(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第6929971号
(P6929971)
(45)発行日　令和3年9月1日(2021.9.1)　(24)登録日　令和3年8月13日(2021.8.13)

(51)Int.Cl.　　F I
G06N 3/08 (2006.01)　　G06N 3/08
G06F 16/2452 (2019.01)　　G06F 16/2452

請求項の数 19　(全 37 頁)

(21)出願番号　特願2019-563399(P2019-563399)
(86)(22)出願日　平成30年5月17日(2018.5.17)
(65)公表番号　特表2020-520516(P2020-520516A)
(43)公表日　令和2年7月9日(2020.7.9)
(86)国際出願番号　PCT/US2018/033099
(87)国際公開番号　WO2018/213530
(87)国際公開日　平成30年11月22日(2018.11.22)
審査請求日　令和1年11月15日(2019.11.15)
(31)優先権主張番号　15/885,613
(32)優先日　平成30年1月31日(2018.1.31)
(33)優先権主張国・地域又は機関
米国(US)
(31)優先権主張番号　62/508,367
(32)優先日　平成29年5月18日(2017.5.18)
(33)優先権主張国・地域又は機関
米国(US)

(73)特許権者　506332063
セールスフォース　ドット　コム　インコーポレイティッド
アメリカ合衆国　カリフォルニア州　９４１０５，サン　フランシスコ，　ミッション　ストリート　４１５，　サード　フロアー
(74)代理人　100107766
弁理士　伊東　忠重
(74)代理人　100070150
弁理士　伊東　忠彦
(74)代理人　100091214
弁理士　大貫　進介

最終頁に続く

(54)【発明の名称】自然言語クエリのデータベースクエリへのニューラルネットワークに基づく翻訳

(57)【特許請求の範囲】
【請求項１】
１つ以上のコンピュータを含むコンピュータシステムにより実行される方法であって、前記コンピュータシステムは、
データベーススキーマを使用して記憶されたデータに基づく入力自然言語クエリを受信するステップと、
前記入力自然言語クエリのターム、
前記データベーススキーマの列のセット、及び
データベースクエリ言語の語彙
を含む複数のタームからトークンのシーケンスを生成するステップと、
１つ以上の入力表現を生成するステップであり、各入力表現は、前記トークンのシーケンスをエンコードすることにより取得される、ステップと、
複数の機械学習に基づくモデルにアクセスするステップであり、各々の機械学習に基づくモデルは、前記入力自然言語クエリに対応するデータベースクエリの部分を予測するように構成される、ステップと、
前記複数の機械学習に基づくモデルの各々について、前記機械学習に基づくモデルを入力表現を使用して実行して前記データベースクエリの部分を生成するステップと、
前記データベースクエリの前記生成された部分を組み合わせて前記データベースクエリを取得するステップと、
前記データベースクエリを実行して結果セットを取得するステップと、

を実行する、方法。
【請求項２】
前記入力自然言語クエリは、クライアントデバイスから受信され、当該方法は、前記結果セットを前記クライアントデバイスに送信するステップをさらに含む、請求項１に記載の方法。
【請求項３】
前記複数の機械学習に基づくモデルは、前記データベースクエリ内の集約演算子を決定する集約分類器モデルを含み、前記集約分類器モデルは、多層パーセプトロンを含む、請求項１に記載の方法。
【請求項４】
前記複数の機械学習に基づくモデルは、前記データベースクエリの結果列を決定する結果列予測器モデルを含み、前記結果列予測器モデルは、多層パーセプトロンを含む、請求項１に記載の方法。
【請求項５】
前記複数の機械学習に基づくモデルは、前記データベースクエリの条件句を決定する条件句予測器モデルを含み、前記条件句予測器モデルは、強化学習に基づく、請求項１に記載の方法。
【請求項６】
グラウンドトルースデータベースクエリに基づく結果セットを受信するステップであり、前記グラウンドトルースデータベースクエリは、前記入力自然言語クエリに対応し、かつ実行の結果としてグラウンドトルースを提供するクエリを表す、ステップと、
前記生成されたクエリから取得された結果と前記グラウンドトルースデータベースクエリから取得された結果との比較に基づいて報酬値を決定するステップと、
前記報酬値に基づいて前記条件句予測器モデルの重みを調整するステップと、
をさらに含む請求項５に記載の方法。
【請求項７】
前記シーケンスの各トークンに対応する列エンコーディングを決定するステップと、
前記シーケンスの各トークンについてのスコアのベクトルを決定するステップと、
前記ベクトルをソフトマックス関数を使用して正規化するステップと、
前記入力表現を、対応する正規化されたスコアにより重み付けされた前記列エンコーディングの和として決定するステップと、
をさらに含む請求項５に記載の方法。
【請求項８】
前記複数の機械学習に基づくモデルを勾配降下を使用して訓練して、前記複数のモデルの各々の結果に基づいて損失を表す目的関数を最小化するステップ
をさらに含む請求項１に記載の方法。
【請求項９】
前記１つ以上の入力表現を生成するステップは、入力表現κ^{agg}を、
トークンのシーケンス内の各ｔ番目のトークンについて、スコア$\alpha^{inp}_t = W^{inp} h^{enc}_t$を計算し、$h^{enc}_t$は、前記シーケンス内のｔ番目のワードに対応するエンコーダの状態であり、
前記スコアのベクトル$\alpha^{inp} = [\alpha^{inp}_1, \alpha^{inp}_2, \cdots]$を正規化して、前記トークンのシーケンス内の前記トークンにわたる分布を生成し、
前記入力表現κ^{agg}を、
【数１】

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

として取得し、$\beta^{agg} = \mathrm{softmax}(\alpha^{agg})$及び$\alpha^{agg} = W^{agg} \tanh$

（$V^{agg}\kappa^{agg}+b^{agg}$）＋$c^{agg}$である
ことにより計算するステップを含む、請求項１に記載の方法。
【請求項１０】
１つ以上の列名を有する前記データベースクエリのｓｅｌｅｃｔ句を定式化するポインタネットワークで長短期記憶（ＬＳＴＭ）を使用するステップであり、
列表現のリスト及び質問表現を所与として、質問に最もマッチする列を選択することを含み、前記列表現のリストは、各列名をＬＳＴＭでエンコードすることにより取得され、特定の列ｊの表現$e^{c}{}_{j}$は、
【数２】

$$h^{c}_{j,t}=\mathrm{LSTM}\left(\mathrm{emb}\left(x^{c}_{j,t}\right),h^{c}_{j,t-1}\right)\qquad e^{c}_{j}=h^{c}_{j,T_j}$$

により与えられる、ステップ、
をさらに含む請求項１に記載の方法。
【請求項１１】
強化学習を使用して、データベースに対して前記生成されたデータベースクエリを実行することにより前記データベースクエリのｗｈｅｒｅ条件を定式化して、
【数３】

$$R(q(y),q_g)=\begin{cases}-2, & \text{q(y)が有効なデータベースクエリでない場合}\\ -1, & \text{q(y)が有効なデータベースクエリであり、不正確な結果に対し実行する場合}\\ +1, & \text{q(y)が有効なデータベースクエリであり、正確な結果に対し実行する場合}\end{cases}$$

として定義される報酬Ｒ（ｑ（ｙ），q_g）を取得するステップであり、ｑ（ｙ）は、前記機械学習に基づくモデルにより生成されたクエリを表し、q_gは、グラウンドトルースクエリを表し、前記グラウンドトルースクエリは、前記入力自然言語クエリに対応し、かつ実行の結果としてグラウンドトルースを提供するクエリを表す、ステップ
をさらに含む請求項１に記載の方法。
【請求項１２】
自然言語要求を処理してＳＱＬクエリの部分を生成するステップであり、
１つ以上の列名を有する前記ＳＱＬクエリのｓｅｌｅｃｔ句を定式化するポインタネットワークで長短期記憶（ＬＳＴＭ）を使用することと、
強化学習により訓練されたモデルを使用して前記ＳＱＬクエリのｗｈｅｒｅ条件を定式化することと、を含む、ステップ、
を含む方法。
【請求項１３】
多層パーセプトロンを使用して、指定された条件下で選択された列に適用可能な前記ＳＱＬクエリの集約演算を決定するステップ、
をさらに含む請求項１２に記載の方法。
【請求項１４】
１つ以上の列名を有する前記ＳＱＬクエリのｓｅｌｅｃｔ句を定式化するポインタネットワークで長短期記憶（ＬＳＴＭ）を使用することは、
列表現のリスト及び質問表現を所与として、質問に最もマッチする列を選択すること
を含む、請求項１２に記載の方法。
【請求項１５】
前記列表現のリストは、各列名をＬＳＴＭでエンコードすることにより取得され、特定の列ｊの表現$e^{c}{}_{j}$は、

【数４】

$$h^e_{j,t} = \mathrm{LSTM}\left(\mathrm{emb}\left(x^e_{j,t}\right), h^e_{j,t-1}\right) \qquad e^e_j = h^e_{j,T_j}$$

により与えられる、請求項１４に記載の方法。

【請求項１６】

前記１つ以上の入力表現を生成することは、入力表現κ^{agg}を、

トークンのシーケンス内の各ｔ番目のトークンについて、スコアα$^{inp}_{t}$＝W^{inp}ｈ$^{enc}_{t}$を計算し、ｈ$^{enc}_{t}$は、前記シーケンス内のｔ番目のワードに対応するエンコーダの状態であり、

前記スコアのベクトルα^{inp}＝［α$^{inp}_{1}$，α$^{inp}_{2}$，・・・］を正規化して、前記トークンのシーケンス内の前記トークンにわたる分布を生成し、

前記入力表現κ^{agg}を、

【数５】

$$\kappa^{\mathrm{agg}} = \sum_t \beta^{\mathrm{inp}}_t h^{\mathrm{enc}}_t$$

として取得し、β^{agg}＝ｓｏｆｔｍａｘ（α^{agg}）及びα^{agg}＝W^{agg}ｔａｎｈ（Ｖ^{agg}κ^{agg}＋ｂ^{agg}）＋ｃ^{agg}である

ことにより計算することを含む、請求項１４に記載の方法。

【請求項１７】

強化学習を使用して、データベースに対して前記生成されたＳＱＬクエリを実行することにより前記ＳＱＬクエリのｗｈｅｒｅ条件を定式化して、

【数６】

$$R(q(y), q_g) = \begin{cases} -2, & \text{q（y）が有効なＳＱＬクエリでない場合} \\ -1, & \text{q（y）が有効なＳＱＬクエリであり、不正確な結果に対し実行する場合} \\ +1, & \text{q（y）が有効なＳＱＬクエリであり、正確な結果に対し実行する場合} \end{cases}$$

として定義される報酬Ｒ（ｑ（ｙ），ｑ$_{g}$）を取得するステップであり、ｑ（ｙ）は、機械学習に基づくモデルにより生成されたクエリを表し、ｑ$_{g}$は、グラウンドトルースクエリを表し、前記グラウンドトルースクエリは、入力自然言語クエリに対応し、かつ実行の結果としてグラウンドトルースを提供するクエリを表す、ステップ

をさらに含む請求項１４に記載の方法。

【請求項１８】

１つ以上のコンピュータにステップを実行させるコンピュータプログラムであって、前記ステップは、

データベーススキーマを使用して記憶されたデータに基づく入力自然言語クエリを受信するステップと、

前記入力自然言語クエリのターム、

前記データベーススキーマの列のセット、及び

データベースクエリ言語の語彙

を含む複数のタームからトークンのシーケンスを生成するステップと、

１つ以上の入力表現を生成するステップであり、各入力表現は、前記トークンのシーケンスをエンコードすることにより取得される、ステップと、

複数の機械学習に基づくモデルにアクセスするステップであり、各々の機械学習に基づくモデルは、前記入力自然言語クエリに対応するデータベースクエリの部分を予測するように構成される、ステップと、

前記複数の機械学習に基づくモデルの各々について、前記機械学習に基づくモデルを入力表現を使用して実行して前記データベースクエリの部分を生成するステップと、

前記データベースクエリの前記生成された部分を組み合わせて前記データベースクエリを取得するステップと、
前記データベースクエリを実行して結果セットを取得するステップと、
を含む、コンピュータプログラム。
【請求項１９】
リカレントニューラルネットワーク（ＲＮＮ）アーキテクチャを生成するコンピュータシステムであって、
１つ以上のコンピュータプロセッサと、
１つ以上のコンピュータにステップを実行させるコンピュータプログラムと、
を含み、前記ステップは、
データベーススキーマを使用して記憶されたデータに基づく入力自然言語クエリを受信するステップと、
前記入力自然言語クエリのターム、
前記データベーススキーマの列のセット、及び
データベースクエリ言語の語彙
を含む複数のタームからトークンのシーケンスを生成するステップと、
１つ以上の入力表現を生成するステップであり、各入力表現は、前記トークンのシーケンスをエンコードすることにより取得される、ステップと、
複数の機械学習に基づくモデルにアクセスするステップであり、各々の機械学習に基づくモデルは、前記入力自然言語クエリに対応するデータベースクエリの部分を予測するように構成される、ステップと、
前記複数の機械学習に基づくモデルの各々について、前記機械学習に基づくモデルを入力表現を使用して実行して前記データベースクエリの部分を生成するステップと、
前記データベースクエリの前記生成された部分を組み合わせて前記データベースクエリを取得するステップと、
前記データベースクエリを実行して結果セットを取得するステップと、
を含む、コンピュータシステム。
【発明の詳細な説明】
【技術分野】
【０００１】
本開示は、一般に、データベースクエリの自動生成に関し、より具体的には、自然言語クエリをデータベースクエリに翻訳するためのニューラルネットワークに基づくモデルに関する。
【背景技術】
【０００２】
世界中で利用可能な相当量のデータは、関係データベースに記憶されている。関係データベースは、医療記録、金融市場、顧客関係管理などのアプリケーションの基礎を提供する。しかしながら、関係データベース内の情報へのアクセスは、構造化問い合わせ言語（ＳＱＬ）などのデータベースクエリ言語の理解を要する。ＳＱＬなどのデータベースクエリ言語は、ユーザが関係データベースからのデータの要求を指定することを可能にする点で強力であるが、これらは学習することが困難である。データベースクエリ言語を使用して効率的にデータベースクエリを書けるようにするには、データベースにおける専門知識と強力な技術知識を要する。
【０００３】
一部のシステムは、システムに記憶されたデータにアクセスするための自然言語をサポートしている。自然言語クエリは、人々が自然言語を使用するための訓練を必要としないため、表現の容易さを提供する。しかしながら、これらのシステムは、ＳＱＬなどのデータベースクエリ言語の表現力を提供しない。例えば、自然言語クエリは、複数の方法で解釈される可能性があり、関係データベースに記憶されたデータにアクセスするための自然言語クエリの対応する実行は、非効率的な可能性があり、要求された正確な情報を取り出

さない可能性がある。したがって、自然言語クエリ又はデータベースクエリのいずれかを使用して関係データベースに記憶されたデータにアクセスする従来の手法は、これらが表現の容易さを提供するか又は表現の力を提供するかのいずれかであり、あるいは双方を提供しないため、欠点を有する。

【図面の簡単な説明】

【０００４】

開示される実施形態は、詳細な説明、別記の特許請求の範囲、及び添付の図（又は図面）からより容易に明らかとなる他の利点及び特徴を有する。以下は、図の簡単な紹介である。

【図１】一実施形態による、自然言語クエリをデータベースクエリに翻訳するための全体的なシステム環境を示す高レベルブロック図である。

【図２】一実施形態による、自然言語クエリをデータベースクエリに翻訳するためのコンピューティングシステムのシステムアーキテクチャを示す。

【図３】一実施形態による、自然言語対データベースクエリ翻訳器により実行される処理の詳細を示す。

【図４】一実施形態による、自然言語クエリをデータベースクエリに翻訳するための全体的な処理を示す。

【図５】一実施形態による、自然言語クエリに基づいて出力データベースクエリの集約演算子を決定する集約分類器の処理を示す。

【図６】一実施形態による、自然言語クエリに基づいて出力データベースクエリのＳＥＬＥＣＴ句の列を決定する結果列予測器の処理を示す。

【図７】一実施形態による、出力データベースクエリの条件句を決定する条件句予測器を訓練する処理を示す。

【図８】図１のクライアントデバイス及び／又はコンピューティングシステムを実現するための一例示的なコンピュータを示す高レベルブロック図である。

【０００５】

図（図面）及び以下の説明は、特定の実施形態を単なる例示として説明する。当業者は、以下の説明から、本明細書に示された構造及び方法の代替的な実施形態が、本明細書で説明される原理から逸脱することなく採用され得ることを容易に理解するであろう。次に、いくつかの実施形態を詳細に参照し、その例を添付の図に示す。

【発明を実施するための形態】

【０００６】

コンピューティングシステムが、自然言語クエリ（natural language queries）を対応するデータベースクエリ、例えば構造化問い合わせ言語（structured query language、ＳＱＬ）を使用して指定されたクエリに翻訳するために、ディープニューラルネットワークを使用する。実施形態は、ＳＱＬクエリの構造を使用して、生成されたクエリの出力空間を大幅に削減する。コンピューティングシステムは、ディープニューラルネットワークを使用して、自然言語クエリをデータベースクエリに翻訳する。

【０００７】

一実施形態において、コンピューティングシステムは、複数の機械学習に基づくモデル、例えばニューラルネットワークに基づくモデルを使用して、出力データベースクエリの異なる部分を生成する。例えば、コンピューティングシステムは、データベースクエリ内の集約演算子を決定する集約分類器（aggregation classifier）モデル、データベースクエリの結果列を決定する結果列予測器（result column predictor）モデル、及びデータベースクエリの条件句を決定する条件句予測器（condition clause predictor）モデルを使用することができる。一実施形態において、集約分類器モデル及び結果列予測器モデルは、多層パーセプトロンを含む。条件句予測器モデルは、ポリシーに基づく強化学習（reinforcement learning、ＲＬ）を使用して、データベースクエリの条件句を生成する。これは、条件句が本質的に順序づけられておらず、条件句の複数の表現がデータベースクエリに対して同じ出力結果を提供する可能性があるためである。ゆえに、条件句は

、交差エントロピー損失（cross entropy loss）を使用した最適化に適さない。ディープニューラルネットワークは、交差エントロピー損失とＲＬ報酬とを組み合わせた混合された目的（mixed objective）を使用して訓練される。

【０００８】

一例として、データベースが、列Ｐｉｃｋ＿ｎｕｍｂｅｒ、ＣＦＬ＿Ｔｅａｍ、Ｐｌａｙｅｒ、Ｐｏｓｉｔｉｏｎ、及びＣｏｌｌｅｇｅを有するテーブルＣＦＬＤｒａｆｔを記憶することができる。このテーブルは、以下の例示的な行を記憶することができる。

【表１】

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
…	…	…	…	…

【０００９】

システムは、自然言語クエリ、例えば「いくつのＣＦＬチームがヨーク大学からか？（How many CFL teams are from York College?）」を受信する。システムは、テーブルＣＦＬＤｒａｆｔを含むデータベーススキーマに関連して受信した自然言語クエリを処理して、ＳＱＬ言語を使用したデータベースクエリ「ＳＥＬＥＣＴ ＣＯＵＮＴ（ＣＦＬ＿Ｔｅａｍ） ＦＲＯＭ ＣＦＬＤｒａｆｔ ＷＨＥＲＥ Ｃｏｌｌｅｇｅ＝“Ｙｏｒｋ”」を生成する。システムは、データベーススキーマを使用してデータベースクエリを実行する。テーブルＣＬＦＤｒａｆｔの２行が、これらが大学「Ｙｏｒｋ」を有するため、データベースクエリのＷＨＥＲＥ句にマッチする。結果として、システムは結果２を返す。

【００１０】

［全体的なシステム環境］

図１は、一実施形態による、自然言語クエリをデータベースクエリに翻訳するための全体的なシステム環境を示す高レベルブロック図である。システム環境１００は、ネットワーク１５０によりコンピューティングシステム１３０に接続された１つ以上のクライアントデバイス１１０を含む。コンピューティングシステム１３０は、オンラインシステムであってもよいが、例えば、自然言語クエリのセットの各々をデータベースクエリに翻訳するバッチ処理を実行することによりオフラインで動作してもよい。

【００１１】

２つのクライアントデバイス１１０ａ、１１０ｂのみがここで示されているが、これらのエンティティの各々の、複数のインスタンスが存在してもよい。例えば、いくつかのコンピューティングシステム１３０と、各コンピューティングシステム１３０と通信する数十又は数百のクライアントデバイス１１０とが存在してもよい。図面は、同様の要素を識別するために、同様の参照番号を使用する。参照番号の後の文字、「１１０ａ」などは、テキストがその特定の参照番号を有する要素を具体的に参照することを示す。続きの文字のないテキストの参照番号、「１１０」などは、その参照番号を有する図中の要素のいず

れか又は全てを参照する。

【００１２】

クライアントデバイス１１０は、ＡＮＤＲＯＩＤ（登録商標）若しくはＡＰＰＬＥ（登録商標）　ＩＯＳ（登録商標）などのオペレーティングシステムを有するスマートフォン、タブレットコンピュータ、ラップトップコンピュータ、デスクトップコンピュータ、自動車若しくは他の車両の電子ステレオ、又はデジタルコンテンツが聴けるか又は他の方法で体験できる任意の他タイプのネットワーク対応デバイスなどのコンピューティングデバイスである。典型的なクライアントデバイス１１０は、（例えば、Ｗｉｆｉ及び／又は４Ｇ又は他の無線電気通信標準を介して）ネットワーク１５０に接続するために必要なハードウェア及びソフトウェアを含む。

【００１３】

クライアントデバイス１１０は、クライアントデバイス１１０のユーザがコンピューティングシステム１３０と対話することを可能にするクライアントアプリケーション１２０を含む。例えば、クライアントアプリケーション１２０は、ユーザがコンピューティングシステム１３０に送信される自然言語クエリを入力することを可能にするユーザインターフェースであってもよい。クライアントアプリケーション１２０は、コンピューティングシステム１３０から結果を受信し、これらをユーザインターフェースを介してユーザに提示する。一実施形態において、クライアントアプリケーション１２０は、クライアントデバイス１１０のユーザがコンピューティングシステム１３０上で実行しているウェブサーバと対話することを可能にするブラウザである。

【００１４】

コンピューティングシステム１３０は、協調された機能又はタスクのグループを実行するソフトウェアを含む。ソフトウェアは、コンピューティングシステム１３０のユーザが関心のある特定のタスク又はアクティビティを実行することを可能にでき、あるいは他のソフトウェアに特定の機能性及びサービスを提供するシステムソフトウェア（例えば、オペレーティングシステム）を含んでもよい。コンピューティングシステム１３０は、クライアントデバイス１１０から要求を受信し、受信した要求に関連づけられたコンピュータプログラムを実行する。一例として、コンピューティングシステム１３０は、クライアントデバイス１１０からの、自然言語クエリをデータベースクエリに翻訳する要求に応答して、コンピュータプログラムを実行してもよい。コンピューティングシステム１３０上で実行するソフトウェアは、複数の当事者又はチームがソフトウェアの異なるコンポーネントを管理する責任を負う協調的な方法で書かれたコンピュータプログラム、ライブラリ、及び関連データの複雑なコレクションを含むことができる。

【００１５】

一実施形態において、コンピューティングシステム１３０は、クライアントデバイス１１０から自然言語クエリ１３５を受信する。自然言語クエリ１３０は、コンピューティングシステム１３０上で実行するクライアントアプリケーション１２０を介してユーザにより提供されてもよい。コンピューティングシステム１３０は、データベースに記憶されるデータの構造を定義するデータベーススキーマ１４５を記憶する。例えば、データベーススキーマ１４５は、データベースに記憶された様々なテーブル、各テーブルの列、外部キー関係などのテーブル間の関係、テーブルに関連づけられた任意の制約などを識別することができる。

【００１６】

自然言語対データベースクエリ翻訳器（natural language to database query translator）１４０は、入力として自然言語クエリ１３５及びデータベーススキーマ１４５を受信し、入力された自然言語クエリ１３５に相当するデータベースクエリ１５５を生成する。生成されたデータベースクエリ１５５は、データベーススキーマ１４５に準拠する。生成されたデータベースクエリ１５５は、データベースクエリプロセッサ１５０により受信され、データベースクエリプロセッサ１５０は、データベース１６０に記憶されたデータを使用してデータベースクエリ１５５を処理する。データベースクエリプロセッサ１

５０は、データベースクエリ１５５を処理することによりクエリ結果１６５を生成する。コンピューティングシステム１３０は、生成されたクエリ結果１６５を、自然言語クエリ１３５を送信したクライアントデバイス１１０上で実行しているクライアントアプリケーション１２０に提供する。

【００１７】

一実施形態において、自然言語対データベースクエリ翻訳器１４０は、シーケンス対シーケンス（sequence to sequence）翻訳を実行する。従来のニューラルネットワークに基づくシーケンス対シーケンス翻訳器は、かなり大きい空間で検索する。対照的に、実施形態は、データベースクエリ言語に固有の構造を活用して検索空間を削減する。詳細には、システムは、テーブルスキーマ、入力質問、及びＳＱＬキーワードの結合に基づいて、生成されたシーケンスの出力空間を制限する。一実施形態において、自然言語対データベースクエリ翻訳器１４０は、拡大された（augmented）入力を有するポインタネットワークであるディープニューラルネットワークを使用する。

【００１８】

ネットワーク１５０は、クライアントデバイス１１０とレコード管理システム１３０との間の通信インフラストラクチャを提供する。ネットワーク１５０は、典型的にはインターネットであるが、これらに限られないがローカルエリアネットワーク（ＬＡＮ）、メトロポリタンエリアネットワーク（ＭＡＮ）、ワイドエリアネットワーク（ＷＡＮ）、モバイル有線若しくは無線ネットワーク、プライベートネットワーク、又は仮想プライベートネットワークを含む任意のネットワークであってもよい。ネットワーク１５０の一部が、ＩＥＥＥ８０２．１１標準に基づくＷｉＦｉ、ＢＬＵＥＴＯＯＴＨ（登録商標）ショートレンジ標準、及びワイヤレスユニバーサルシリアルバス（ＵＳＢ）標準を含む通信技術を使用するリンクにより提供されてもよい。

【００１９】

［システムアーキテクチャ］

図２は、一実施形態による、自然言語クエリをデータベースクエリに翻訳するためのコンピューティングシステムのシステムアーキテクチャを示す。コンピューティングシステム１３０は、入力エンコーディングモジュール２１０、訓練モジュール２４０、自然言語対データベースクエリ翻訳器１４０、クエリ合成モジュール２２０、クエリ実行エンジン２３０、訓練データストア２１５、及びデータベース１６０を含む。ネットワークインターフェース、セキュリティ機能、ロードバランサ、フェイルオーバサーバ、管理及びネットワーク操作コンソールなどの従来のコンポーネントは、システムアーキテクチャの詳細を分かりにくくしないよう示されていない。

【００２０】

入力前処理モジュール２１０は、自然言語対データベースクエリ翻訳器１４０への入力として提供するために入力データを前処理する。一実施形態において、入力前処理モジュール２１０は、データベーススキーマからの列名と、入力自然言語クエリと、データベースクエリ言語、例えばＳＱＬの語彙（vocabulary）とを連結することにより、トークンのシーケンスを生成する（４２０）。入力前処理モジュール２１０は、出力データベースクエリの様々な部分を生成する様々なモデルに提供するために１つ以上の入力表現を生成する。

【００２１】

自然言語対データベースクエリ翻訳器１４０は、自然言語クエリに対応するデータベースクエリを生成するために入力自然言語クエリを処理する。一実施形態において、自然言語対データベースクエリ翻訳器１４０は、図３に関連して本明細書でさらに説明される他のコンポーネント、例えば、集約分類器２６０、結果列予測器２７０、及び条件句予測器２８０を含む。

【００２２】

自然言語対データベースクエリ翻訳器１４０は、異なるニューラルネットワークを使用してデータベースクエリの異なるコンポーネントを生成する。一実施形態において、自然

言語対データベースクエリ翻訳器１４０は、異なるニューラルネットワークを使用して、選択列と集約演算子とｗｈｅｒｅ句とを含むデータベースクエリのコンポーネントを生成する。

【００２３】

訓練モジュール２４０は、訓練データストア２１５に記憶された過去データを使用して、自然言語対データベースクエリ翻訳器１４０内のニューラルネットワークを訓練する。一実施形態において、訓練モジュール２４０は、集約分類器２６０及び結果列予測器２７０を交差エントロピー損失を使用して訓練するが、条件句予測器２８０をポリシー勾配（policy gradient）強化学習を使用して訓練して、クエリ条件の順序づけられていない性質に対処する。ＳＱＬクエリの構造を利用することで、自然言語対データベースクエリ翻訳器１４０がデータベースクエリの出力空間を削減することを可能にする。これは、クエリ構造を活用しない他の手法と比較して、有意により高い性能をもたらす。

【００２４】

クエリ合成モジュール２２０は、自然言語対データベースクエリ翻訳器１４０により生成されたデータベースクエリの様々なコンポーネントを受信し、これらを組み合わせてデータベースクエリを取得する。クエリ実行モジュール２３０は、データベース１６０に記憶されたデータを使用して、クエリ合成モジュール２２０により提供されたデータベースクエリを実行する。コンピューティングシステム１３０は、クエリの実行の結果を、結果の要求元、例えばクライアントデバイス１１０上で実行しているクライアントアプリケーション１２０に返す。

【００２５】

図３は、一実施形態による、自然言語対データベースクエリ翻訳器１４０により実行される処理の詳細を示す。図３に示すように、自然言語対データベースクエリ翻訳器１４０への入力は、自然言語クエリ３２０及びデータベーススキーマ３２０を含む。ＣＦＬＤｒａｆｔテーブルに基づく上述の例では、自然言語クエリ３２０は、「いくつのＣＦＬチームがヨーク大学からか？」であり、データベーススキーマ３２０は、列Ｐｉｃｋ＿ｎｕｍｂｅｒ、ＣＦＬ＿Ｔｅａｍ、Ｐｌａｙｅｒ、Ｐｏｓｉｔｉｏｎ、及びＣｏｌｌｅｇｅを含む様々な列を含む。例示的な出力データベースクエリは、「ＳＥＬＥＣＴ　ＣＯＵＮＴ（ＣＦＬ＿Ｔｅａｍ）　ＦＲＯＭ　ＣＦＬＤｒａｆｔ　ＷＨＥＲＥ　Ｃｏｌｌｅｇｅ＝“Ｙｏｒｋ”」である。

【００２６】

入力前処理モジュール２１０は、１つ以上の入力表現を生成し、集約分類器２６０、結果列予測器２７０、及び条件句予測器２８０を含む自然言語対データベースクエリ翻訳器１４０の各コンポーネントに入力表現を提供する。集約分類器２６０、結果列予測器２７０、及び条件句予測器２８０の各々は、出力データベースクエリの一部を生成する。

【００２７】

結果列予測器２７０は、結果列、例えば、ＳＱＬを使用して表現された出力データベースクエリのＳＥＬＥＣＴ句３１０で指定される列を生成する。結果列の一例が、例示的な出力データベースクエリ内の列ＣＦＬ＿Ｔｅａｍである。一実施形態において、結果列予測器２７０は、列のシーケンスのエンコーディングを入力として受信し、かつＳＥＬＥＣＴ列に対応する該列のシーケンス内の列を指し示すポインタネットワークである。

【００２８】

条件句予測器２８０は、出力データベースクエリの出力行をフィルタリングするために使用される条件を指定する出力データベースクエリのＷＨＥＲＥ句３２０を生成する。上記の例では、ＷＨＥＲＥ句「Ｃｏｌｌｅｇｅ＝“Ｙｏｒｋ”」が出力データベースクエリ内の条件句である。

【００２９】

集約分類器２６０は、もしあれば出力データベースクエリ内の集約演算子３３０、例えば、例示的な出力データベースクエリ内のＣＯＵＮＴ演算子を生成する。集約演算子は、ＳＱＬにより選択された行の要約を生成する。集約分類器２６０により生成され得る集約

演算子の例は、最大（ＭＡＸ）、最小（ＭＩＮ）、平均（ＡＶＧ）、和（ＳＵＭ）などを含む。集約分類器２６０は、出力クエリに集約演算子が存在しない場合、ＮＵＬＬ集約演算子を生成することができる。

【００３０】

ＳＥＬＥＣＴ句３１０、ＷＨＥＲＥ句３２０、及び集約演算子３３０を含む出力データベースクエリの様々なコンポーネントは、クエリ合成モジュール２７０に入力として提供される。クエリ合成モジュール２７０は、出力データベースクエリの個々のコンポーネントを組み合わせて、完全な出力データベースクエリ３４０を生成する。

【００３１】

［全体的な処理］

図４～図７は、自然言語クエリをデータベースクエリに翻訳するための様々な処理を示す。当業者は、他の実施形態が図４～図７のステップをフローチャートに示される順序と異なる順序で実行できることを認識するであろう。さらに、他の実施形態は、本明細書で説明されるステップと異なる及び／又は追加のステップを含むことができる。特定のモジュールにより実行されるものとして示されるステップは、他のモジュールにより実行されてもよい。

【００３２】

図４は、一実施形態による、自然言語クエリをデータベースクエリに翻訳するための全体的な処理を示す。自然言語対データベースクエリ翻訳器１４０が、入力自然言語クエリを受信する（４１０）。入力前処理モジュール２１０が、データベーススキーマからの列名と、入力自然言語クエリと、データベースクエリ言語の語彙、例えば、ＳＥＬＥＣＴ、ＦＲＯＭ、ＷＨＥＲＥなどのＳＱＬ言語の様々なキーワードとを連結することにより、トークンのシーケンスを生成する（４２０）。例えば、式（１）は、列名x_i^cと、ＳＱＬ語彙を表すターム（terms）x^sと、入力自然言語クエリを表すタームx^qとを含むトークンのシーケンスを示す。

【数１】

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

【００３３】

式（１）において、シーケンスａ及びｂの間の連結は［ａ；ｂ］で表される。さらに、組み合わせられたシーケンスｘは、境界を画定するために、近隣のシーケンスの間にセンチネルトークンを含む。例えば、トークン＜ｃｏｌ＞は列名を識別し、トークン＜ｓｑｌ＞はＳＱＬ語彙を表すタームを識別し、トークン＜ｑｕｅｓｔｉｏｎ＞は入力自然言語クエリのタームを識別する。

【００３４】

入力前処理モジュール２１０は、トークンのシーケンスの入力表現を生成する（４３０）。一実施形態において、入力前処理モジュール２１０は、複数のモデルの各々について１つで、複数の入力表現を生成する。

【００３５】

自然言語対データベースクエリ翻訳器１４０は、複数のニューラル機械学習モデルにアクセスし、各モデルは、出力データベースクエリの部分を生成するように構成される。一実施形態において、自然言語対データベースクエリ翻訳器１４０は、複数の訓練されたニューラルネットワークに基づくモデルをストレージデバイスからメモリにロードする。自然言語対データベースクエリ翻訳器１４０は、複数の機械学習に基づくモデルの各々に入力表現を提供する（４５０）。複数の機械学習に基づくモデルの各々は、データベースクエリの部分を生成する。

【００３６】

いくつかの実施形態において、入力前処理モジュール２１０は、複数の入力表現を生成し、自然言語対データベースクエリ翻訳器１４０は、各々の機械学習に基づくモデルに異なる入力表現を提供することができる。各々の機械学習に基づくモデルは、データベース

クエリの部分を生成し、それをクエリ合成モジュール２７０に提供する。クエリ合成モジュール２７０は、データベースクエリの複数の部分を組み合わせてフルのデータベースクエリを生成する（４６０）。クエリ実行エンジン２３０が、データベースクエリを実行して結果セットを生成する（４７０）。

【００３７】

［集約分類器］

図５は、一実施形態による、自然言語クエリに基づいて出力データベースクエリの集約演算子を決定する集約分類器の処理を示す。集約分類器２６０は、入力自然言語クエリで指定された質問のタイプに基づいて、出力データベースクエリの集約演算子を決定する。例えば、集約分類器２６０は、文字列「いくつの（how many）」を含む入力質問を集約演算子ＣＯＵＮＴにマッピングすることができ、集約分類器２６０は、「何が最も高い」を含む入力質問を集約演算子最大にマッピングすることができ、集約分類器２６０は、「何が最も小さい」を含む入力質問を集約演算子最小にマッピングすることができる、などである。

【００３８】

集約分類器２６０は、トークンの入力シーケンスの入力表現を決定する（５１０）。集約分類器２６０は、入力シーケンス内の各ｔ番目のトークンについて、スカラー注目スコア（scalar attention score）$\alpha_t^{inp} = W^{inp} * h_t^{enc}$を計算する。したがって、集約分類器２６０は、スコアのベクトル$\alpha^{inp} = [\alpha_1^{inp}, \alpha_2^{inp}, \ldots]$を生成する。集約分類器２６０は、$\alpha^{inp}$ベクトルにソフトマックス関数を適用して$\beta^{inp} = softmax(\alpha^{inp})$を決定することにより、スコアのベクトル$\alpha^{inp}$を正規化し、入力エンコーディングにわたる分布を生成する。集約分類器２６０は、入力エンコーディングにわたる分布を生成する。集約分類器２６０は、入力表現κ^{agg}を、以下の式で示されるように、正規化されたスコアβ^{inp}により重み付けされた入力エンコーディングh^{enc}にわたる和として決定する（５１０）。

【数２】

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

【００３９】

集約分類器２６０は、生成された入力表現κ^{agg}に適用される多層パーセプトロンを含み、例えば、ＣＯＵＮＴ、ＭＩＮ、ＭＡＸ、集約なしを示すＮＵＬＬ演算子など、様々な集約演算に対応するスコアα^{agg}を生成する。集約分類器２６０は、生成されたスコアに基づいて、データベースクエリに対する集約演算を識別する（５３０）。

【００４０】

一実施形態において、集約分類器２６０は、以下の式を使用してα^{agg}を決定する。

【数３】

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

【００４１】

項W^{agg}、V^{agg}、b^{agg}及びc^{agg}は、多層パーセプトロンに対応する重みを表す。集約分類器２６０は、ソフトマックス関数を適用して、可能な集約演算のセットにわたる分布$\beta^{agg} = softmax(\alpha^{agg})$を取得する。集約分類器は、交差エントロピー損失$L^{agg}$に基づいて訓練される。

【００４２】

［結果列予測器］

ＳＥＬＥＣＴ句は、選択列又は結果列とも呼ばれる。結果列予測器２７０は、データベーススキーマ内のテーブル列及び自然言語クエリに基づいて選択列を決定する。例えば、自然言語クエリ「いくつのＣＦＬチームが・・・」を所与として、結果列予測器２７０は

、選択列がＣＦＬＤｒａｆｔテーブルからのＣＦＬ＿Ｔｅａｍｓ列を含むことを決定する。したがって、結果列予測器２７０は、ＳＥＬＥＣＴ列予測の問題をマッチング問題として解決する。一実施形態において、結果列予測器２７０は、ポインタを使用してＳＥＬＥＣＴ列を識別する。列表現のリスト及び自然言語クエリの表現を所与として、結果列予測器２７０は、自然言語クエリに最もマッチする列を選択する。

【００４３】

図６は、一実施形態による、自然言語クエリに基づいて出力データベースクエリのＳＥＬＥＣＴ句の列を決定する結果列予測器により実行される処理を示す。結果列予測器２７０は、各列名をＬＳＴＭ（長短期記憶ネットワーク（long short term memory network））でエンコードすること（６１０）により、列に対する入力表現を使用する。入力前処理モジュール２１０は、以下の式を使用して、特定の列ｊの入力表現e_c^jを生成する（６２０）。

【数４】

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

【００４４】

この式において、$h^c_{j,t}$はｊ番目の列のｔ番目のエンコーダ状態を表し、ｅｍｂは埋め込みを返す関数である。入力前処理モジュール２１０は、最後のエンコーダ状態を、e^c_j、列ｊの表現であると見なす。

【００４５】

入力前処理モジュール２１０は、κ^{agg}について上述したものと同様のアーキテクチャを使用して、自然言語クエリκ^{sel}に対する表現を構築する。結果列予測器２７０は、入力表現を条件として列表現にわたり多層パーセプトロンを適用して、以下の式を使用して各列ｊについてスコアを計算する（６３０）。

【数５】

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

【００４６】

この式において、W^{sel}、V^{sel}、及びV^cは、多層パーセプトロンの重みである。結果列予測器２７０は、スコアをソフトマックス関数で正規化して、可能なＳＥＬＥＣＴ列にわたる分布β^{sel}＝ｓｏｆｔｍａｘ（α^{sel}）を生成する（６４０）。上記のＣＦＬＤｒａｆｔテーブルの例では、分布は、列Ｐｉｃｋ＿ｎｕｍｂｅｒ、ＣＦＬ＿Ｔｅａｍ、Ｐｌａｙｅｒ、Ｐｏｓｉｔｉｏｎ、及びＣｏｌｌｅｇｅにわたる。結果列予測器２７０は、正規化されたスコアに基づいて、出力データベースクエリの結果列を選択する（６５０）。集約分類器は、交差エントロピー損失L^{sel}に基づいて訓練される。

【００４７】

［条件句予測器］

一実施形態において、条件句予測器は、ポインタデコーダを使用してＷＨＥＲＥ句を生成する。しかしながら、クエリのＷＨＥＲＥ条件は入れ替えられる可能性があり、クエリは同じ結果を生成する。例えば、自然言語クエリ「どの男性が１８歳より上か（which males are older than 18）」を所与として、出力データベースクエリは、「ＳＥＬＥＣＴ ｎａｍｅ ＦＲＯＭ ｉｎｓｕｒａｎｃｅ ＷＨＥＲＥ ａｇｅ＞１８ ＡＮＤ ｇｅｎｄｅｒ＝“ｍａｌｅ”」又は「ＳＥＬＥＣＴ ｎａｍｅ ＦＲＯＭ ｉｎｓｕｒａｎｃｅ ＷＨＥＲＥ ｇｅｎｄｅｒ＝“ｍａｌｅ” ＡＮＤ ａｇｅ＞１８」のいずれかであり得る。２つのデータベースクエリが、２つのクエリ文字列間の文字列マッチに基づきマッチしない場合でも、双方のデータベースクエリが正しい実行結果を取得する。第１のデータベースクエリが、ニューラルネットワークを訓練する間にグラウンドトルースとして提供され、交差エントロピー損失が、訓練を監督する（supervise）ために使用され

る場合、第２のデータベースクエリは、それが文字列マッチに基づき第１のデータベースクエリにマッチしないため、誤ってペナルティを課される。ゆえに、実施形態は、強化学習を適用してポリシーを学習し、データベースクエリの実行結果の期待された正確さを直接最適化する。

【００４８】

図７は、一実施形態による、出力データベースクエリの条件句を決定する条件句予測器を訓練する処理を示す。条件句予測器２８０は、入力として自然言語クエリ７１０及びデータベーススキーマ７２０を受信して、データベースクエリ７３０を生成する。条件句予測器２８０は、データベース１６０を使用した実行のためにデータベースクエリを送信して、報酬メトリックを取得する。クエリ実行エンジン２３０は、生成されたデータベースクエリ７３０を実行して、予測クエリ結果７５０を取得する。コンピューティングシステム１３０は、グラウンドトルースクエリ結果７５０を訓練データストア２１５に記憶する。条件句予測器２８０は、予測クエリ結果７５０をグラウンドトルースクエリ結果７５０と比較して、報酬７５０を決定する。報酬は、条件句予測器２８０を訓練するためのフィードバックとして、条件句予測器２８０に入力として提供される。

【００４９】

ＷＨＥＲＥ句内の条件句予測器２８０により生成されるトークンのシーケンスは、$y=[y^1, y^2, \ldots, y^T]$で表される。ｑ（ｙ）がモデルにより生成されたクエリを表し、q_gが自然言語クエリに対応するグラウンドトルースデータベースクエリを表すとする。条件句予測器２８０は、報酬メトリックＲ（ｑ（ｙ），q_g）として以下の式を使用する。

【数６】

$$R(q(y), q_g = \begin{cases} -2, & \text{ｑ（ｙ）が有効なＳＱＬクエリでない場合} \\ -1, & \text{ｑ（ｙ）が有効なＳＱＬクエリであり、不正確な結果に対し実行する場合} \\ +1, & \text{ｑ（ｙ）が有効なＳＱＬクエリであり、正確な結果に対し実行する場合} \end{cases} \quad (6)$$

【００５０】

したがって、条件句予測器２８０は、生成されたデータベースクエリの実行の結果がグラウンドトルースとして提供される期待された結果にマッチする場合、正の報酬を割り当てる。条件句予測器２８０は、生成されたデータベースクエリの実行の結果がグラウンドトルースとして提供される期待された結果にマッチするのに失敗した場合、又は生成されたデータベースクエリが有効なデータベースクエリでない場合、負の報酬を割り当てる。

【００５１】

条件句予測器２８０は、損失L^{whe}を、可能なＷＨＥＲＥ句にわたる負の期待された報酬として決定する。訓練モジュールは、勾配降下（gradient descent）を使用して条件句予測器２８０を訓練して、目的関数$L=L^{agg}+L^{sel}+L^{whe}$を最小化する。したがって、条件句予測器２８０は、ＳＥＬＥＣＴ列を予測する際の交差エントロピー損失からの、集約演算を予測する際の交差エントロピー損失からの、及び条件句のためのポリシー学習からの勾配の重み付け和として、総勾配を決定する。

【００５２】

自然言語対データベースクエリ翻訳器１４０における構造の組み込みは、生成される可能性のある無効なデータベースクエリを削減する。大量の無効なクエリは、列名から結果として生じ、生成されたクエリは、テーブルに存在しない選択列を参照する。これは、列名が多くのトークン、例えば４つのトークンを有する「マイル（ｋｍ）」などを含むとき、特に役立つ。集約のために分類器を導入することも、誤り率を削減する。集約分類器の使用は、ＣＯＵＮＴ演算子を予測するための適合率及び再現率を向上させる。条件句を生成するための表現学習の使用は、グラウンドトルースと異なって順序づけられ得る、より高品質のＷＨＥＲＥ句の生成を結果としてもたらす。ポリシーに基づく表現学習での訓練は、条件の順序がグラウンドトルースクエリと異なる場合でも、正しい結果をもたらす。

【００５３】

［コンピュータアーキテクチャ］
図８は、図１のクライアントデバイス及び／又はコンピューティングシステムを実現するための一例示的なコンピュータを示す高レベルブロック図である。コンピュータ８００は、チップセット８０４に結合された少なくとも１つのプロセッサ８０２を含む。チップセット８０４は、メモリコントローラハブ８２０及び入力／出力（Ｉ／Ｏ）コントローラハブ８２２を含む。メモリ８０６及びグラフィックスアダプタ８１２がメモリコントローラハブ８２０に結合され、ディスプレイ８１８がグラフィックスアダプタ８１２に結合される。ストレージデバイス８０８、入力デバイス８１４、及びネットワークアダプタ８１６が、Ｉ／Ｏコントローラハブ８２２に結合される。コンピュータ８００の他の実施形態は、異なるアーキテクチャを有する。

【００５４】
ストレージデバイス８０８は、ハードドライブ、コンパクトディスク読取専用メモリ（ＣＤ‐ＲＯＭ）、ＤＶＤ、又はソリッドステートメモリデバイスなどの、非一時的コンピュータ読取可能記憶媒体である。メモリ８０６は、プロセッサ８０２により使用される命令及びデータを保持する。入力インターフェース８１４は、タッチスクリーンインターフェース、マウス、トラックボール、若しくは他タイプのポインティングデバイス、キーボード、又はこれらの何らかの組み合わせであり、コンピュータ８００にデータを入力するために使用される。いくつかの実施形態において、コンピュータ８００は、ユーザからのジェスチャを介して入力インターフェース８１４から入力（例えば、コマンド）を受け取るように構成されてもよい。グラフィックスアダプタ８１２は、画像及び他の情報をディスプレイ８１８に表示する。ネットワークアダプタ８１６は、コンピュータ８００を１つ以上のコンピュータネットワークに結合する。

【００５５】
コンピュータ８００は、本明細書で説明される機能性を提供するコンピュータプログラムモジュールを実行するように適合される。本明細書で用いられるとき、用語「モジュール」は、指定された機能性を提供するために使用されるコンピュータプログラム論理を参照する。ゆえに、モジュールは、ハードウェア、ファームウェア、及び／又はソフトウェアで実現できる。一実施形態において、プログラムモジュールは、ストレージデバイス８０８に記憶され、メモリ８０６にロードされ、プロセッサ８０２により実行される。

【００５６】
図１のエンティティにより使用されるコンピュータ８００のタイプは、実施形態及びエンティティにより必要とされる処理能力に依存して変わってもよい。コンピュータ８００は、グラフィックスアダプタ８１２及びディスプレイ８１８など、上述のコンポーネントのうちいくつかがなくてもよい。例えば、コンピューティングシステム１３０は、サーバファーム内などのネットワークを通して通信する複数のブレードサーバから形成されてもよい。

【００５７】
本出願の対象事項の実現は、特に、以下の例１～２１であってもよい。
〔例１〕
データベーススキーマを使用して記憶されたデータに基づく入力自然言語クエリを受信するステップと、
上記入力自然言語クエリのターム、
上記データベーススキーマの列のセット、及び
データベースクエリ言語の語彙
を含む複数のタームからトークンのシーケンスを生成するステップと、
１つ以上の入力表現を生成するステップであり、各入力表現は、上記トークンのシーケンスをエンコードすることにより取得される、ステップと、
複数の機械学習に基づくモデルにアクセスするステップであり、各モデルは、上記入力自然言語クエリに対応するデータベースクエリの部分を予測するように構成される、ステップと、

上記複数のモデルの各々について、上記モデルを入力表現に基づいて実行して上記データベースクエリの部分を生成するステップと、
上記データベースクエリの上記生成された部分を組み合わせて上記データベースクエリを取得するステップと、
上記データベースクエリを実行して結果セットを取得するステップと、
を含む方法。
〔例２〕
上記入力自然言語クエリは、クライアントデバイスから受信され、当該方法は、上記結果セットを上記クライアントデバイスに送信するステップをさらに含む、例１に記載の方法。
〔例３〕
上記複数のモデルは、上記データベースクエリ内の集約演算子を決定する集約分類器モデルを含み、上記集約分類器モデルは、多層パーセプトロンを含む、例１に記載の方法。
〔例４〕
上記複数のモデルは、上記データベースクエリの結果列を決定する結果列予測器モデルを含み、上記結果列予測器モデルは、多層パーセプトロンを含む、例１に記載の方法。
〔例５〕
上記複数のモデルは、上記データベースクエリの条件句を決定する条件句予測器モデルを含み、上記条件句予測器モデルは、強化学習に基づく、例１に記載の方法。
〔例６〕
グラウンドトルースデータベースクエリに基づく結果セットを受信するステップと、
上記生成されたクエリから取得された結果と上記グラウンドトルースクエリから取得された結果との比較に基づいて報酬値を決定するステップと、
上記報酬値に基づいて上記条件句予測器モデルの重みを調整するステップと、
をさらに含む例５に記載の方法。
〔例７〕
上記シーケンスの各トークンに対応する列エンコーディングを決定するステップと、
上記入力シーケンスの各トークンについてのスカラー注目スコアを含むベクトルを決定するステップと、
上記ベクトルをソフトマックス関数を使用して正規化するステップと、
上記入力表現を、対応する正規化されたスコアにより重み付けされた上記列エンコーディングの和として決定するステップと、
をさらに含む例５に記載の方法。
〔例８〕
上記複数のモデルを勾配降下を使用して訓練して、上記複数のモデルの各々の結果に基づいて損失を表す目的関数を最小化するステップ
をさらに含む例１に記載の方法。
〔例９〕
上記１つ以上の入力表現を生成するステップは、入力表現κ^{agg}を、
トークンのシーケンス内の各ｔ番目のトークンについて、スカラー注目スコア$\alpha^{in}_{t}=W^{inp}h^{enc}_{t}$を計算し、$h^{enc}_{t}$は、上記入力シーケンス内のｔ番目のワードに対応するエンコーダの状態であり、
上記スコアのベクトル$\alpha^{in}=[\alpha^{inp}_{1},\alpha^{inp}_{2},\cdots]$を正規化して、上記トークンのシーケンス内の上記トークンにわたる分布を生成し、
上記入力表現κ^{agg}を、
【数７】

$$\kappa^{\mathrm{agg}}=\sum_{t}\beta_{t}^{\mathrm{inp}}h_{t}^{\mathrm{enc}}$$

として取得し、$\beta^{agg}=softmax(\alpha^{agg})$及び$\alpha^{agg}=W^{agg}tanh$

（$V^{agg}\kappa^{agg}+b^{agg}$）＋$c^{agg}$である
ことにより計算するステップを含む、例１乃至８のうちいずれか１項に記載の方法。
〔例１０〕
１つ以上の列名を有する上記ＳＱＬクエリのｓｅｌｅｃｔ句を定式化する（formulates）ポインタネットワークで長短期記憶（ＬＳＴＭ）を使用するステップであり、
列表現のリスト及び質問表現を所与として、質問に最もマッチする列を選択することを含み、上記列表現のリストは、各列名をＬＳＴＭでエンコードすることにより取得され、特定の列ｊの表現$e^{c}{}_{j}$は、
【数８】

$$h^{c}_{j,t}=\mathrm{LSTM}\left(\mathrm{emb}\left(x^{c}_{j,t}\right),h^{c}_{j,t-1}\right)\qquad e^{c}_{j}=h^{c}_{j,T_j}$$

により与えられる、ステップ、
をさらに含む例１乃至９のうちいずれか１項に記載の方法。
〔例１１〕
強化学習を使用して、データベースに対して上記生成されたＳＱＬクエリを実行することにより上記ＳＱＬクエリのｗｈｅｒｅ条件を定式化して、
【数９】

$$R(q(y),q_g)=\begin{cases}-2, & \text{q（y）が有効なＳＱＬクエリでない場合}\\ -1, & \text{q（y）が有効なＳＱＬクエリであり、不正確な結果に対し実行する場合}\\ +1, & \text{q（y）が有効なＳＱＬクエリであり、正確な結果に対し実行する場合}\end{cases}$$

として定義される報酬Ｒ（ｑ（ｙ），q_g）を取得するステップであり、ｑ（ｙ）は、上記モデルにより生成されたクエリを表し、q_gは、上記入力自然言語クエリに対応するグラウンドトルースクエリを表す、ステップ
をさらに含む例１乃至１０のうちいずれか１項に記載の方法。
〔例１２〕
自然言語要求を処理してＳＱＬクエリの部分を生成するステップであり、
１つ以上の列名を有する上記ＳＱＬクエリのｓｅｌｅｃｔ句を定式化するポインタネットワークで長短期記憶（ＬＳＴＭ）を使用することと、
強化学習により訓練された拡大ポインタデコーダ（augmented pointer decoder）を使用して上記ＳＱＬクエリのｗｈｅｒｅ条件を定式化するステップと、
を含む方法。
〔例１３〕
多層パーセプトロンを使用して、指定された条件下で選択された列に適用可能な上記ＳＱＬクエリの集約演算を決定するステップ、
をさらに含む例１２に記載の方法。
〔例１４〕
１つ以上の列名を有する上記ＳＱＬクエリのｓｅｌｅｃｔ句を定式化するポインタネットワークで長短期記憶（ＬＳＴＭ）を使用することは、
列表現のリスト及び質問表現を所与として、質問に最もマッチする列を選択すること
を含む、例１２又は１３に記載の方法。
〔例１５〕
上記列表現のリストは、各列名をＬＳＴＭでエンコードすることにより取得され、特定の列ｊの表現$e^{c}{}_{j}$は、
【数１０】

$$h^{c}_{j,t}=\mathrm{LSTM}\left(\mathrm{emb}\left(x^{c}_{j,t}\right),h^{c}_{j,t-1}\right)\qquad e^{c}_{j}=h^{c}_{j,T_j}$$

により与えられる、例１４に記載の方法。

〔例１６〕
上記１つ以上の入力表現を生成することは、入力表現κ^{agg}を、
トークンのシーケンス内の各ｔ番目のトークンについて、スカラー注目スコア$\alpha^{inp}_t = W^{inp} h^{enc}_t$を計算し、$h^{enc}_t$は、上記入力シーケンス内のｔ番目のワードに対応するエンコーダの状態であり、
上記スコアのベクトル$\alpha^{inp} = [\alpha^{inp}_1, \alpha^{inp}_2, \cdots]$を正規化して、上記トークンのシーケンス内の上記トークンにわたる分布を生成し、
上記入力表現κ^{agg}を、
【数１１】

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

として取得し、$\beta^{agg} = softmax(\alpha^{agg})$及び$\alpha^{agg} = W^{agg} \tanh(V^{agg} \kappa^{agg} + b^{agg}) + c^{agg}$である
ことにより計算することを含む、例１４乃至１５のうちいずれか１項に記載の方法。
〔例１７〕
強化学習を使用して、データベースに対して上記生成されたＳＱＬクエリを実行することにより上記ＳＱＬクエリのｗｈｅｒｅ条件を定式化して、
【数１２】

$$R(q(y), q_g) = \begin{cases} -2, & \text{q（y）が有効なＳＱＬクエリでない場合} \\ -1, & \text{q（y）が有効なＳＱＬクエリであり、不正確な結果に対し実行する場合} \\ +1, & \text{q（y）が有効なＳＱＬクエリであり、正確な結果に対し実行する場合} \end{cases}$$

として定義される報酬Ｒ（ｑ（ｙ），q_g）を取得するステップであり、ｑ（ｙ）は、上記モデルにより生成されたクエリを表し、q_gは、上記入力自然言語クエリに対応するグラウンドトルースクエリを表す、ステップ
をさらに含む例１４乃至１６のうちいずれか１項に記載の方法。
〔例１８〕
１つ以上のプロセッサにより実行されたときに上記１つ以上のプロセッサに例１乃至１７のうちいずれか１項に記載のステップを実行させるコンピュータ実行可能コードを含む非一時的コンピュータ読取可能記憶媒体。
〔例１９〕
リカレントニューラルネットワーク（ＲＮＮ）アーキテクチャを生成するコンピュータシステムであって、
１つ以上のコンピュータプロセッサと、
１つ以上のコンピュータにより実行されたときに上記１つ以上のコンピュータに請求項１８に記載の方法のステップを実行させるコンピュータ実行可能コードを含む非一時的コンピュータ読取可能記憶媒体と、
を含むコンピュータシステム。
〔例２０〕
１つ以上のコンピュータプロセッサと少なくとも１つの非一時的記憶媒体とを含むコンピュータシステムであって、
データベーススキーマを使用して記憶されたデータに基づく入力自然言語クエリを受信する手段と、
上記入力自然言語クエリのターム、
上記データベーススキーマの列のセット、及び、
データベースクエリ言語の語彙
を含む複数のタームからトークンのシーケンスを生成する手段と、
１つ以上の入力表現を生成する手段であり、各入力表現は、上記トークンのシーケンスをエンコードすることにより取得される、手段と、

複数の機械学習に基づくモデルにアクセスする手段であり、各モデルは、上記入力自然言語クエリに対応するデータベースクエリの部分を予測するように構成される、手段と、
上記複数のモデルの各々について、上記モデルを入力表現に基づいて実行して上記データベースクエリの部分を生成する手段と、
上記データベースクエリの上記生成された部分を組み合わせて上記データベースクエリを取得する手段と、
上記データベースクエリを実行して結果セットを取得する手段と、
をさらに含むコンピュータシステム。
〔例２１〕
１つ以上のコンピュータプロセッサと少なくとも１つの非一時的記憶媒体とを含むコンピュータシステムであって、
自然言語要求を処理してＳＱＬクエリの部分を生成する手段であり、
１つ以上の列名を有する上記ＳＱＬクエリのｓｅｌｅｃｔ句を定式化するポインタネットワークで長短期記憶（ＬＳＴＭ）を使用する手段と、
強化学習により訓練された拡大ポインタデコーダを使用して上記ＳＱＬクエリのｗｈｅｒｅ条件を定式化する手段と、
を含む、手段
をさらに含むコンピュータシステム。

【００５８】
［代替的な実施形態］
開示された実施形態は、関係データベースに基づいており、ＳＱＬを使用して例示されているが、開示された技術は、他のタイプのデータベース、例えば、オブジェクトベースのデータベース、オブジェクト関係データベースなどに適用可能である。開示された技術は、特定タイプのデータベースに使用されるデータベースクエリ言語が結果列、集約句、又は条件句に相当する機能をサポートする場合、適用可能である。例えば、データベースクエリ言語が条件句をサポートする場合、条件句予測器を使用して、入力自然言語クエリに基づいて出力データベースクエリの条件句を予測することができる。
【００５９】
本発明の図及び説明は、明確さの目的で、典型的な分散システムに見られる他の多くの要素を排除すると同時に、本発明の明確な理解に関連する要素を例示するよう簡略化されていることを理解されたい。当業者は、他の要素及び／又はステップが実施形態を実現する際に望ましく、かつ／あるいは必要とされることを認識し得る。しかしながら、こうした要素及びステップは当該分野において良く知られているため、かつそれらは実施形態のより良い理解を促進しないため、こうした要素及びステップの議論は本明細書で提供されない。本明細書の開示は、当業者に知られるこうした要素及び方法に対するすべてのこうした変形及び修正に向けられる。
【００６０】
上記説明のいくつかの部分は、実施形態を、情報に対する演算のアルゴリズム及びシンボル表現の観点から説明する。これらのアルゴリズム的説明及び表現は、データ処理分野の当業者により一般的に使用され、その作用の実体を他の当業者に効果的に伝達する。これらの動作は、機能的、計算的、又は論理的に説明されているが、コンピュータプログラム又は同等の電気回路、マイクロコードなどにより実現されるものと理解される。さらに、これらの動作の配置を一般性を損なうことなくモジュールとして参照することは、時に便利であることも証明されている。説明した動作及びその関連モジュールは、ソフトウェア、ファームウェア、ハードウェア、又はこれらの任意の組み合わせで具体化されてもよい。
【００６１】
本明細書で用いられるとき、「１つの実施形態」又は「一実施形態」へのいずれの参照も、実施形態に関連して説明された特定の要素、特徴、構造、又は特性が少なくとも１つ

の実施形態に含まれることを意味する。明細書中の様々な箇所におけるフレーズ「１つの実施形態において」の出現は、必ずしもすべてが同じ実施形態を参照しているわけではない。

【００６２】

いくつかの実施形態は、表現「結合された」及び「接続された」をその派生形と共に使用して説明され得る。これらの用語は、互いに同義語として意図されてはいないことを理解されたい。例えば、いくつかの実施形態は、用語「接続された」を使用して説明されて、２つ以上の要素が互いに直接物理的又は電気的に接触していることを示し得る。別の例において、いくつかの実施形態は、用語「結合された」を使用して説明されて、２つ以上の要素が直接物理的又は電気的に接触していることを示し得る。しかしながら、用語「結合された」は、２つ以上の要素が互いに直接接触していないが、なお依然として互いに協調又は対話することをさらに意味し得る。実施形態は、この文脈において限定されない。

【００６３】

本明細書で用いられるとき、用語「含む」、「含んでいる」、「含める」、「含めている」、「有する」、「有している」、又はこれらの任意の他の変形は、非排他的な包含をカバーすることが意図される。例えば、要素のリストを含む処理、方法、物品、又は装置は、必ずしもこれらの要素のみに限定されず、明確に列挙されていないか又はこうした処理、方法、物品、若しくは装置に固有の他の要素を含んでもよい。さらに、逆のことを明確に示されない限り、「又は」は、排他的ＯＲでなく包括的ＯＲを参照する。例えば、条件Ａ又はＢは、Ａが真であり（又は存在し）Ｂが偽である（又は存在しない）と、Ａが偽であり（又は存在せず）Ｂが真である（又は存在する）と、Ａ及びＢの双方が真である（又は存在する）と、のうちいずれか１つにより満たされる。

【００６４】

さらに、一の（“ａ”又は“ａｎ”）の使用は、本明細書における実施形態の要素及びコンポーネントを説明するために採用されている。これは単に、簡便さのため、及び本発明の一般的な意味を与えるために行われる。本説明は、１つ又は少なくとも１つを含むように読まれるべきであり、その他を意図されることが明らかでない限り、単数形は複数形も含む。

【００６５】

本開示を読むと、当業者は、本明細書で開示された原理を通して、歪み領域を使用してチャートを表示するシステム及び処理のためのなおさらなる代替的な構造及び機能設計を理解するであろう。ゆえに、特定の実施形態及び適用が例示され説明されたが、開示された実施形態は、本明細書に開示された正確な構造及びコンポーネントに限定されないことが理解されるべきである。当業者に明らかであろう様々な修正、変更及び変形が、別記の特許請求の範囲に定義された主旨及び範囲から逸脱することなく、本明細書に開示された方法及び装置の配置、動作、及び詳細において行われ得る。

【００６６】

［付録］

以下の「Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning」と題された文献は、本付録の一部であり、ゆえに本出願の開示の一部である「Ｓｅｑ２ＳＱＬ」を開示しており、「Ｓｅｑ２ＳＱＬ」、自然言語の質問を対応するＳＱＬクエリに翻訳するためのディープニューラルネットワークを開示している。Ｓｅｑ２ＳＱＬは、ＳＱＬクエリの構造を活用して、生成されたクエリの出力空間を有意に削減する。Ｓｅｑ２ＳＱＬは、ＳＱＬクエリの構造を活用する３つのコンポーネントを含み、生成されたクエリの出力空間を大幅に削減する。詳細には、Ｓｅｑ２ＳＱＬは、ポインタデコーダ及びポリシー勾配を使用してクエリの条件を生成し、これは、順序づけられていない性質に起因して交差エントロピー損失を使用した最適化に適さないことを我々は示す。Ｓｅｑ２ＳＱＬは、交差エントロピー損失とデータベースに対するライブのクエリ実行からの強化学習報酬とを組み合わせた混合された目的を使用して訓練される。これらの特性は、モデルがクエリ生成において一層向上した結果を達成することを

可能にする。

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.

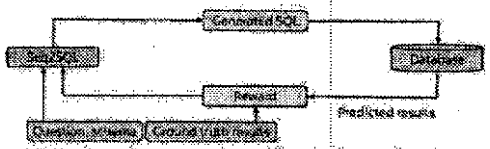


Figure 1: Seq2SQL takes as input a question and list of columns of a table. It generates the corresponding SQL query, which, during training, is executed against a database. The result of the execution is utilized as the reward to train the reinforcement learning algorithm.

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, Wang 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 [Ling 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

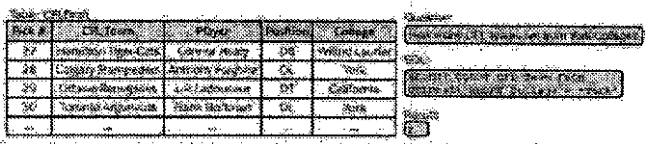


Figure 2: An example in WikiSQL. The inputs consist of a table and question. The outputs consist of a ground truth SQL query and the corresponding result from execution.

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^v 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:

$$x = [\langle col \rangle; x_1^c; x_2^c; ...; x_N^c; \langle sql \rangle; x^q; \langle question \rangle; x^v] \quad (1)$$

[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 tokens 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_i, where h_i is the state of the encoder corresponding to the ith 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_i.

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 $a_{s,i}^{ptr}$ for each position i of the input sequence x:

$$a_{s,i}^{ptr} = W^{ptr} \tanh\left(U^{ptr} g_s + V^{ptr} h_i\right) \quad (2)$$

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

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 case 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".

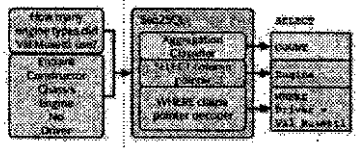


Figure 3: The Seq2SQL model has three components, corresponding to the three parts of a SQL query (right). The input to the model are the question (top left) and the table column names (bottom left).

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 c^{agg}. Similar to Equation 2, we first compute the scalar attention score, $a_i^{agg} = W^{agg} h_i^{enc}$, for each ith token in the input sequence. The vector of all scores, $a^{agg} = [a_1^{agg}, a_2^{agg}, \ldots]$, is then normalized to produce a distribution over all the tokens in the input sequence, $\beta^{agg} = \text{softmax}\{a^{agg}\}$. The input representation, c^{agg}, is the sum over the column encodings weighted by the corresponding

normalized attention weight:

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

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

$$o^{agg} = W_2^{agg} \tanh\left(W_1^{agg} x^{agg} + b^{agg}\right) + c^{agg} \tag{4}$$

The distribution over the set of possible aggregation operations, $\beta^{agg} = \text{softmax}(o^{agg})$, 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^e, is given by:

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

Here, $h_{j,i}^e$ denotes the ith encoder state of the jth column, c_j^e, 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 x^{sel} using the same architecture as for x^{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 i score for each column j:

$$o_j^{sel} = W_2^{sel} \tanh\left(V^{sel} c_j^{sel} + V^e c_j^e\right) \tag{6}$$

We normalize the scores with a softmax function to produce a distribution over the possible SELECT columns: $\beta^{sel} = \text{softmax}(o^{sel})$. 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^{sel}.

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 $q(y)$ denote the query generated by the model and q_g denote the ground truth query corresponding to the question. The reward $R(q(y), q_g)$ is defined as:

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

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 $R(q(y), q_s)$. The loss, $L^{mle} = -E_{y}[R(q(y), q_s)]$, 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 $P(y^t)$ denote the probability of choosing token y^t during time step t, the corresponding synthetic gradient is $R(q(y), q_s) \log P(y^t)$.

Mixed Objective Function

The model is trained using gradient descent to minimize the objective function $L^{mix} = L^{seq} + L^{wl} + L^{wc}$. 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.

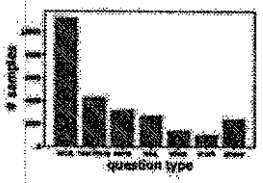


Figure 4: Distribution of question types in WikiSQL.

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 3. 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/25625?2+25

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

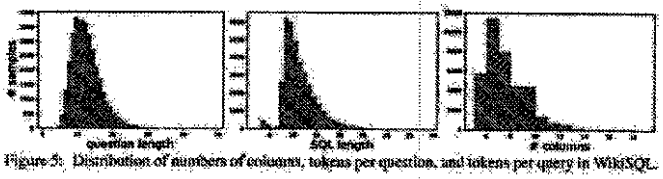


Figure 5: Distribution of numbers of columns, tokens per question, and tokens per query in WikiSQL.

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 $Acc_{ex} = \frac{N_{ex}}{N}$.

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 $Acc_{lf} = \frac{N_{lf}}{N}$.

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 and name and SSN pair with different names share the SSN 123. Due to these limitations, 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	26008	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. Early ATIS versions contain no logical forms.

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 n-gram embeddings [Hashimoto et al., 2016]. Let w^g denote the GloVe embedding and w^c the character embedding for word w. Here, w^c_n is the mean of the embeddings of all the character n-grams in w. For a given word w, we define its word embedding $w_e = [w^g; w^c_n]$ 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 300 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 [4].

[4] http://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, 1830?", 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 1830`. 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. "1830" 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 29.8% 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 treston, new jersey race where a.j. foyt had the pole position?", the ablation of Seq2SQL trained without reinforcement learning generates `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`.

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.

Model	Precision	Recall	F1
Avg. 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.

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. Šuster 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 phase 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.

$$y_t = \text{LSTM}\left(\left[\text{emb}\left(y_{t-1}\right), a_{t-1}^{\text{dec}}\right], y_{t-1}\right) \tag{8}$$

where the d_{hid} dimensional a_t^{dec} is the attentional context over the input sequence during the tth decoding step, computed as

$$e_{t,i}^{\text{dec}} = h_t^{\text{dec}} \left(W^{\text{dec}} h_i^{\text{enc}}\right)^T \qquad \delta_t^{\text{dec}} = \text{softmax}\left(e_t^{\text{dec}}\right) \tag{9}$$

$$a_t = \sum_i \delta_{t,i} h_i^{\text{enc}} \tag{10}$$

To produce the output token during the tth 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

$$o^{\text{dec}} = \text{softmax}\left(U^{\text{dec}}\left[h_t^{\text{dec}}, a_t^{\text{dec}}\right]\right) \tag{11}$$

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

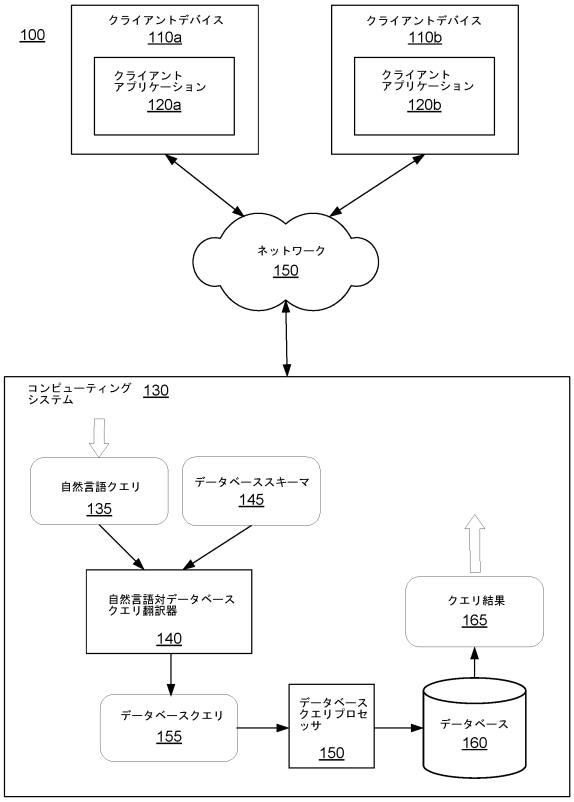
During training, teacher forcing is used. During inference, a beam size of 5 is used and generated unknown words are replaced by the input words with the highest attention weight.

C Predictions by Seq2SQL

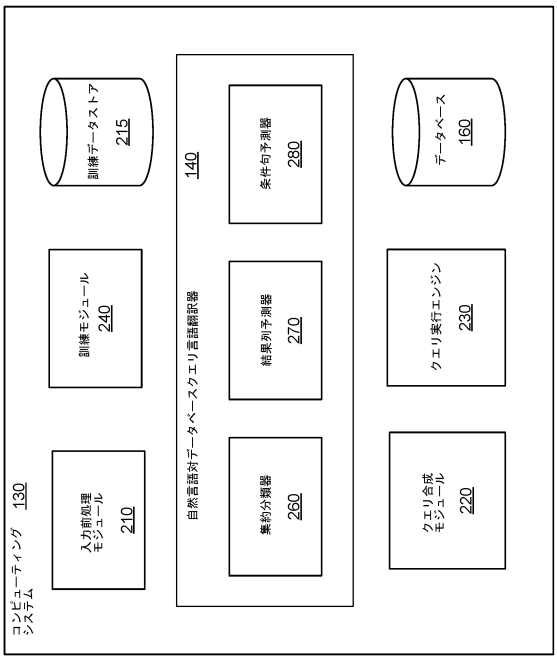
Q	when connecticut & villanova are the regular season winner how much tournament venues (city) are there?
P	SELECT COUNT tournament_venue (city) WHERE regular_season_winner = connecticut & villanova
S	SELECT COUNT tournament_venue (city) WHERE tournament = connecticut & villanova
S'	SELECT COUNT tournament_venue (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 2nd lap results are 0.1?
P	SELECT aggregate_score lap_2 = 0.1
S	SELECT COUNT agg. score WHERE lap_2 = 0.1
S'	SELECT agg. score WHERE lap_2 = 0.1
G	SELECT agg. score WHERE lap_2 = 0.1
Q	what is the race name of the 125 united states grand prix where the pole position?
P	SELECT race_name WHERE location = 125th AND result_position = a_1 type, new_jersey AND
S	SELECT race_name WHERE rnd = 12 AND track = a_1 type AND pole_position = a_1 type
S'	SELECT race_name WHERE rnd = 12 AND pole_position = a_1 type
G	SELECT race_name WHERE rnd = 12 AND pole_position = a_1 type
Q	what city is at 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 votes from the bronx voted for the socialist party?
P	SELECT MIN 2 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 release february 9?
P	SELECT MIN year (ad) WHERE date = february 9 AND race_time = plymouth 8
S	SELECT year (ad) WHERE date = plymouth 9 AND race_time = february 9
S'	SELECT year (ad) WHERE date = plymouth 9 AND race_time= february 9
G	SELECT year (ad) 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.

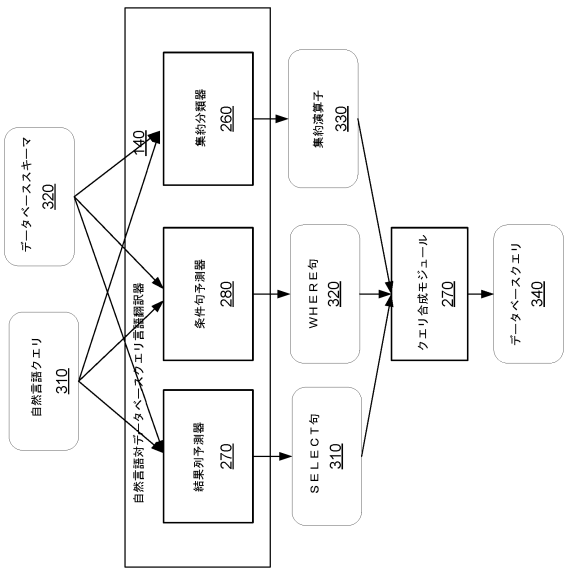
【図１】


100
クライアントデバイス
110a
クライアント
アプリケーション
120a
クライアントデバイス
110b
クライアント
アプリケーション
120b
ネットワーク
150
コンピューティング
システム
130
自然言語クエリ
135
データベーススキーマ
145
自然言語対データベース
クエリ翻訳器
140
データベースクエリ
155
データベース
クエリプロセッサ
150
データベース
160
クエリ結果
165

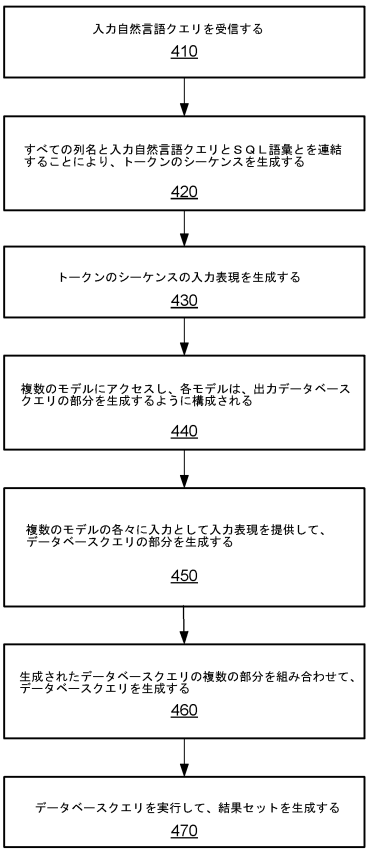
【図２】


コンピューティング
システム
130
入力前処理
モジュール
210
訓練モジュール
240
訓練データストア
215
自然言語対データベースクエリ言語翻訳器
140
集約分類器
260
結果列予測器
270
条件句予測器
280
クエリ合成
モジュール
220
クエリ実行エンジン
230
データベース
160

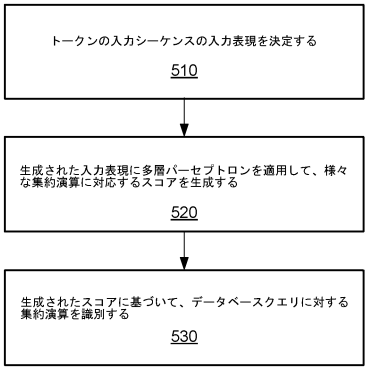
【図３】


データベーススキーマ
320
自然言語クエリ
310
自然言語対データベースクエリ言語翻訳器
140
集約分類器
260
条件句予測器
280
結果列予測器
270
集約演算子
330
WHERE句
320
SELECT句
310
クエリ合成モジュール
270
データベースクエリ
340

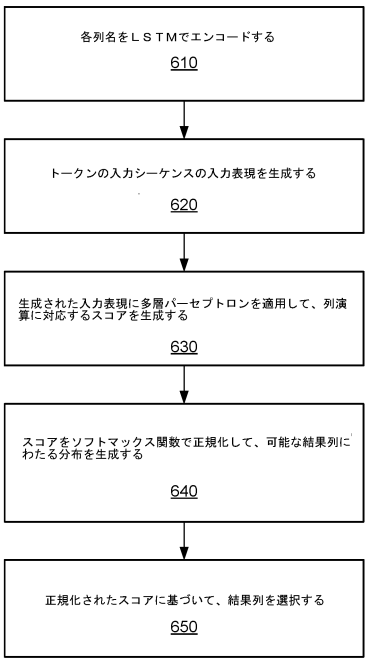
【図４】


入力自然言語クエリを受信する
410
すべての列名と入力自然言語クエリとＳＱＬ語彙とを連結することにより、トークンのシーケンスを生成する
420
トークンのシーケンスの入力表現を生成する
430
複数のモデルにアクセスし、各モデルは、出力データベースクエリの部分を生成するように構成される
440
複数のモデルの各々に入力として入力表現を提供して、データベースクエリの部分を生成する
450
生成されたデータベースクエリの複数の部分を組み合わせて、データベースクエリを生成する
460
データベースクエリを実行して、結果セットを生成する
470

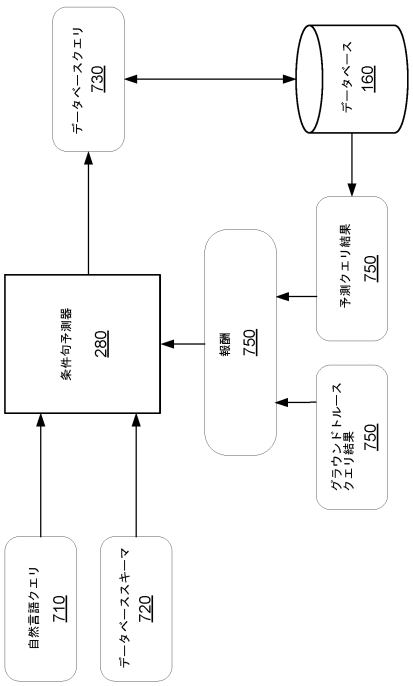
【図５】


トークンの入力シーケンスの入力表現を決定する
510
生成された入力表現に多層パーセプトロンを適用して、様々な集約演算に対応するスコアを生成する
520
生成されたスコアに基づいて、データベースクエリに対する集約演算を識別する
530

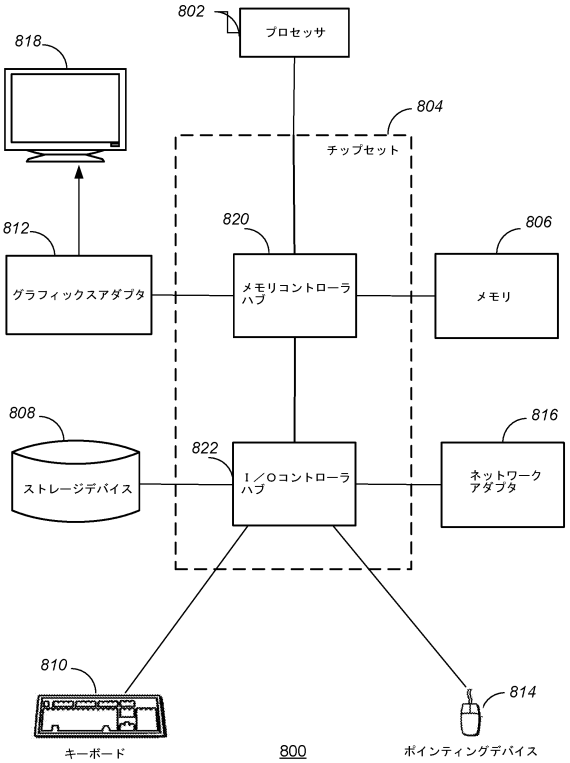
【図６】


各列名をＬＳＴＭでエンコードする
610
トークンの入力シーケンスの入力表現を生成する
620
生成された入力表現に多層パーセプトロンを適用して、列演算に対応するスコアを生成する
630
スコアをソフトマックス関数で正規化して、可能な結果列にわたる分布を生成する
640
正規化されたスコアに基づいて、結果列を選択する
650

【図７】


データベースクエリ 730
データベース 160
予測クエリ結果 750
報酬 750
グラウンドトルースクエリ結果 750
条件句予測器 280
自然言語クエリ 710
データベーススキーマ 720

【図８】


802
プロセッサ
818
804
チップセット
812
820
806
グラフィックスアダプタ
メモリコントローラハブ
メモリ
808
816
822
ストレージデバイス
Ｉ／Ｏコントローラハブ
ネットワークアダプタ
810
814
キーボード
800
ポインティングデバイス

フロントページの続き

(72)発明者　ゾン，ヴィクター
アメリカ合衆国　カリフォルニア州　９４１０５，サンフランシスコ，ザ　ランドマーク　アット　ワン　マーケット　ストリート，スイート　３００，セールスフォース　ドット　コム　インコーポレイティッド
(72)発明者　ション，カイミング
アメリカ合衆国　カリフォルニア州　９４１０５，サンフランシスコ，ザ　ランドマーク　アット　ワン　マーケット　ストリート，スイート　３００，セールスフォース　ドット　コム　インコーポレイティッド
(72)発明者　ソーチャー，リチャード
アメリカ合衆国　カリフォルニア州　９４１０５，サンフランシスコ，ザ　ランドマーク　アット　ワン　マーケット　ストリート，スイート　３００，セールスフォース　ドット　コム　インコーポレイティッド

審査官　多胡　滋

(56)参考文献　国際公開第２０１６／１５１６９０（WO，Ａ１）
米国特許出願公開第２０１３／０２３９００６（ＵＳ，Ａ１）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　１６／２４
Ｇ０６Ｎ　３／００