column predictor 270 selects the column that best matches the natural language query.

[0051] FIG. 6 illustrates the process performed by the result column predictor for determining the columns of the SELECT clause of the output database query based on a natural language query, according to an embodiment. The result column predictor 270 uses an input representation for the columns by encoding 610 each column name with an LSTM (long short term memory network). The input preprocessing module 210 generates 620 an input representation of a particular column j, e_c^j, using the following equation.

$$h_{j,t}^c = \text{LSTM}\left(\text{emb}\left(x_{j,t}^c\right), h_{j,t-1}^c\right) \qquad e_j^c = h_{j,Tj}^c \tag{4}$$

[0052] In this equation, $h^c_{j,t}$ denotes the t^{th} encoder state of the j^{th} column and emb is a function that returns an embedding. The input preprocessing module 210 takes the last encoder state to be e^c_j, column j's representation.

[0053] The input preprocessing module 210 constructs a representation for the natural language query κ^{sel} using an architecture similar to that described above for κ^{agg}. The result column predictor 270 applies 630 a multi-layer perceptron over the column representations, conditioned on the input representation, to compute the score for each column j using the following equation.

$$\alpha_j^{\text{sel}} = W^{\text{sel}} \tanh\left(V^{\text{sel}}\kappa^{\text{sel}} + V^c e_j^c\right) \tag{5}$$

[0054] In this equation W^{sel}, V^{sel}, and V^c are weights of the multi-layer perceptron. The result column predictor 270 normalizes 640 the scores with a softmax function to produce a

distribution over the possible SELECT columns $\beta^{sel} = softmax(\alpha^{sel})$. In the above example of the CFLDraft table, the distribution is over the columns Pick_number, CFL_Team, Player, Position, and College. The result column predictor 270 selects 650 the result columns of the output database query based on the normalized scores. The aggregation classifier is trained based on the cross entropy loss L^{sel}.

CONDITION CLAUSE PREDICTOR

[0055] In an embodiment, the condition clause predictor generates the WHERE clause using a pointer decoder. However, the WHERE conditions of a query can be swapped and the query would yield the same result. For example, given a natural language query "which males are older than 18", the output database query can be either "SELECT name FROM insurance WHERE age > 18 AND gender = "male"" or "SELECT name FROM insurance WHERE gender = "male" AND age > 18". Both database queries obtain the correct execution result even though the two database queries do not match based on a string match between the two query strings. If the first database query is provided as the ground truth while training the neural network and cross entropy loss is used to supervise the training, the second database query will be wrongly penalized since it does not match the first database query based on a string match. Therefore embodiments apply reinforcement learning to learn a policy to directly optimize the expected correctness of the execution result of the database query.

[0056] FIG. 7 illustrates the process of training the condition clause predictor for determining the condition clause of the output database query, according to an embodiment. The condition clause predictor 280 receives as input a natural language query 710 and a database schema 720 to generate the database query 730. The condition clause predictor 280 sends the database query for execution using the database 160 to obtain a reward metric. The query execution engine 230 executes the generated database query 730 to obtain the predicted query results 750. The computing system 130 stores the ground truth query results 750 in training data store 215. The condition clause predictor 280 compares the predicted query results 750 with the ground truth query results 750 to determine the reward 750. The reward is provided as input to the condition clause predictor 280 as feedback for training the condition clause predictor 280.

[0057] The sequence of tokens generated by the condition clause predictor 280 in the WHERE clause is denoted by $y = [y^1, y^2, \ldots, y^T]$. Let $q(y)$ denote the query generated by the model and q_g denote the ground truth database query corresponding to the natural language

query. The condition clause predictor 280 uses the following equation as the reward metric $R(q(y), q_g)$.

$$R(q(y), q_g = \begin{cases} -2, & \text{if } q(y) \text{ is not a valid SQL query} \\ -1, & \text{if } q(y) \text{ is a valid SQL query and executes to an incorrect result} \\ +1, & \text{if } q(y) \text{ is a valid SQL query and executes to the correct result} \end{cases} \quad (6)$$

[0058] Accordingly, the condition clause predictor 280 assigns a positive reward if the result of execution of the generated database query matches the expected results provided as ground truth. The condition clause predictor 280 assigns a negative reward if the result of execution of the generated database query fails to match the expected results provided as ground truth or if the generated database query is not a valid database query.

[0059] The condition clause predictor 280 determines the loss L^{whe} as the negative expected reward over possible WHERE clauses. The training module trains the condition clause predictor 280 using gradient descent to minimize the objective function $L = L^{agg} + L^{sel} + L^{whe}$. Accordingly, the condition clause predictor 280 determines a total gradient as the weighted sum of the gradients from the cross entropy loss in predicting the SELECT column, from the cross entropy loss in predicting the aggregation operation, and from policy learning for the condition clause.

[0060] The incorporation of structure in the natural language to database query translator 140 reduces invalid database queries that may be generated. A large quantity of invalid queries result from column names – the generated query refers to selection columns that are not present in the table. This is particularly helpful when the column name contain many tokens, such as "Miles (km)", which has 4 tokens. Introducing a classifier for the aggregation also reduces the error rate. Use of the aggregation classifier improves the precision and recall for predicting the COUNT operator. Use of representation learning for generating condition clause results in generation of higher quality WHERE clause that may be ordered differently than ground truth. Training with policy-based representation learning results in correct results even if the order of conditions is differs from the ground truth query.

COMPUTER ARCHITECTURE

[0061] FIG. 8 is a high-level block diagram illustrating an example computer for implementing the client device and/or the computing system of FIG. 1. The computer 800 includes at least one processor 802 coupled to a chipset 804. The chipset 804 includes a memory controller hub 820 and an input/output (I/O) controller hub 822. A memory 806 and a graphics adapter 812 are coupled to the memory controller hub 820, and a display 818 is

coupled to the graphics adapter 812. A storage device 808, an input device 814, and network adapter 816 are coupled to the I/O controller hub 822. Other embodiments of the computer 800 have different architectures.

[0062] The storage device 808 is a non-transitory computer-readable storage medium such as a hard drive, compact disk read-only memory (CD-ROM), DVD, or a solid-state memory device. The memory 806 holds instructions and data used by the processor 802. The input interface 814 is a touch-screen interface, a mouse, track ball, or other type of pointing device, a keyboard, or some combination thereof, and is used to input data into the computer 800. In some embodiments, the computer 800 may be configured to receive input (e.g., commands) from the input interface 814 via gestures from the user. The graphics adapter 812 displays images and other information on the display 818. The network adapter 816 couples the computer 800 to one or more computer networks.

[0063] The computer 800 is adapted to execute computer program modules for providing functionality described herein. As used herein, the term "module" refers to computer program logic used to provide the specified functionality. Thus, a module can be implemented in hardware, firmware, and/or software. In one embodiment, program modules are stored on the storage device 808, loaded into the memory 806, and executed by the processor 802.

[0064] The types of computers 800 used by the entities of FIG. 1 can vary depending upon the embodiment and the processing power required by the entity. The computers 800 can lack some of the components described above, such as graphics adapters 812, and displays 818. For example, the computing system 130 can be formed of multiple blade servers communicating through a network such as in a server farm.

[0065] Realizations of the subject matter of the application may especially be the following examples 1 to 21.

1. A method comprising:

receiving an input natural language query based on data stored using a database schema;

generating a sequence of tokens from a plurality of terms comprising:

terms of the input natural language query,

a set of columns of the database schema, and

vocabulary of a database query language;

generating one or more input representations, each input representation obtained by encoding the sequence of tokens;

accessing a plurality of machine learning based models, each model configured to predict a portion of a database query corresponding to the input natural language query;

for each of the plurality of models, executing the model based on an input representation to generate a portion of the database query;

combining the generated portions of the database query to obtain the database query; and

executing the database query to obtain a result set.

2. The method of example 1, wherein the input natural language query is received from a client device and the method further comprises a step of sending the result set to the client device.

3. The method of example 1, wherein the plurality of models comprise an aggregation classifier model for determining an aggregation operator in the database query, wherein the aggregation classifier model comprises a multi-layer perceptron.

4. The method of example 1, wherein the plurality of models comprise a result column predictor model for determining the result columns of the database query, wherein the result column predictor model comprises a multi-layer perceptron.

5. The method of example 1, wherein the plurality of models comprise a condition clause predictor model for determining the condition clause of the database query, wherein the condition clause predictor model is based on reinforcement learning.

6. The method of example 5, further comprising:

receiving a result set based on a ground truth database query;

determining reward values based on a comparison of the result obtained from the generated query and the result obtained from the ground truth query; and

adjusting weights of the condition clause predictor model based on the reward values.

7. The method of example 5, further comprising:

determining column encodings corresponding to each token of the sequence;

determining a vector comprising scalar attention scores for each token of the input sequence;

normalizing the vector using a softmax function; and

determining the input representation as the sum of the column encodings weighted by the corresponding normalized score.

8. The method of example 1, further comprising:

training the plurality of models using gradient descent to minimize an objective function representing a loss based on the result of each of the plurality of models.

9. The method of one of the previous examples, wherein generating the one or more input representations comprises computing an input representation κ^{agg} by:

computing a scalar attention score, $\alpha^{in}{}_t = W^{inp} h_t^{enc}$, for each tth token in sequence of tokens, wherein h_t^{enc} is a state of an encoder corresponding to a tth word in the input sequence,

normalizing the vector of scores $\alpha^{in} = [\alpha_1^{inp}, \alpha_2^{inp}, \ldots]$ to produce a distribution over the tokens in the sequence of tokens,

obtaining the input representation κ^{agg} as

$$\kappa^{\text{agg}} = \sum_t \beta_t^{\text{inp}} h_t^{\text{enc}}$$

wherein $\beta^{\text{agg}} = \text{softmax}(\alpha^{\text{agg}})$, and $\alpha^{\text{agg}} = W^{\text{agg}} \tanh(V^{\text{agg}} \kappa^{\text{agg}} + b^{\text{agg}}) + c^{\text{agg}}$.

10. The method of one of the previous examples, further comprising:

using a long-short term memory (LSTM) with a pointer network that formulates a select clause of the SQL query with one or more column names, comprising:

given a list of column representations and a question representation, selecting the column that best matches the question, wherein the list of column representations is obtained by encoding each column name with a LSTM, wherein the representation of a particular column j, e_j^c is given by

$$h_{j,t}^c = \text{LSTM}\left(\text{emb}\left(x_{j,t}^c\right), h_{j,t-1}^c\right) \qquad e_j^c = h_{j,T_j}^c$$

11. The method of one of the previous examples, further comprising:

using reinforcement learning to formulate where conditions of the SQL query by executing the generated SQL query against the database to obtain a reward $R(q(\hat{y}), q_g)$ defined as:

$$R(q(y), q_g) = \begin{cases} -2, & \text{if } q(y) \text{ is not a valid SQL query} \\ -1, & \text{if } q(y) \text{ is a valid SQL query and executes to an incorrect result} \\ +1, & \text{if } q(y) \text{ is a valid SQL query and executes to the correct result} \end{cases}$$

where $q(y)$ denotes the query generated by the model, and q_g denotes the ground truth query corresponding to the input natural language query.

12. A method comprising:

processing a natural language request to generate parts of an SQL query, including:

using a long-short term memory (LSTM) with a pointer network that formulates a select clause of the SQL query with one or more column names; and

using an augmented pointer decoder trained by reinforcement learning to formulate where conditions of the SQL query.

13. The method of example 12, further comprising:

using a multilayer perceptron to determine an aggregation operation of the SQL query applicable to columns selected under specified conditions.

14. The method of example 12 or 13, wherein using a long-short term memory (LSTM) with a pointer network that formulates a select clause of the SQL query with one or more column names comprises:

given a list of column representations and a question representation, selecting the column that best matches the question.

15. The method of example 14, wherein the list of column representations is obtained by encoding each column name with a LSTM, wherein the representation of a particular column j, e_j^c is given by

$$h_{j,t}^c = \text{LSTM}\left(\text{emb}\left(x_{j,t}^c\right), h_{j,t-1}^c\right) \qquad e_j^c = h_{j,T_j}^c$$

16. The method of one of the examples 14-15, wherein generating the one or more input representations comprises computing an input representation κ^{agg} by:

computing a scalar attention score, $\alpha^{in}{}_t = W^{inp} h_t^{enc}$, for each tth token in sequence of tokens, wherein h_t^{enc} is a state of an encoder corresponding to a tth word in the input sequence,

normalizing the vector of scores $\alpha^{in} = [\alpha_1^{inp}, \alpha_2^{inp}, \ldots]$ to produce a distribution over the tokens in the sequence of tokens,

obtaining the input representation κ^{agg} as

$$\kappa^{\text{agg}} = \sum_t \beta_t^{\text{inp}} h_t^{\text{enc}}$$

wherein $\beta^{\text{agg}} = \text{softmax}(\alpha^{\text{agg}})$, and $\alpha^{\text{agg}} = W^{\text{agg}} \tanh(V^{\text{agg}} \kappa^{\text{agg}} + b^{\text{agg}}) + c^{\text{agg}}$

.17. The method of one of the examples 14-16, further comprising:

using reinforcement learning to formulate where conditions of the SQL query by executing the generated SQL query against the database to obtain a reward $R(q(\hat{y}), q_g)$ defined as:

$$R(q(y), q_g) = \begin{cases} -2, & \text{if } q(y) \text{ is not a valid SQL query} \\ -1, & \text{if } q(y) \text{ is a valid SQL query and executes to an incorrect result} \\ +1, & \text{if } q(y) \text{ is a valid SQL query and executes to the correct result} \end{cases}$$

where $q(y)$ denotes the query generated by the model, and q_g denotes the ground truth query corresponding to the input natural language query.

18. A non-transitory computer readable storage medium comprising computer executable code that when executed by one or more processors causes the one or more processors to perform steps of any of the examples 1-17.

19. A computer system for generating a recurrent neural network (RNN) architecture, the computer system comprising:

one or more computer processors; and

a non-transitory computer-readable storage medium comprising computer executable code that when executed by one or more computers causes the one or more computers to perform steps of a method according to claim 18.

20. A computer system comprising one or more computer processors and at least one non-transitory storage medium and further comprising:

means for receiving an input natural language query based on data stored using a database schema;

means for generating a sequence of tokens from a plurality of terms comprising:

terms of the input natural language query,

a set of columns of the database schema, and

vocabulary of a database query language;

means for generating one or more input representations, each input representation obtained by encoding the sequence of tokens;

means for accessing a plurality of machine learning based models, each model configured to predict a portion of a database query corresponding to the input natural language query;

means for performing, for each of the plurality of models, executing the model based on an input representation to generate a portion of the database query;

means for combining the generated portions of the database query to obtain the database query; and

means for executing the database query to obtain a result set.

21. A computer system comprising one or more computer processors and at least one non-transitory storage medium and further comprising:

means processing a natural language request to generate parts of an SQL query, including:

means for using a long-short term memory (LSTM) with a pointer network that formulates a select clause of the SQL query with one or more column names; and

means for using an augmented pointer decoder trained by reinforcement learning to formulate where conditions of the SQL query.

ALTERNATIVE EMBODIMENTS

[0066] Although the embodiments disclosed are based on relational databases and illustrated using SQL, the techniques disclosed are applicable to other types of databases, for example, object based databases, object relational databases, and so on. The techniques disclosed are applicable if the database query language used for the particular type of database supports features equivalent to result columns, aggregation clauses, or condition clause. For example, if a database query language supports condition clause, the condition clause predictor can be used to predict the condition clause for an output database query based on an input natural language query.

[0067] It is to be understood that the Figures and descriptions of the present invention have been simplified to illustrate elements that are relevant for a clear understanding of the present invention, while eliminating, for the purpose of clarity, many other elements found in a typical distributed system. Those of ordinary skill in the art may recognize that other elements and/or steps are desirable and/or required in implementing the embodiments. However, because such elements and steps are well known in the art, and because they do not

facilitate a better understanding of the embodiments, a discussion of such elements and steps is not provided herein. The disclosure herein is directed to all such variations and modifications to such elements and methods known to those skilled in the art.

[0068] Some portions of above description describe the embodiments in terms of algorithms and symbolic representations of operations on information. These algorithmic descriptions and representations are commonly used by those skilled in the data processing arts to convey the substance of their work effectively to others skilled in the art. These operations, while described functionally, computationally, or logically, are understood to be implemented by computer programs or equivalent electrical circuits, microcode, or the like. Furthermore, it has also proven convenient at times, to refer to these arrangements of operations as modules, without loss of generality. The described operations and their associated modules may be embodied in software, firmware, hardware, or any combinations thereof.

[0069] As used herein any reference to "one embodiment" or "an embodiment" means that a particular element, feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment. The appearances of the phrase "in one embodiment" in various places in the specification are not necessarily all referring to the same embodiment.

[0070] Some embodiments may be described using the expression "coupled" and "connected" along with their derivatives. It should be understood that these terms are not intended as synonyms for each other. For example, some embodiments may be described using the term "connected" to indicate that two or more elements are in direct physical or electrical contact with each other. In another example, some embodiments may be described using the term "coupled" to indicate that two or more elements are in direct physical or electrical contact. The term "coupled," however, may also mean that two or more elements are not in direct contact with each other, but yet still co-operate or interact with each other. The embodiments are not limited in this context.

[0071] As used herein, the terms "comprises," "comprising," "includes," "including," "has," "having" or any other variation thereof, are intended to cover a non-exclusive inclusion. For example, a process, method, article, or apparatus that comprises a list of elements is not necessarily limited to only those elements but may include other elements not expressly listed or inherent to such process, method, article, or apparatus. Further, unless

expressly stated to the contrary, “or” refers to an inclusive or and not to an exclusive or. For example, a condition A or B is satisfied by any one of the following: A is true (or present) and B is false (or not present), A is false (or not present) and B is true (or present), and both A and B are true (or present).

[0072] In addition, use of the “a” or “an” are employed to describe elements and components of the embodiments herein. This is done merely for convenience and to give a general sense of the invention. This description should be read to include one or at least one and the singular also includes the plural unless it is obvious that it is meant otherwise.

[0073] Upon reading this disclosure, those of skill in the art will appreciate still additional alternative structural and functional designs for a system and a process for displaying charts using a distortion region through the disclosed principles herein. Thus, while particular embodiments and applications have been illustrated and described, it is to be understood that the disclosed embodiments are not limited to the precise construction and components disclosed herein. Various modifications, changes and variations, which will be apparent to those skilled in the art, may be made in the arrangement, operation and details of the method and apparatus disclosed herein without departing from the spirit and scope defined in the appended claims.

APPENDIX

The following document titled “Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning" discloses "Seq2SQL”, which is part of this appendix and thus part of the disclosure of this application, discloses “Seq2SQL” - a deep neural network for translating natural language questions to corresponding SQL queries. Seq2SQL leverages the structure of SQL queries to significantly reduce the output space of generated queries. Seq2SQL comprises three components that leverage the structure of a SQL query to greatly reduce the output space of generated queries. In particular, it uses a pointer decoder and policy gradient to generate the conditions of the query, which we show are unsuitable for optimization using cross entropy loss due to their unordered nature. Seq2SQL is trained using a mixed objective combining cross entropy losses and reinforcement learning rewards from live query execution on a database. These characteristics allow the model to achieve much improved results on query generation.

Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

A significant amount of the world's knowledge is stored in relational databases. However, the ability for users to retrieve facts from a database is limited due to a lack of understanding of query languages such as SQL. We propose Seq2SQL, a deep neural network for translating natural language questions to corresponding SQL queries. Our model leverages the structure of SQL queries to significantly reduce the output space of generated queries. Moreover we apply policy gradient reinforcement learning using in-the-loop query execution to learn a policy for generating the conditions of the SQL query. Elements of queries are unordered and not suitable for optimization via cross entropy loss. In addition, we will publish WikiSQL, a dataset of 37,000 hand-annotated examples of questions and SQL queries distributed across 9,500 tables from Wikipedia. This dataset is required to train our model and is an order of magnitude larger than comparable datasets. By applying policy gradient methods using rewards from in-the-loop query execution on WikiSQL, we show that Seq2SQL outperforms attentional sequence to sequence models, improving execution accuracy from 21.8% to 52.2% and logical form accuracy from 20.4% to 46.6%.

1 Introduction

A vast amount of today's information is stored in relational databases, which provide the foundation of crucial applications such as medical records [Hillestad et al., 2005], financial markets [Beck et al., 2000], and customer relations management [Ngai et al., 2009]. However, accessing information in relational databases requires an understanding of query languages such as SQL, which, while powerful, is difficult to master. A prominent research area at the intersection of natural language processing and human-computer interactions is the study of natural language interfaces (NLI) [Androutsopoulos et al., 1995], which seek to provide means for humans to interact with computers through the use of natural language. We investigate one particular aspect of natural language interfaces applied to relational databases: translating natural language questions to SQL queries.

Our main contributions in this work are two-fold. First, we introduce Seq2SQL, a deep neural network for translating natural language questions to corresponding SQL queries. Seq2SQL, shown in Figure 1, consists of three components that leverage the structure of a SQL query to greatly reduce the output space of generated queries. In particular, it uses a pointer decoder and policy gradient to generate the conditions of the query, which we show are unsuitable for optimization using cross entropy loss due to their unordered nature. Seq2SQL is trained using a mixed objective combining cross entropy losses and reinforcement learning rewards from live query execution on a database. These characteristics allow the model to achieve much improved results on query generation.

Second, we release WikiSQL, a corpus of 37,000 hand-annotated instances of natural language questions, SQL queries, and SQL tables extracted from 9,500 HTML tables from Wikipedia. WikiSQL

Submitted to 31st Conference on Neural Information Processing Systems (NIPS 2017). Do not distribute.

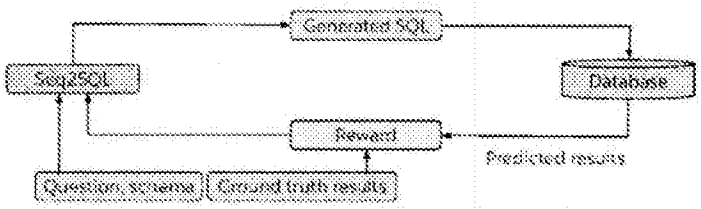

Figure 1: Seq2SQL takes as input a question and list of columns of a table. It generates the corresponding SQL query, which, during training, is executed against a database. The result of the execution is utilized as the reward to train the reinforcement learning algorithm.

is an order of magnitude larger than comparable datasets which also provide logical forms along with natural language utterances. We release the tables used in WikiSQL both in raw JSON format as well as in the form of a SQL database. Along with WikiSQL, we also release a query execution engine for the database, which we use for in-the-loop query execution during policy learning. On WikiSQL, we show that Seq2SQL significantly outperforms an attentional sequence to sequence baseline, which obtains 30.3% execution accuracy, as well as a pointer network baseline, which obtains 43.2% execution accuracy. By leveraging the inherent structure of SQL queries and applying policy gradient methods using the reward signals from live query execution during training, Seq2SQL achieves a marked improvement on WikiSQL, obtaining 52.2% execution accuracy.

2 Related Work

Learning logical forms from natural language utterances. In semantic parsing for question answering (QA), natural language questions are parsed into logical forms, which are then executed on a knowledge graph [Zelle and Mooney, 1996, Wong and Mooney, 2007, Zettlemoyer and Collins, 2005, 2007]. Other work in semantic parsing has focused on learning parsers without relying on annotated logical forms by leveraging conversational logs [Artzi and Zettlemoyer, 2011], demonstrations [Artzi and Zettlemoyer, 2013], distant supervision [Cai and Yates, 2013, Reddy et al., 2014], and question-answer pairs [Liang et al., 2011]. Recently, Pasupat and Liang [2015] introduced a floating parser to address the large amount of variation in utterances and table schema for semantic parsing on web tables. Our approach is different from these semantic parsing techniques in that it does not rely on a high quality grammar and lexicon, and instead relies on representation learning.

Semantic parsing datasets. Previous semantic parsing systems were typically designed to answer complex and compositional questions over close-domain, fixed-schema datasets such as GeoQuery [Tang and Mooney, 2001] and ATIS [Price, 1990]. Researchers have also investigated QA over subsets of large-scale knowledge graphs such as DBPedia [Stanc and Mladenic, 2017] and Freebase [Cai and Yates, 2013, Berant et al., 2013]. Compared to these datasets, WikiSQL is the largest by a significant margin - it is an order of magnitude larger in the number of examples and/or the number of domains/table schema. The dataset "Overnight," Wang et al. [2015] uses a similar crowd-sourcing process to build a dataset of (natural language question,logical form) pairs, but it is comprised of only 8 domains. WikiTableQuestions [Pasupat and Liang, 2015], which, like WikiSQL, is also a collection of question and answers over a large quantity of tables extracted from Wikipedia, achieves a balance between the breadth of the problem domain and the depth of question complexity. However WikiTableQuestions does not provide logical forms whereas WikiSQL does. In addition, WikiTableQuestions is focused on the task of QA over potentially noisy web tables, whereas WikiSQL is focused on generating SQL queries for questions over relational database tables, with the intent of building natural language interfaces for real-world, production databases.

Representation learning for sequence generation. Our baseline model is based on the attentional sequence to sequence model (Seq2Seq) proposed by [Bahdanau et al., 2015]. Seq2Seq has lead to significant progress in a variety of natural language processing tasks such as machine translation [Sutskever et al., 2014, Bahdanau et al., 2015], summarization [Nallapati et al., 2016, Paulus et al., 2017], and semantic parsing [Dong and Lapata, 2016]. Seq2SQL is inspired by pointer networks [Vinyals et al., 2015], which, instead of generating words from a fixed vocabulary like attentional sequence to sequence models, generates by selecting words from the input sequence. This class of models has been successfully applied to tasks such as language modeling [Merity et al., 2017], summarization [Gu et al., 2016], combinatorial optimization [Bello et al., 2017], and question answering [Seo et al., 2017, Xiong et al., 2017]. Neural methods have also been applied to semantic

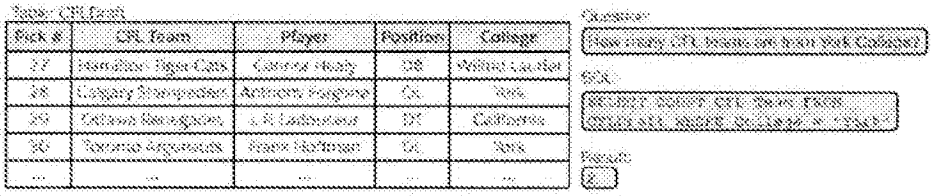

Figure 2: An example in WikiSQL. The inputs consist of a table and question. The outputs consist of a ground truth SQL query and the corresponding result from execution.

parsing [Dong and Lapata, 2016, Neelakantan et al., 2017]. However, unlike [Dong and Lapata, 2016], we use pointer based generation instead of Seq2Seq generation and show that the former approach works significantly better, especially for unseen words and column names. Unlike [Neelakantan et al., 2017], our model does not require access to the table content during inference. In contrast to both methods, we train our model using policy gradient, which again significantly improves performance.

Natural language interface for databases. The practical applications of WikiSQL have much to do with Natural Language Interfaces (NLI). One of the prominent works in this area is PRECISE [Popescu et al., 2003], which translates questions to SQL queries and identifies questions that it is not confident about. Giordani and Moschitti [2012] proposed another system to translate questions to SQL, in which the model first generates candidate queries from a grammar, then ranks the queries using tree kernels. Both of these approaches rely on a high quality grammar and are likely unsuitable for tasks such as WikiSQL, which requires generalization to new schema. Iyer et al. [2017] also proposed a system to translate to SQL, but with a Seq2Seq model that is further improved with human feedback. Compared to this model, we show that Seq2SQL substantially outperforms Seq2Seq and uses reinforcement learning instead of human feedback during training.

3 Model

The Seq2SQL model generates a SQL query from a natural language question and table schema.[1] One powerful baseline model is the Seq2Seq model utilized for machine translation [Bahdanau et al., 2015]. However, the output space of the standard softmax in a Seq2Seq model is unnecessarily large for this task. In particular, we note that the problem can be dramatically simplified by limiting the output space of the generated sequence to the union of the table schema, the question utterance, and SQL key words. The resulting model is similar to a pointer network [Vinyals et al., 2015] with augmented inputs. We first show this augmented pointer network model, then address its limitations, particularly with respect to generating unordered query conditions, in our description of Seq2SQL.

3.1 Augmented Pointer Network

The augmented pointer network generates the SQL query token-by-token by selecting from an input sequence. In our case, the input sequence is the concatenation of the column names, required for the selection column and the condition columns of the query, the question, required for the conditions of the query, and the limited vocabulary of the SQL language such as `SELECT`, `COUNT` etc. In the example shown in Figure 2, the column name tokens consist of "Pick", "#", "CFL", "Team" etc.; the question tokens consist of "How", "many", "CFL", "teams" etc.; the SQL tokens consist of `SELECT`, `WHERE`, `COUNT`, `MIN`, `MAX` etc. With this augmented input sequence, the pointer network is able to fully produce the SQL query by selecting exclusively from the input.

Suppose we are given a list of N table columns and a question such as in Figure 2, and asked to produce the corresponding SQL query. Let $x_j^c = \{x_{j,1}^c, x_{j,2}^c, ..., x_{j,T_j}^c\}$ denote the sequence of words in the name of the jth column, where $x_{j,i}^c$ represents the ith word in the jth column and T_j represents the total number of words in the jth column. Similarly, let x^q and x^a respectively denote the sequence of words in the question and the set of all unique words in the SQL vocabulary.

We define the input sequence x as the concatenation of all the column names, the question, and the SQL vocabulary:

$$x = [\texttt{<col>}; x_1^c; x_2^c; ...; x_N^c; \texttt{<sql>}; x^a; \texttt{<question>}; x^q] \quad (1)$$

[1] For simplicity, we define table schema as the names of the columns in the table.

where $[a; b]$ denotes the concatenation between the sequences a and b and we add special sentinel token between all three sequences to demarcate the boundaries.

The network first encodes x using a two layer, bidirectional Long Short-Term Memory network [Hochreiter and Schmidhuber, 1997]. The input to the encoder are the embeddings corresponding to words in the input sequence. We denote the output of the encoder by h, where h_t is the state of the encoder corresponding to the tth word in the input sequence. For brevity, we do not write out the LSTM equations, which are described by Hochreiter and Schmidhuber [1997]. We then apply a pointer network similar to that proposed by Vinyals et al. [2015] to the input encodings h.

The decoder network uses a two layer, unidirectional LSTM. During each decoder step s, the decoder LSTM takes as input y_{s-1}, the query token generated during the previous decoding step, and outputs the state g_s. Next, the decoder produces a scalar attention score $\alpha_{s,t}^{ptr}$ for each position t of the input sequence x:

$$\alpha_{s,t}^{ptr} = W^{ptr}\tanh\left(U^{ptr}g_s + V^{ptr}h_t\right) \quad (2)$$

The input token with the highest score is then chosen by the decoder to be the next token of the SQL query, $y_s = \text{argmax}(\alpha_s^{ptr})$.

3.2 Seq2SQL

While the augmented pointer model is able to solve the SQL generation problem, it does not leverage the structure inherent in SQL queries. Normally, a SQL query such as that shown in Figure 3 consists of three components. The first component is the aggregation operator, in this case `COUNT`, which produces a summary of the rows selected by the SQL query. Alternatively the query may request no summary statistics, in which case an aggregation operator is not provided. The second component is the selection column(s), in this case `Engine`, which identifies the column(s) that are to be included in the returned results. The third component is the `WHERE` clause of the query, in this cases `WHERE Driver = Val Musetti`, which contains conditions by which to filter the rows. Here, we want only rows in which the driver is "Val Musetti".

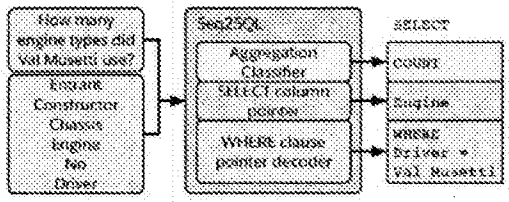

Figure 3: The Seq2SQL model has three components, corresponding to the three parts of a SQL query (right). The input to the model are the question (top left) and the table column names (bottom left).

To leverage the structure present in SQL queries, we introduce Seq2SQL. Seq2SQL, as shown in Figure 3, is composed of three parts that correspond to the aggregation operator, the selection column, and the `WHERE` clause. First, the network classifies an aggregation operation for the query, with the addition of a null operation that corresponds to no aggregation. Next, the network points to a column in the input table corresponding to the `SELECT` column. Finally, the network generates the conditions for the query using a pointer network. The first two components are supervised using cross entropy loss, whereas the third generation component is trained using policy gradient reinforcement learning in order to address the unordered nature of query conditions (we explain this in detail in Section 3.2). Similar to moving from Seq2Seq to the augmented pointer model, utilizing the structure of a SQL query allows Seq2SQL to further reduce the output space of SQL queries. We show later that this leads to a significantly higher performance compared to both the Seq2Seq model and the pointer model.

Aggregation Operation

The aggregation operation depends on the question. For the example shown in Figure 3, we intuit that the correct operator is `COUNT` because the question asks for "How many".

To compute the aggregation operation, we first compute an input representation x^{agg}. Similar to Equation 2, we first compute the scalar attention score, $\alpha_t^{agg} = W^{agg}h_t^{enc}$, for each tth token in the input sequence. The vector of all scores, $\alpha^{agg} = [\alpha_1^{agg}, \alpha_2^{agg}, ...]$, is then normalized to produce a distribution over all the tokens in the input sequence x, $\beta^{agg} = \text{softmax}(\alpha^{agg})$. The input representation, x^{agg}, is the sum over the column encodings weighted by the corresponding

normalized attention weight:

$$\kappa^{\text{agg}} = \sum_i \beta_i^{\text{agg}} h_i^{\text{enc}} \tag{3}$$

Let α^{agg} denote the scores over the aggregation operations such as COUNT, MIN, MAX, and the no-aggregation operation NULL. We compute α^{agg} by applying a multi-layer perceptron to the input representation κ^{agg}:

$$\alpha^{\text{agg}} = W^{\text{agg}} \tanh\left(V^{\text{agg}} \kappa^{\text{agg}} + b^{\text{agg}}\right) + c^{\text{agg}} \tag{4}$$

The distribution over the set of possible aggregation operations, $\beta^{\text{agg}} = \text{softmax}\left(\alpha^{\text{agg}}\right)$, is then obtained by applying the softmax function. We use cross entropy loss L^{agg} for the aggregation operation.

SELECT **Column**

The selection column depends on the table columns as well as the question. Namely, for the example in Figure 3, we see from the "How many engine types" part of the question that the query is asking for the "Engine" column. SELECT column prediction is then a matching problem, solvable using a pointer: given the list of column representation and a question representation, we select the column that best matches the question.

In order to produce the representations for the columns, we first encode each column name with a LSTM. The representation of a particular column j, c_j^c, is given by:

$$h_{j,i}^c = \text{LSTM}\left(\text{emb}\left(x_{j,i}^c\right), h_{j,i-1}^c\right) \qquad c_j^c = h_{j,T_j}^c \tag{5}$$

Here, $h_{j,i}^c$ denotes the ith encoder state of the jth column. c_j^c, column j's representation, is then taken to be the last encoder state.

To construct a representation for the question, we compute another input representation κ^{col} using the same architecture as for κ^{agg} (Equation 3) but with untied weights. Finally, we apply a multi-layer perceptron over the column representations for the pointer estimation, conditioned on the input representation, to compute the a score for each column j:

$$\alpha_j^{\text{col}} = W^{\text{col}} \tanh\left(V^{\text{col}} \kappa^{\text{col}} + V^c c_j^c\right) \tag{6}$$

We normalize the scores with a softmax function to produce a distribution over the possible SELECT columns $\beta^{\text{col}} = \text{softmax}\left(\alpha^{\text{col}}\right)$. For the example shown in Figure 3, the distribution would be over the columns "Entrant", "Constructor", "Chassis", "Engine", "No", and the ground truth SELECT column "Driver". We train the SELECT network using cross entropy loss L^{col}.

WHERE **Clause**

The WHERE clause can be trained using a pointer decoder similar to that described in Section 3.1. However, there is a crucial limitation in using the cross entropy loss to optimize the network: the WHERE conditions of a query can be arbitrarily swapped and the query still yield the same result. Suppose we have the question "what are the names of men who are older than 18" and the queries `SELECT name FROM insurance WHERE age > 18 AND gender = "male"` and `SELECT name FROM insurance WHERE gender = "male" AND age > 18`. Both queries obtain the correct execution result despite not having exact string match. Suppose the former query is provided as the ground truth, using cross entropy loss to supervise the generation would then wrongly penalize the latter query. To address this problem, we apply reinforcement learning to learn a policy to directly optimize the expected "correctness" of the execution result (Equation 7).

Instead of teacher forcing at each step, we sample from the output distribution to obtain the next token. At the end of the generation procedure, we execute the generated SQL query against the database to obtain a reward. Let $q\left(y\right)$ denote the query generated by the model and q_g denote the ground truth query corresponding to the question. The reward $R\left(q\left(y\right), q_g\right)$ is defined as:

$$R\left(q\left(y\right), q_g\right) = \begin{cases} -2. & \text{if } q\left(y\right) \text{ is not a valid SQL query} \\ -1. & \text{if } q\left(y\right) \text{ is a valid SQL query and executes to an incorrect result} \\ +1. & \text{if } q\left(y\right) \text{ is a valid SQL query and executes to the correct result} \end{cases} \tag{7}$$

Let $y = [y^1, y^2, ..., y^T]$ denote the sequence of generated tokens in the WHERE clause of a SQL query $q(y)$ that resulted in an execution reward of $R(q(y), q_g)$. The loss, $L^{whe} = -\mathbb{E}_y[R(q(y), q_g)]$, is the negative expected reward over possible WHERE clauses.

As shown by Schulman et al. [2015], the act of sampling during the forward pass of the network results in the corresponding injection of a synthetic gradient, which is a function of the reward, during the backward pass. Specifically, let $P_t(y^t)$ denote the probability of choosing token y^t during time step t, the corresponding synthetic gradient is $R(q(y), q_g) \log P_t(y^t)$.

Mixed Objective Function

The model is trained using gradient descent to minimize the objective function $f^{obj} = L^{agg} + L^{sel} + L^{whe}$. The total gradient during the backward pass is then the equally weighted sum of the gradients from the cross entropy loss in predicting the SELECT column, the cross entropy loss in predicting the aggregation operation, and the expected reward for policy learning as described above.

4 WikiSQL

WikiSQL is a collection of natural language utterances and corresponding SQL queries and SQL tables. To our knowledge, it is the largest corpus available of its kind. A single example in WikiSQL, such as that shown in Figure 2, contains a SQL table, a SQL query, as well as the natural language utterance corresponding to the SQL query. In this work, we do not use the content of the SQL table, but only its schema (e.g. column names). This means that systems trained on WikiSQL do not need to access the end user's database contents during inference.

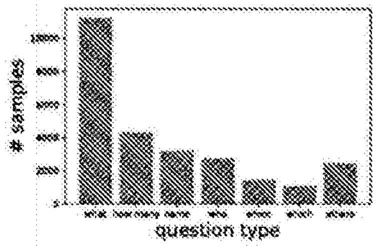

Figure 4: Distribution of question types in WikiSQL.

Table 1 shows how WikiSQL compares to related datasets. Namely, we note that WikiSQL is the largest hand-annotated semantic parsing dataset to date - it is an order of magnitude larger than other datasets that have logical forms, either in terms of the number of examples or the number of table schemas. Moreover, the queries in WikiSQL span over a large number of table schemas and cover a large diversity of data. The large quantity of schemas presents a challenge compared to other datasets: the model must be able to not only generalize to new queries, but to new schemas as well. Finally, WikiSQL contains challenging, realistic data extracted from the web. This is evident in the distributions of the number of columns, the lengths of questions, and the length of queries, which are respectively shown in Figure 5. Another indicator of the variety of questions in the dataset is the distribution of question types, which is shown in Figure 4.

WikiSQL is collected using crowd-sourcing on Amazon Mechanical Turk (AMT) in two phases. In the first phase, a worker is asked to paraphrase a generated question for a table. The generated question itself is formed using a template, filled using a randomly generated SQL query. Due to the large variation in HTML tables, we ensure the validity and complexity of the tables by keeping only those that are legitimate database tables and are sufficiently large in the number of rows and columns. In the second phase, two other workers are asked to verify that the paraphrase has the same meaning as the generated question. We discard paraphrases that do not show enough variation, as measured by the character edit distance from the generated question, as well as those that are deemed incorrect by both workers during the verification phase. More details on how WikiSQL was collected is shown in Section A of the Appendix. We make available examples of the interface used during the paraphrase phase [2] and during the verification phase [3]. The HTML pages of the interfaces are also included in the supplementary materials.

The tables and their corresponding paraphrases and SQL queries are randomly slotted into train, dev, and test splits, such that each table is only present in exactly one split. In addition to the raw content

[2] Example of the paraphrasing interface used for WikiSQL: https://tableqa.herokuapp.com/example/3562572-25

[3] Example of the verification interface used for WikiSQL: https://paraphrase.herokuapp.com/example/3WHP9MDOKQ3FQ9XNG0PC1AFKJBNCFZ

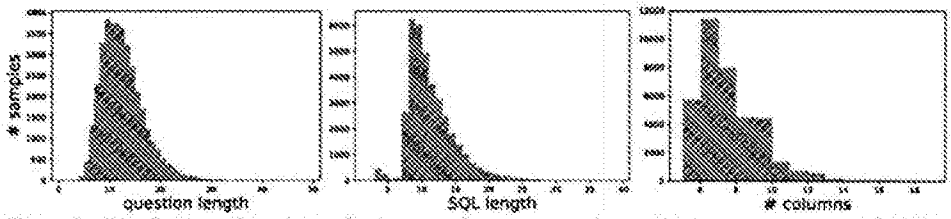

Figure 5: Distribution of numbers of columns, tokens per question, and tokens per query in WikiSQL.

of tables, queries, results, and natural utterances, we also release a corresponding SQL database as well as an execution engine for queries system.

4.1 Evaluation

One natural way to evaluate WikiSQL is to use the execution accuracy metric. Let N denote the total number of examples in the dataset and N_{ex} denote the total number of queries that, when executed, result in the correct result. The execution accuracy is defined as $\text{Acc}_{ex} = \frac{N_{ex}}{N}$.

Another metric is the logical form accuracy. Let N_{lf} denote the total number of queries has exact string match with the ground truth query used to collect the paraphrase. The logical form accuracy is defined as $\text{Acc}_{lf} = \frac{N_{lf}}{N}$.

There are benefits and drawbacks to both metrics. As we will show in Section 3.2, the logical form accuracy incorrectly penalizes queries that achieve the correct result but do not have exact string match with the ground truth query. Alternatively, it is possible to construct a SQL query that does not correspond to the natural language question but nevertheless obtain the same result. An example in which two different queries produce the same result is `SELECT COUNT(name) WHERE SSN = 123` and `SELECT COUNT(SSN) WHERE SSN = 123` if two people with different names share the SSN 123. Due to these observations, we use both metrics to evaluate our models.

Dataset	Size	LF	Schema
WikiSQL	**37000**	**yes**	**9500**
Geoquery	880	yes	8
ATIS	5871	yes*	141
Freebase917	917	yes	81*
Overnight	26098	yes	8
WebQuestions	5810	no	2420
WikiTableQuestions	22033	no	2108

Table 1: Comparison between WikiSQL and existing datasets. The datasets are GeoQuery880 [Tang and Mooney, 2001], ATIS [Price, 1990], Free917 [Cai and Yates, 2013], Overnight [Wang et al., 2015], WebQuestions [Berant et al., 2013], and WikiTableQuestions [Pasupat and Liang, 2015]. "Size" denotes the total number of examples in the dataset. "LF" indicates whether the dataset has annotated logical forms. "Schema" denotes the number of domains or schemas in the dataset. The format of ATIS is presented as a slot filling task. Each [Schema] pair corresponds to a table.

5 Experiments

The dataset is tokenized using Stanford CoreNLP. In particular, we used the normalized tokens for training and reverse them into their original gloss before outputting the query. This way, the queries are composed of tokens from the original gloss and are hence executable in the database.

We use pretrained word embeddings from GloVe [Pennington et al., 2014] and character ngram embeddings [Hashimoto et al., 2016]. Let w_x^g denote the GloVe embedding and w_x^c the character embedding for word x. Here, w_x^c is the mean of the embeddings of all the character n-grams in x. For a given word x, we define its word embedding $w_x = [w_x^g; w_x^c]$ as the concatenation of its GloVe embedding and its character embedding. For words that do not have pretrained embeddings, we assign the embeddings to the zero vector. We do not train embeddings and instead fix the pretrained embeddings.

All networks are run for a maximum of 100 epochs with early stopping using the execution accuracy criteria on the dev split. We train the networks using ADAM [Kingma and Ba, 2014] and regularize using dropout [Srivastava et al., 2014]. We implement all models using PyTorch [5].

[5] https://pytorch.org

Model	Dev Acc_{lf}	Dev Acc_{ex}	Test Acc_{lf}	Test Acc_{ex}
Attentional Seq2Seq	20.3%	22.4%	20.4%	21.8%
Aug. Pointer Network	38.5%	43.5%	37.7%	43.2%
Seq2SQL (no RL)	46.2%	49.9%	44.4%	47.3%
Seq2SQL	**48.3%**	**54.2%**	**46.6%**	**52.2%**

Table 2: Performance on WikiSQL. Both metrics are defined in Section 4.1. The Seq2SQL (no RL) model is identical in architecture when compared to Seq2SQL, except the WHERE clause is supervised via teacher forcing as opposed to reinforcement learning.

To train Seq2SQL, we first pretrain a version in which the WHERE clause is supervised via teacher forcing (e.g. the sampling procedure is not trained from scratch) and then continue training using reinforcement learning. In order to obtain the rewards described in Section 3.2, we use the query execution engine described in Section 4.

5.1 Result

We compare results against an attentional sequence to sequence baseline used by Bahdanau et al. [2015] for machine translation. This model is described in detail in Section B of the Appendix. We also present an ablation of Seq2SQL, in which the WHERE clause of the SQL query is trained using the cross entropy loss (e.g. teacher forcing) in the same fashion as the augmented pointer network. The performance of the three models are shown in Table 2.

We note that reducing the output space by utilizing the augmented pointer network significantly improves upon the attentional sequence to sequence model by 21.4%. Moreover, leveraging the structure of SQL queries leads to another significant improvement of 4.1%, as is shown by the performance of Seq2SQL without RL compared to the augmented pointer network. Finally, the full Seq2SQL model, trained using reinforcement learning based on rewards from live query executions on a database, leads to yet another significant performance of 4.9%.

5.2 Analysis

Limiting the output space lead to more accurate conditions. When we compare the outputs of the attentional sequence to sequence model with those of the augmented pointer model, we find that the latter generates much higher quality WHERE clause conditions. For example, for the question "in how many districts was a successor seated on march 4, 1850?", former generates `SELECT COUNT District WHERE Date successor seated = seated march 4` whereas the latter generates `SELECT COUNT District WHERE Date successor seated = seated march 4 1850`. Similarly, for the question "what's doug battaglia's pick number?", the former generates `SELECT Pick WHERE Player = doug` whereas the latter generates `SELECT Pick WHERE Player = doug battaglia`. A key reason for this is that conditions tend to be comprised of rare words (e.g. "1850" occurs very infrequently in the corpus). The Seq2Seq model is then inclined to produce a condition that contains frequently occurring tokens in the training corpus, such as "march 4" for date, or "doug" for player name. The pointer network does not face this problem, as the output distribution is exclusively constructed from the input sequence.

Incorporating structure reduces invalid queries. As expected, incorporating the simple selection and aggregation structure into the model dramatically reduces the percentage of invalid SQL queries generated from 39.8% to 7.1%. Generally, we find that a large quantity of invalid queries result from invalid column names. That is, the model generates a query that refers to columns that are not present in the table. This is particularly helpful when the names of columns contain many tokens, such as "Miles (km)", which is comprised of 4 tokens after tokenization. Introducing a specialized classifier for the aggregation also reduces the error rate due to having the incorrect aggregation operator. For example, Table 3 shows that adding the aggregation classifier substantially improves the recall for predicting the COUNT operator. For more queries produced by the different models, please see Section C of the Appendix.

Reinforcement learning generates higher quality WHERE clause that are at times ordered differently compared to ground truth. As expected, the model trained with policy learning ob-

tains correct results in which the order of the conditions is different than the order in the ground truth query. For example, for the question "in what district was the democratic candidate first elected in 1992?", the ground truth conditions are `WHERE First elected = 1992 AND Party = Democratic` whereas Seq2SQL generates `WHERE Party = Democratic AND First elected = 1992`. More generally, we note that in most cases where Seq2SQL is correct and Seq2SQL without policy learning is not, the latter produces an incorrect `WHERE` clause. For example, for the rather complex question "what is the race name of the 12th round trenton, new jersey race where a.j. foyt had the pole position?", the ablation of Seq2SQL trained without reinforcement learning generates `WHERE` clause `WHERE rnd = 12 and track = a.j. foyt AND pole position = a.j. foyt` whereas the policy gradient based Seq2SQL correctly generates the clause `WHERE rnd = 12 AND pole position = a.j. foyt`.

Model	Precision	Recall	F1
Aug. Pointer	87.0%	47.8%	61.7%
Seq2SQL	85.9%	79.7%	82.7%

Table 3: Performance on the COUNT aggregation operator. The Seq2SQL model's inclusion of the COUNT operator is compared with the ground truth query's inclusion of the COUNT operator.

Difficult table schema present a challenge. In numerous HTML tables, the column names do not accurately portray the column contents. For example, in one of the tables in the dev set, the number of voters from the Bronx is listed under a column title "the bronx". When asked the question "how many voters from the bronx voted for the socialist party", the model mistakenly generate the aggregation operator COUNT due to the phrase "how many". However, due to the way this table is organized, the number of voters are actually listed in the column and the COUNT operation is superfluous. Another difficult example is the question "in what year did a plymouth vehicle win on february 9?", because the model does not understand that "plymouth" refers to a manufacturer of vehicles.

6 Conclusion

We proposed Seq2SQL, a deep neural network for translating natural language questions to corresponding SQL queries that leverages the structure of SQL queries to significantly reduce the output space of generated queries. As a part of Seq2SQL, we proposed using in-the-loop query execution to learn a policy for generating the conditions of the SQL query, which is unordered in nature and not suitable for optimization via cross entropy loss. Along with Seq2SQL, we introduced WikiSQL, a dataset of natural language questions and SQL queries that is an order of magnitude larger than comparable datasets. Finally, we showed that Seq2SQL significantly outperforms attentional sequence to sequence models on WikiSQL, improving execution accuracy from 21.8% to 52.2% and logical form accuracy from 20.4% to 46.6%. In future work, we plan on investigating more complex queries such as nested queries and more sophisticated reward functions.

References

I. Androutsopoulos, G. Ritchie, and P. Thanisch. Natural language interfaces to databases - an introduction. 1995.

Y. Artzi and L. S. Zettlemoyer. Bootstrapping semantic parsers from conversations. In *EMNLP*, 2011.

Y. Artzi and L. S. Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *TACL*, 1:49–62, 2013.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *ICLR*, 2015.

T. Beck, A. Demirgüç-Kunt, and R. Levine. A new database on the structure and development of the financial sector. *The World Bank Economic Review*, 14(3):597–605, 2000.

I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. *ICRL*, 2017.

J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, 2013.

C. Bhagavatula, T. Noraset, and D. Downey. Methods for exploring and mining tables on wikipedia. In *IDEA@KDD*, 2013.

Q. Cai and A. Yates. Large-scale semantic parsing via schema matching and lexicon extension. In *ACL*, 2013.

L. Dong and M. Lapata. Language to logical form with neural attention. *ACL*, 2016.

A. Giordani and A. Moschitti. Translating questions to SQL queries with generative parsers discriminatively reranked. In *COLING*, 2012.

J. Gu, Z. Lu, H. Li, and V. O. K. Li. Incorporating copying mechanism in sequence-to-sequence learning. *ACL*, 2016.

K. Hashimoto, C. Xiong, Y. Tsuruoka, and R. Socher. A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks. *arXiv*, cs.CL 1611.01587, 2016.

R. Hillestad, J. Bigelow, A. Bower, F. Girosi, R. Meili, R. Scoville, and R. Taylor. Can electronic medical record systems transform health care? potential health benefits, savings, and costs. *Health affairs*, 24(5):1103–1117, 2005.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 1997.

S. Iyer, I. Konstas, A. Cheung, J. Krishnamurthy, and L. Zettlemoyer. Learning a neural semantic parser from user feedback. In *ACL*, 2017.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv*, abs/1412.6980, 2014.

G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. OpenNMT: Open-Source Toolkit for Neural Machine Translation. *ArXiv*, abs/1701.02810.

P. Liang, M. I. Jordan, and D. Klein. Learning dependency-based compositional semantics. *Computational Linguistics*, 39:389–446, 2011.

T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. In *EMNLP*, 2015.

S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer sentinel mixture models. *ICLR*, 2017.

R. Nallapati, B. Zhou, C. dos Santos, Ç. glar Gulçehre, and B. Xiang. Abstractive text summarization using sequence-to-sequence rnns and beyond. *CoNLL 2016*, page 280, 2016.

A. Neelakantan, Q. V. Le, M. Abadi, A. McCallum, and D. Amodei. Learning a natural language interface with neural programmer. In *ICLR*, 2017.

E. W. Ngai, L. Xiu, and D. C. Chau. Application of data mining techniques in customer relationship management: A literature review and classification. *Expert systems with applications*, 36(2): 2592–2602, 2009.

P. Pasupat and P. Liang. Compositional semantic parsing on semi-structured tables. In *ACL*, 2015.

R. Paulus, C. Xiong, and R. Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

A.-M. Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*, pages 149–157. ACM, 2003.

P. J. Price. Evaluation of spoken language systems: The ATIS domain. 1990.

S. Reddy, M. Lapata, and M. Steedman. Large-scale semantic parsing without question-answer pairs. *TACL*, 2:377–392, 2014.

J. Schulman, N. Heess, T. Weber, and P. Abbeel. Gradient estimation using stochastic computation graphs. In *NIPS*, 2015.

M. J. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. Bidirectional attention flow for machine comprehension. *ICLR*, 2017.

N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15: 1929–1958, 2014.

J. Starc and D. Mladenic. Joint learning of ontology and semantic parser from text. *Intelligent Data Analysis*, 21:19–38, 2017.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

L. R. Tang and R. J. Mooney. Using multiple clause constructors in inductive logic programming for semantic parsing. In *ECML*, 2001.

O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *NIPS*, 2015.

Y. Wang, J. Berant, and P. Liang. Building a semantic parser overnight. In *ACL*, 2015.

Y. W. Wong and R. J. Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *ACL*, 2007.

C. Xiong, V. Zhong, and R. Socher. Dynamic coattention networks for question answering. *ICRL*, 2017.

J. M. Zelle and R. J. Mooney. Learning to parse database queries using inductive logic programming. In *AAAI/IAAI, Vol. 2*, 1996.

L. S. Zettlemoyer and M. Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Uncertainty in Artificial Intelligence*, 2005.

L. S. Zettlemoyer and M. Collins. Online learning of relaxed ccg grammars for parsing to logical form. In *EMNLP-CoNLL*, 2007.

A Collection of WikiSQL

WikiSQL is collected in a paraphrase phases as well as a verification phase. In the paraphrase phase, we use tables extracted from Wikipedia by Bhagavatula et al [Bhagavatula et al., 2013] and remove small tables according to the following criteria:

- the number of cells in each row is not the same
- the content in a cell exceed 50 characters
- a header cell is empty
- the table has less than 5 rows or 5 columns
- over 40% of the cells of a row contain identical content

We also remove the last row of a table because a large quantity of HTML tables tend to have summary statistics in the last row, and hence the last row does not adhere to the table schema defined by the header row.

For each of the table that passes the above criteria, we randomly generate 6 SQL queries according to the following rules:

- the query follows the format `SELECT agg_op agg_col from table where cond1_col cond1_op cond1 AND cond2_col cond2_op cond2 ...`
- the aggregation operator `agg_op` can be empty or `COUNT`. In the event that the aggregation column `agg_col` is numeric, `agg_op` can additionally be one of `MAX` and `MIN`
- the condition operator `cond_op` is =. In the event that the corresponding condition column `cond_col` is numeric, `cond_op` can additionally be one of > and <
- the condition `cond` can be any possible value present in the table under the corresponding `cond_col`. In the event that `cond_col` is numerical, `cond` can be any numerical value sampled from the range from the minimum value in the column to the maximum value in the column.

We only generate queries that produce a non-empty result set. To enforce succinct queries, we remove conditions from the generated queries if doing so does not change the execution result.

For each query, we generate a crude question using a template and obtain a human paraphrase via crowdsourcing on Amazon Mechanical Turk. In each Amazon Mechanical Turk HIT, a worker is shown a table as well as its generated questions and asked to provide paraphrases for each question.

After obtaining natural language utterances from the paraphrase phase, we give each question-paraphrase pair to two other workers in the verification phase to verify that the paraphrase and the original question contain the same meaning.

We then filter the initial collection of paraphrases using the following criteria:

- the paraphrase must be deemed correct by at least one worker during the verification phrase
- the paraphrase must be sufficiently different from the generated question, with a character-level edit distance greater than 10

Figure 2 shows one example from WikiSQL. Here, the corresponding generated question is "what is the total number of CFL Team where College is York".

B Attentional Seq2Seq Baseline

For our baseline, we employ the attentional sequence to sequence model, which has led to significant improvements in machine translation [Bahdanau et al., 2015] and neural semantic parsing [Dong and Lapata, 2016]. The model is based on the global attention encoder-decoder model (with input feeding) described by Luong et al. [2015]. We use an implementation based on OpenNMT [Klein et al.].

We use the same two-layer, bidirectional, stacked LSTM encoder as described previously. The decoder is a two-layer, unidirectional, stacked LSTM. The decoder LSTM is almost identical to that described by Equation 2, with the sole difference coming from input feeding.

$$g_s = \text{LSTM}\left(\left[\text{emb}\left(y_{s-1}\right); \kappa_{s-1}^{\text{dec}}\right], g_{s-1}\right) \tag{8}$$

where the $d_{h,\text{dec}}$ dimensional κ_s^{dec} is the attentional context over the input sequence during the sth decoding step, computed as

$$\alpha_{s,i}^{\text{dec}} = h_s^{\text{dec}} \left(W^{\text{enc}} h_i^{\text{enc}}\right)^T \qquad \beta_s^{\text{dec}} = \text{softmax}\left(\alpha_s^{\text{dec}}\right) \tag{9}$$

$$\kappa_s = \sum_i \beta_{s,i} h_i^{\text{enc}} \tag{10}$$

To produce the output token during the sth decoder step, the concatenation of the decoder state and the attention context is given to a final linear layer to produce a distribution o^{dec} over words in the target vocabulary

$$o^{\text{dec}} = \text{softmax}\left(U^{\text{dec}}\left[h_s^{\text{dec}}; \kappa_s^{\text{dec}}\right]\right) \tag{11}$$

where U^{dec} has dimensions $N^{\text{vocab}} \times d_{h,\text{dec}}$ and N^{vocab} is the size of the output vocabulary.

During training, teacher forcing is used. During inference, a beam size of 5 is used and generated unknown words are replaced by the input words with the highest attention weight.

C Predictions by Seq2SQL

Q	when connecticut & villanova are the regular season winner how many tournament venues (city) are there?
P	SELECT COUNT tournament player (city) WHERE regular season winner city 3 = connecticut & villanova
S'	SELECT COUNT tournament venue (city) WHERE tournament winner = connecticut & villanova
S	SELECT COUNT tournament venue (city) WHERE regular season winner = connecticut & villanova
G	SELECT COUNT tournament venue (city) WHERE regular season winner = connecticut & villanova
Q	what are the aggregate scores of those races where the first leg results are 0-1?
P	SELECT aggregate WHERE 1st . = 0-1
S'	SELECT COUNT agg. score WHERE 1st leg = 0-1
S	SELECT agg. score WHERE 1st leg = 0-1
G	SELECT agg. score WHERE 1st leg = 0-1
Q	what is the race name of the 12th round location, new jersey race where a.j. foyt had the pole position?
P	SELECT race name WHERE location = 12th AND round position = a.j. foyt, new jersey AND
S'	SELECT race name WHERE rnd = 12 AND track = a.j. foyt AND pole position = a.j. foyt
S	SELECT race name WHERE rnd = 12 AND pole position = a.j. foyt
G	SELECT race name WHERE rnd = 12 AND pole position = a.j. foyt
Q	what city is on 89.9?
P	SELECT city WHERE frequency = 89.9
S'	SELECT city of license WHERE frequency = 89.9
S	SELECT city of license WHERE frequency = 89.9
G	SELECT city of license WHERE frequency = 89.9
Q	how many voters from the bronx voted for the socialist party?
P	SELECT MIN % party = socialist
S'	SELECT COUNT the bronx where the bronx = socialist
S	SELECT COUNT the bronx WHERE the bronx = socialist
G	SELECT the bronx WHERE party = socialist
Q	in what year did a plymouth vehicle win on february 9 ?
P	SELECT MIN year (ko) WHERE date = february 9 AND race time = plymouth 9
S'	SELECT year (ko) WHERE date = plymouth 9 AND race time = february 9
S	SELECT year (ko) WHERE date = plymouth 9 AND race time= february 9
G	SELECT year (ko) WHERE manufacturer = plymouth AND date = february 9

Table 4: Examples predictions by the models on the dev split. Q denotes the natural language question and G denotes the corresponding ground truth query. P, S', and S denote, respectively, the queries produced by the Augmented Pointer Network, Seq2SQL without reinforcement learning, Seq2SQL. We omit the FROM table part of the query for succinctness.

We claim:

1. A method carried out by a computer system comprising one or more computers, wherein the computer system performs the following steps:

receiving an input natural language query based on data stored using a database schema;

generating a sequence of tokens from a plurality of terms comprising:

terms of the input natural language query,

a set of columns of the database schema, and

vocabulary of a database query language;

generating one or more input representations, each input representation obtained by encoding the sequence of tokens;

accessing a plurality of machine learning based models, each model configured to predict a portion of a database query corresponding to the input natural language query;

for each of the plurality of models, executing the model based on an input representation to generate a portion of the database query;

combining the generated portions of the database query to obtain the database query; and

executing the database query to obtain a result set.

2. The method of claim 1, wherein the input natural language query is received from a client device and the method further comprises a step of sending the result set to the client device.

3. The method of claim 1, wherein the plurality of models comprise an aggregation classifier model for determining an aggregation operator in the database query, wherein the aggregation classifier model comprises a multi-layer perceptron.

4. The method of claim 1, wherein the plurality of models comprise a result column predictor model for determining the result columns of the database query, wherein the result column predictor model comprises a multi-layer perceptron.

5. The method of claim 1, wherein the plurality of models comprise a condition clause predictor model for determining the condition clause of the database query, wherein the condition clause predictor model is based on reinforcement learning.

6. The method of claim 5, further comprising:

receiving a result set based on a ground truth database query;

determining reward values based on a comparison of the result obtained from the generated query and the result obtained from the ground truth query; and

adjusting weights of the condition clause predictor model based on the reward values.

7. The method of claim 5, further comprising:

determining column encodings corresponding to each token of the sequence;

determining a vector comprising scalar attention scores for each token of the input sequence;

normalizing the vector using a softmax function; and

determining the input representation as the sum of the column encodings weighted by the corresponding normalized score.

8. The method of claim 1, further comprising:

training the plurality of models using gradient descent to minimize an objective function representing a loss based on the result of each of the plurality of models.

9. The method of one of the previous claims, wherein generating the one or more input representations comprises computing an input representation κ^{agg} by:

computing a scalar attention score, $\alpha^{in}{}_t = W^{inp} h_t^{enc}$, for each tth token in sequence of tokens, wherein h_t^{enc} is a state of an encoder corresponding to a tth word in the input sequence,

normalizing the vector of scores $\alpha^{in} = [\alpha_1^{inp}, \alpha_2^{inp}, \ldots]$ to produce a distribution over the tokens in the sequence of tokens,

obtaining the input representation κ^{agg} as

$$\kappa^{\text{agg}} = \sum_t \beta_t^{\text{inp}} h_t^{\text{enc}}$$

wherein $\beta^{\text{agg}} = \text{softmax}(\alpha^{\text{agg}})$, and $\alpha^{\text{agg}} = W^{\text{agg}} \tanh\left(V^{\text{agg}} \kappa^{\text{agg}} + b^{\text{agg}}\right) + c^{\text{agg}}$

10. The method of one of the previous claims, further comprising:

using a long-short term memory (LSTM) with a pointer network that formulates a select clause of the SQL query with one or more column names, comprising:

given a list of column representations and a question representation, selecting the column that best matches the question, wherein the list of column representations is obtained by encoding each column name with a

LSTM, wherein the representation of a particular column j, e_j^c is given by

$$h_{j,t}^c = \text{LSTM}\left(\text{emb}\left(x_{j,t}^c\right), h_{j,t-1}^c\right) \qquad e_j^c = h_{j,T_j}^c$$

11. The method of one of the previous claims, further comprising:

using reinforcement learning to formulate where conditions of the SQL query by executing the generated SQL query against the database to obtain a reward $R\left(q\left(\hat{y}\right), q_g\right)$ defined as:

$$R\left(q\left(y\right), q_g\right) = \begin{cases} -2, & \text{if } q\left(y\right) \text{ is not a valid SQL query} \\ -1, & \text{if } q\left(y\right) \text{ is a valid SQL query and executes to an incorrect result} \\ +1, & \text{if } q\left(y\right) \text{ is a valid SQL query and executes to the correct result} \end{cases}$$

where $q\left(y\right)$ denotes the query generated by the model, and q_g denotes the ground truth query corresponding to the input natural language query.

12. A method comprising:

processing a natural language request to generate parts of an SQL query, including:

using a long-short term memory (LSTM) with a pointer network that formulates a select clause of the SQL query with one or more column names; and

using an augmented pointer decoder trained by reinforcement learning to formulate where conditions of the SQL query.

13. The method of claim 12, further comprising:

using a multilayer perceptron to determine an aggregation operation of the SQL query applicable to columns selected under specified conditions.

14. The method of claim 12 or 13, wherein using a long-short term memory (LSTM) with a pointer network that formulates a select clause of the SQL query with one or more column names comprises:

given a list of column representations and a question representation, selecting the column that best matches the question.

15. The method of claim 14, wherein the list of column representations is obtained by encoding each column name with a LSTM, wherein the representation of a particular column j, e_j^c is given by

$$h_{j,t}^c = \text{LSTM}\left(\text{emb}\left(x_{j,t}^c\right), h_{j,t-1}^c\right) \qquad e_j^c = h_{j,T_j}^c$$

16. The method of one of the claims 14-15, wherein generating the one or more input representations comprises computing an input representation $\kappa^{agg}$by:

computing a scalar attention score, $\alpha^{in}{}_t = W^{inp} h_t^{enc}$, for each tth token in sequence of tokens, wherein h_t^{enc} is a state of an encoder corresponding to a tth word in the input sequence,

normalizing the vector of scores $\alpha^{in} = [\alpha_1^{inp}, \alpha_2^{inp}, \ldots]$ to produce a distribution over the tokens in the sequence of tokens,

obtaining the input representation κ^{agg} as

$$\kappa^{\text{agg}} = \sum_t \beta_t^{\text{inp}} h_t^{\text{enc}}$$

wherein $\beta^{\text{agg}} = \text{softmax}(\alpha^{\text{agg}})$, and $\alpha^{\text{agg}} = W^{\text{agg}} \tanh(V^{\text{agg}} \kappa^{\text{agg}} + b^{\text{agg}}) + c^{\text{agg}}$

17. The method of one of the claims 14-16, further comprising:

using reinforcement learning to formulate where conditions of the SQL query by executing the generated SQL query against the database to obtain a reward $R(q(\hat{y}), q_g)$ defined as:

$$R(q(y), q_g) = \begin{cases} -2, & \text{if } q(y) \text{ is not a valid SQL query} \\ -1, & \text{if } q(y) \text{ is a valid SQL query and executes to an incorrect result} \\ +1, & \text{if } q(y) \text{ is a valid SQL query and executes to the correct result} \end{cases}$$

where $q(y)$ denotes the query generated by the model, and q_g denotes the ground truth query corresponding to the input natural language query.

18. A non-transitory computer-readable storage medium comprising computer executable code that when executed by one or more computers causes the one or more computers to perform the steps of a method according to any of the preceding claims.

19. A computer system for generating a recurrent neural network (RNN) architecture, the computer system comprising:

one or more computer processors; and

a non-transitory computer-readable storage medium according to claim 18.

20. A computer system comprising one or more computer processors and at least one non-transitory storage medium and further comprising:

means for receiving an input natural language query based on data stored using a database schema;

means for generating a sequence of tokens from a plurality of terms comprising:

terms of the input natural language query,

a set of columns of the database schema, and

vocabulary of a database query language;

means for generating one or more input representations, each input representation obtained by encoding the sequence of tokens;

means for accessing a plurality of machine learning based models, each model configured to predict a portion of a database query corresponding to the input natural language query;

means for performing, for each of the plurality of models, executing the model based on an input representation to generate a portion of the database query;

means for combining the generated portions of the database query to obtain the database query; and

means for executing the database query to obtain a result set.

21. A computer system comprising one or more computer processors and at least one non-transitory storage medium and further comprising:

means processing a natural language request to generate parts of an SQL query, including:

means for using a long-short term memory (LSTM) with a pointer network that formulates a select clause of the SQL query with one or more column names; and

means for using an augmented pointer decoder trained by reinforcement learning to formulate where conditions of the SQL query.

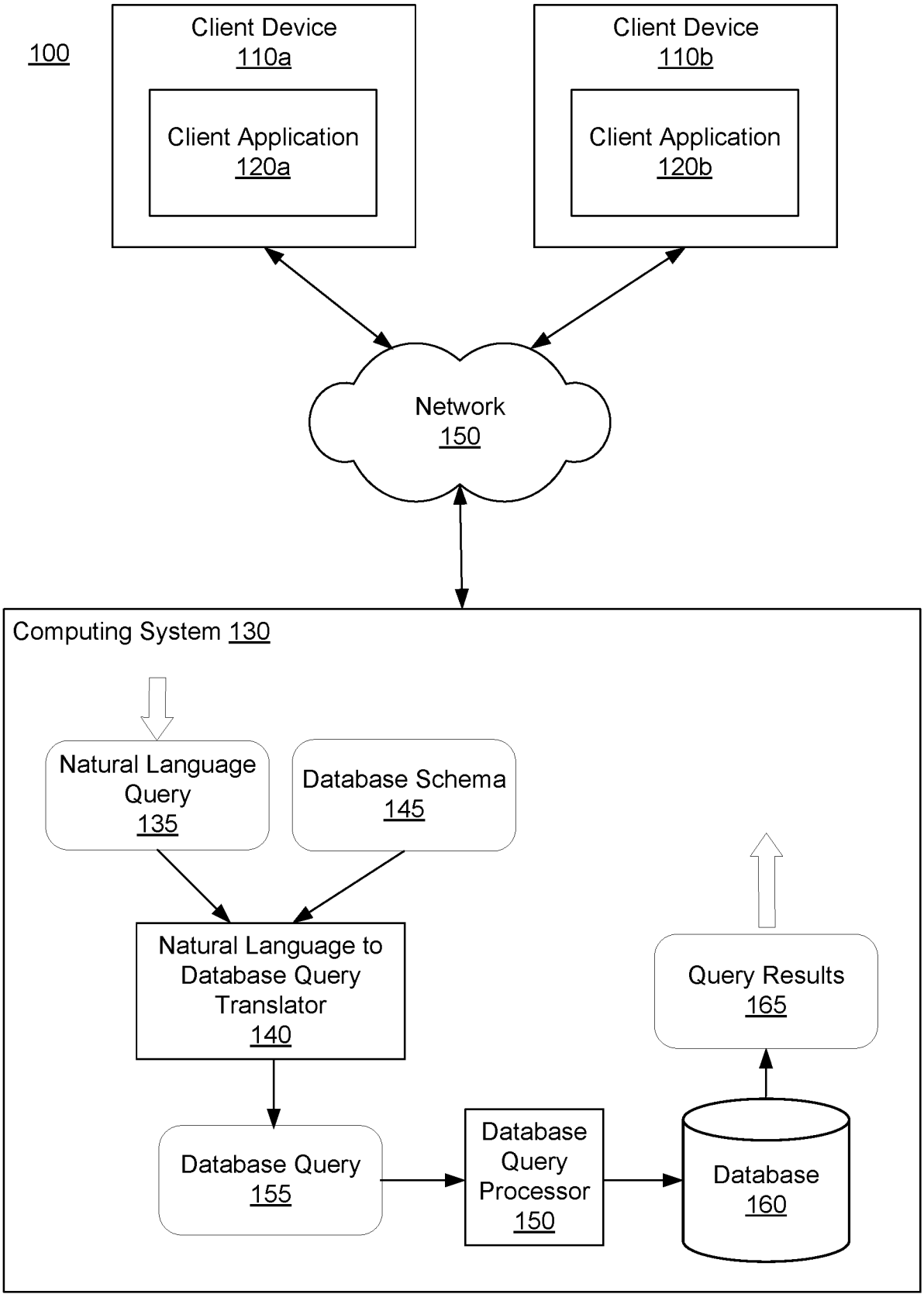
100
Client Device
110a
Client Application
120a
Client Device
110b
Client Application
120b
Network
150
Computing System 130
Natural Language
Query
135
Database Schema
145
Natural Language to
Database Query
Translator
140
Database Query
155
Database
Query
Processor
150
Database
160
Query Results
165

FIG. 1

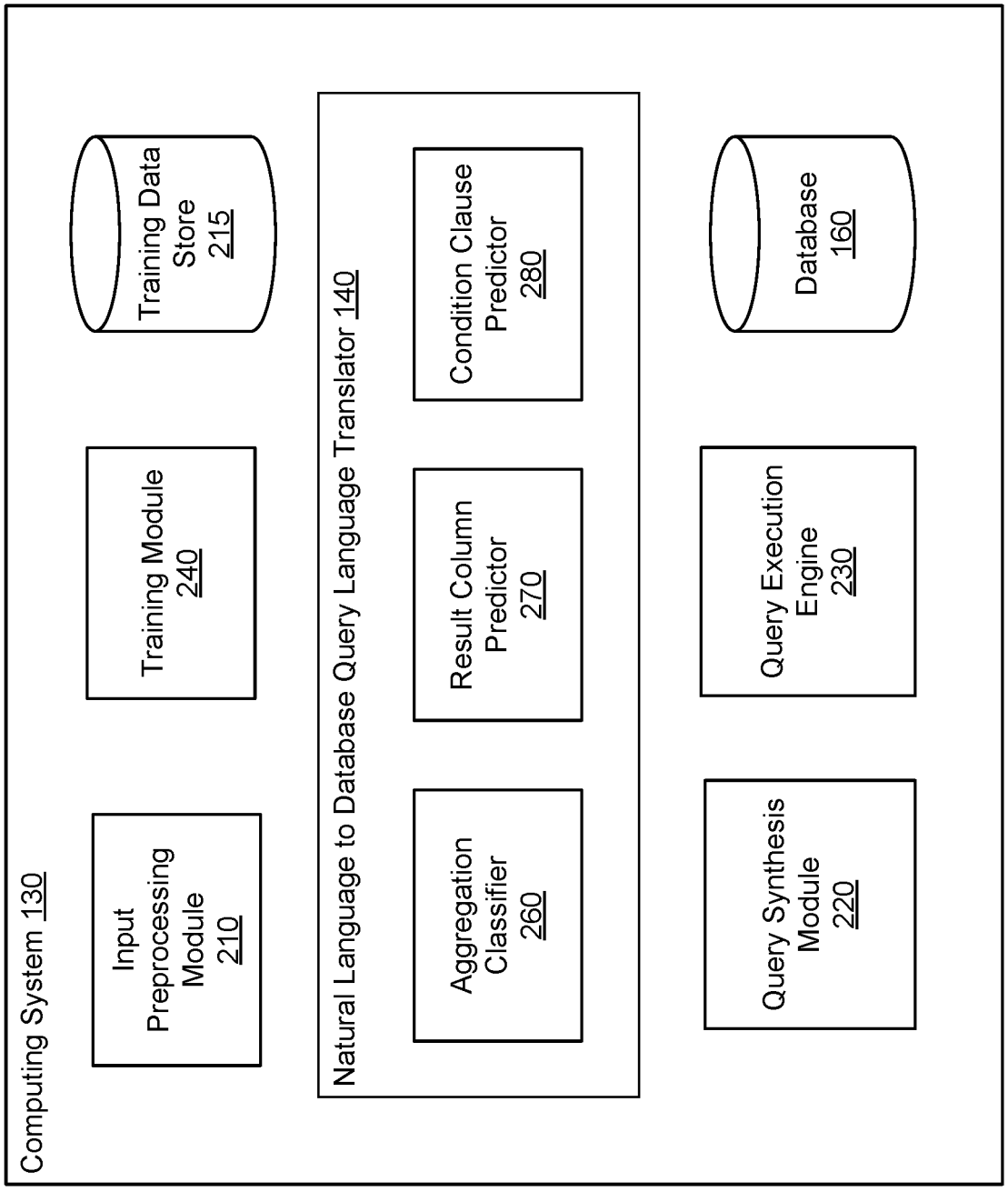
Computing System 130
Input Preprocessing Module 210
Training Module 240
Training Data Store 215
Natural Language to Database Query Language Translator 140
Aggregation Classifier 260
Result Column Predictor 270
Condition Clause Predictor 280
Query Synthesis Module 220
Query Execution Engine 230
Database 160

FIG. 2

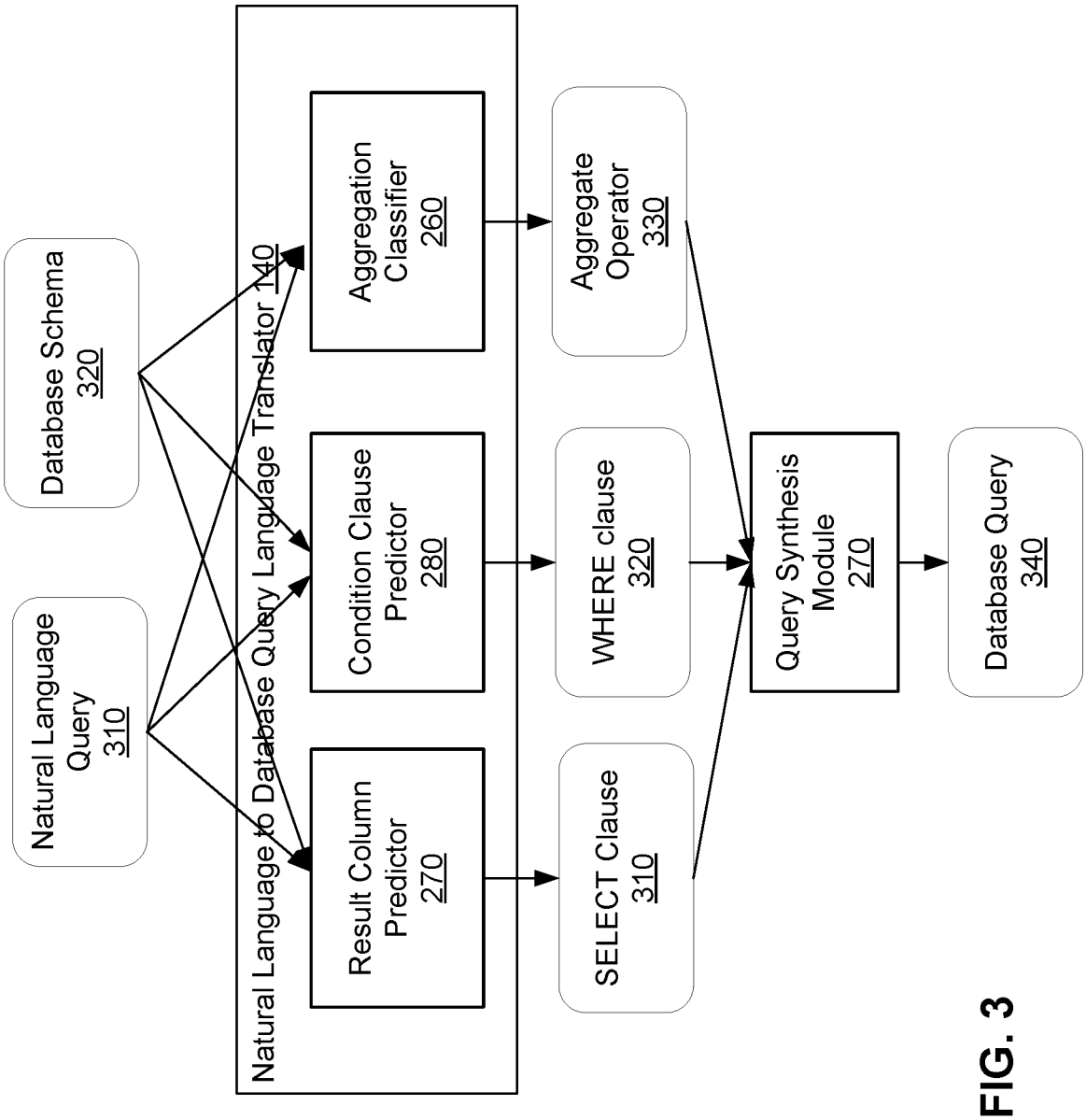
Natural Language Query 310
Database Schema 320
Natural Language to Database Query Language Translator 140
Result Column Predictor 270
Condition Clause Predictor 280
Aggregation Classifier 260
SELECT Clause 310
WHERE clause 320
Aggregate Operator 330
Query Synthesis Module 270
Database Query 340

FIG. 3

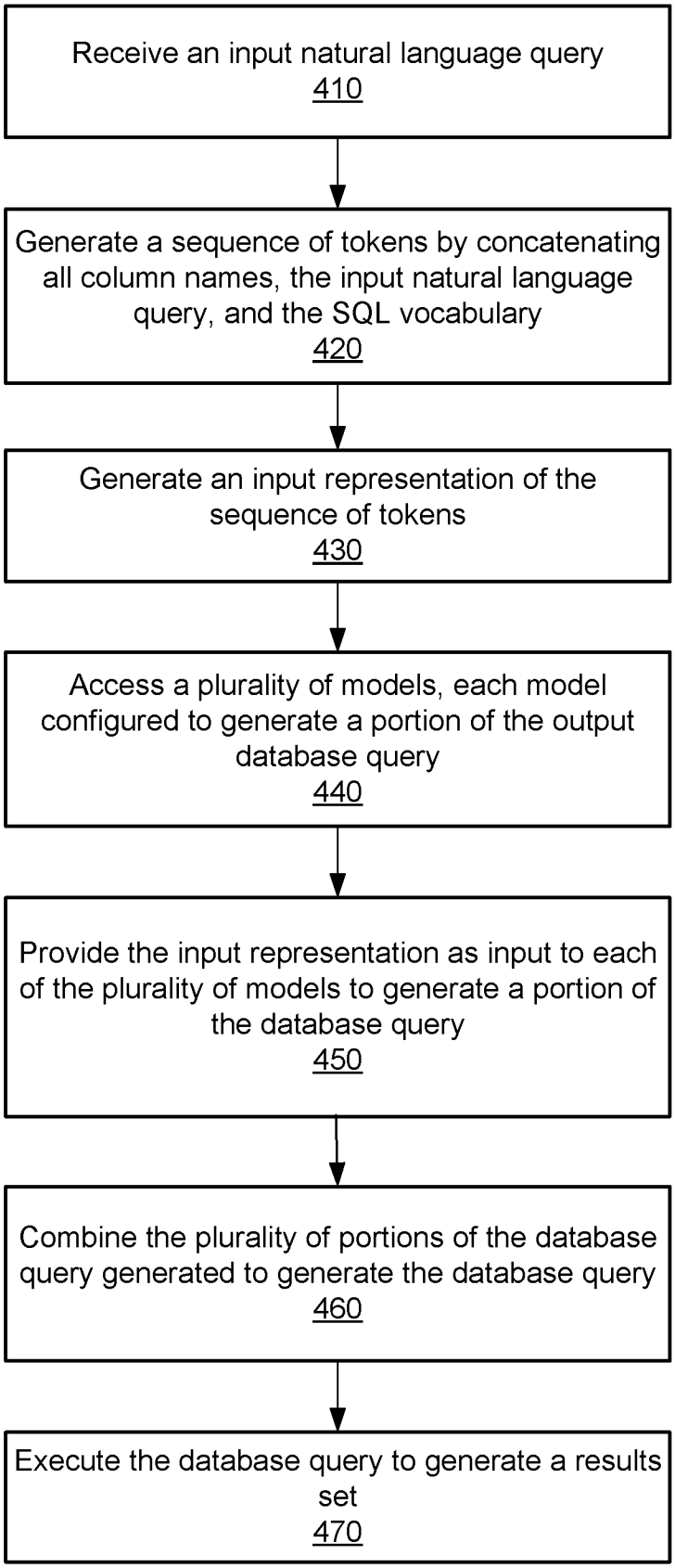
Receive an input natural language query
410
Generate a sequence of tokens by concatenating all column names, the input natural language query, and the SQL vocabulary
420
Generate an input representation of the sequence of tokens
430
Access a plurality of models, each model configured to generate a portion of the output database query
440
Provide the input representation as input to each of the plurality of models to generate a portion of the database query
450
Combine the plurality of portions of the database query generated to generate the database query
460
Execute the database query to generate a results set
470

FIG. 4

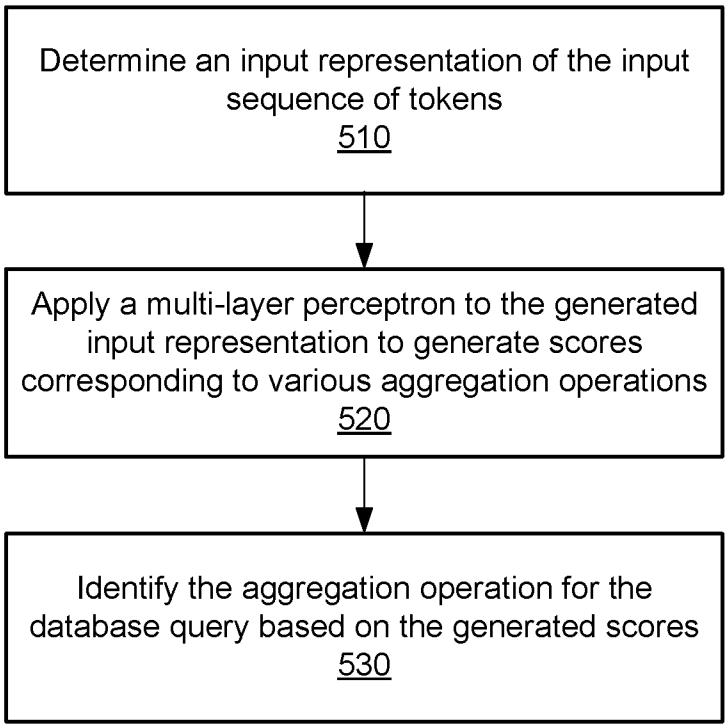
Determine an input representation of the input sequence of tokens
510
Apply a multi-layer perceptron to the generated input representation to generate scores corresponding to various aggregation operations
520
Identify the aggregation operation for the database query based on the generated scores
530

FIG. 5

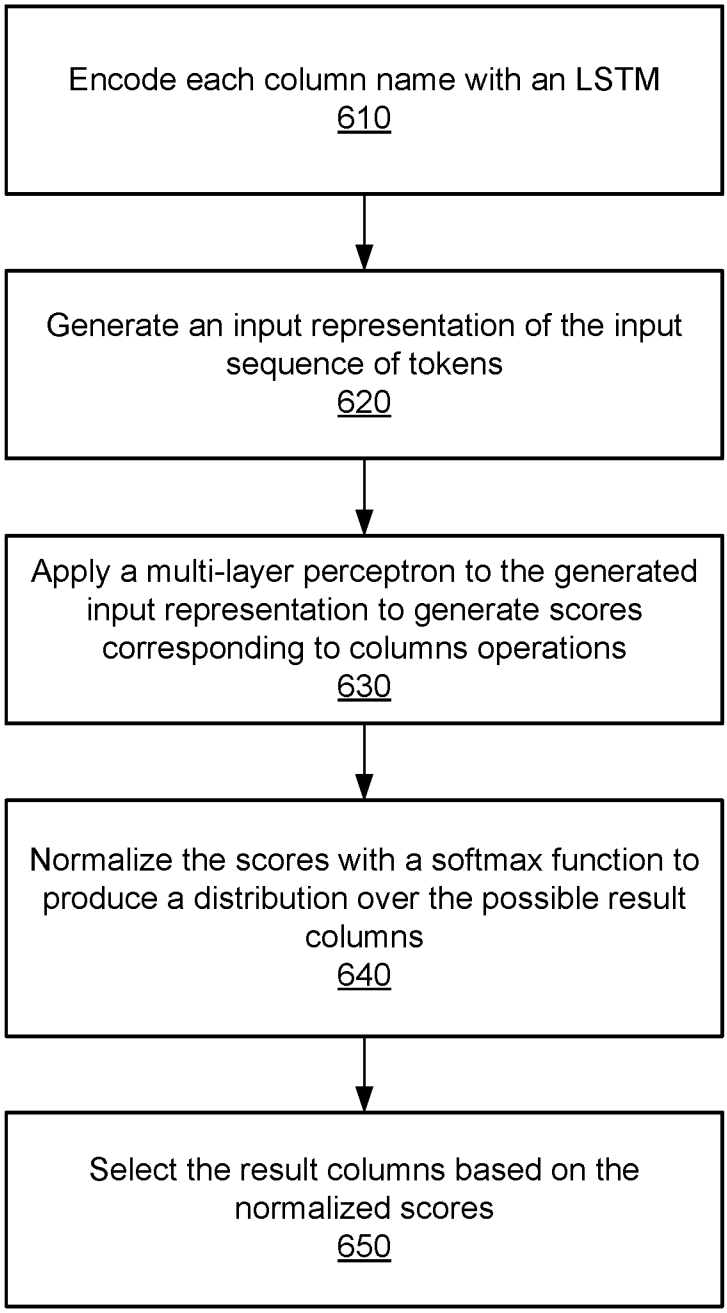
Encode each column name with an LSTM
610
Generate an input representation of the input sequence of tokens
620
Apply a multi-layer perceptron to the generated input representation to generate scores corresponding to columns operations
630
Normalize the scores with a softmax function to produce a distribution over the possible result columns
640
Select the result columns based on the normalized scores
650

FIG. 6

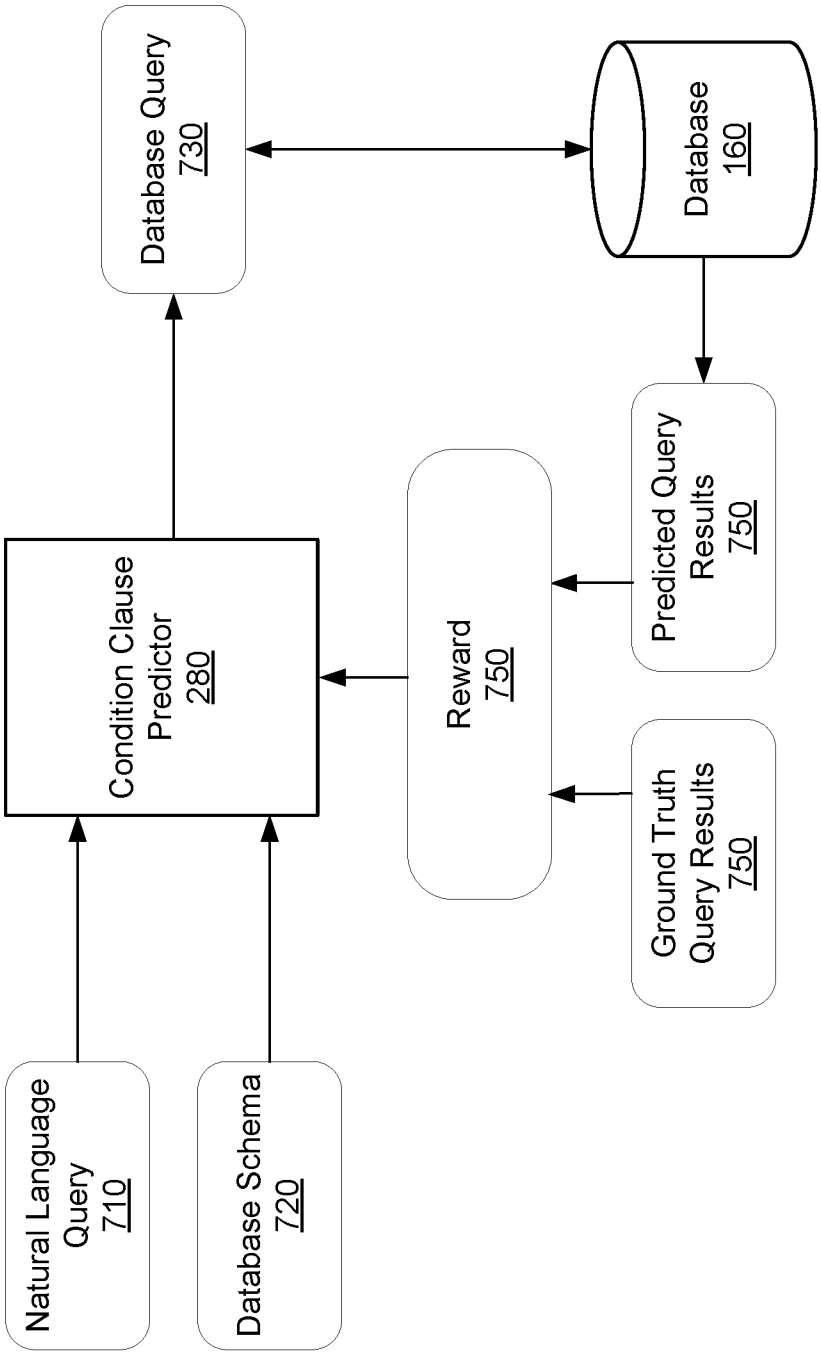
Natural Language Query 710
Database Schema 720
Condition Clause Predictor 280
Database Query 730
Database 160
Predicted Query Results 750
Reward 750
Ground Truth Query Results 750

FIG. 7

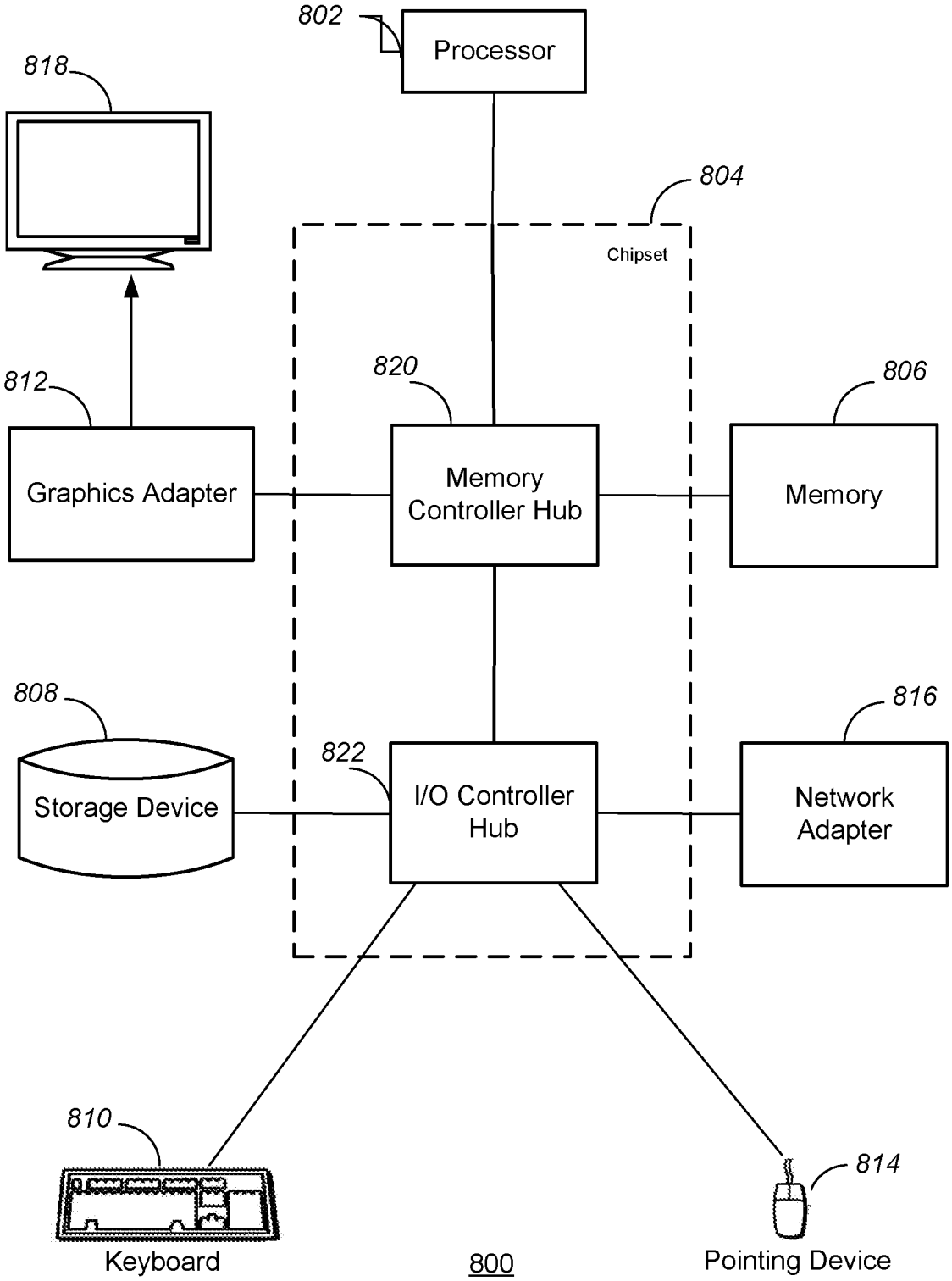
802
Processor
818
804
Chipset
812
820
806
Graphics Adapter
Memory Controller Hub
Memory
808
816
822
Storage Device
I/O Controller Hub
Network Adapter
810
814
Keyboard
800
Pointing Device

FIG. 8