Applying ML techniques in CL - FINAL PROJECT

Gopal Seshadri, Raja Rajeshwari Premkumar, Yashaswini Dhatrika

Abstract
We aim to make information easily available to non technical users. One of the major sources of information is Databases. Users of any background can analyze large amounts of data if they are able to use natural language processing to query databases.Hence we need a system that is able to yield SQL queries automatically by simply interpreting the user questions given to it. We use various end to end deep learning models and probability model such as conditional random field to achieve this.

Contents

1. Introduction

In this modern world which prefers automation and communication with systems, having a natural language based conversation system is a boon in many ways. This is a time where laymen are able to access a myriad of data sources. If there exists a system that can query databases by taking natural language queries as input- it can break potential barriers between them. Hence, the rising need for Natural Language Interface to Database(NLIDB). An NLIDB system translates a natural language sentence into a database query. NLIDB is a commonly researched problem in natural language. There are two stages in solving NLIDB system - The first stage is Linguistic processing and second stage is Database processing.

1. Linguistic processing- It uses mapping functions to translate natural language query to structured query language.
2. Database processing- The data is extracted from the database using the SQL query generated in the first stage.

The focus of our work is on the Linguistic processing phase where the scope is limited to providing SQL queries for relational databases with input natural language which is English.

Neural networks in general consider the inputs to be independent of each other. Output is generated once the input is processed by number of hidden layers.On the other hand recurrent neural networks consider the previous computations as well. Yet RNN is unable to map long term dependencies due to vanishing gradient issue. Hence, the LSTMs are used widely in Natural Language Processing which takes the entire sequence into consideration. Main problem here is that only the previous computations are taken into consideration, we need an architecture that also considers the information later in the sequence. Recent works uses recurrent neural network with attention or an encoder decoder

architecture (Dong Lapata, 2016; [4] Wang et al., 2016 [20]) for creating high accuracy models for sequence to sequence task like Machine Translation. Such approaches involve using various recurrent units that can remember the state like LSTM and GRU. The approach to create SQL queries from text resembles the strategy used for Neural Machine Translation which translates one natural language to another [1]. In this paper, we used a similar encoder decoder model and attention based model to create an end to end sequence to sequence architecture for converting natural language to SQL queries.

First, we start of with the data processing step, which involves creating synsets for all words except the nouns, the stop words and the words that are part of a compound tag in dependency tree. We identify three synsets for all other words in the sentence and replace the actual word with synonyms to paraphrase a sentence into new one. This greatly increases the size of our data set. We also identified the query template for each SQL queries and we found that all the queries in the data set follows nine different templates. We used this template creation strategy extensively for our Conditional Random Field Model.We implemented sequence to sequence models and probabilistic model like conditional random field on the extended data set.

We used the Automatic Terminal Information Service(ATIS) dataset to train and evaluate our models. The training data set has four thousand records, but after paraphrasing we got a new data set of around 41 thousand records, which we indeed used to train our models.

2. Previous Work

Previous works in this area mainly use template-based approach and end-to-end deep learning approach. The template-based approach involves creating various templates from the given training data, identifying the corresponding template for each input query and then filling the clauses based on maximum likelihood strategy. There are few existing systems that can directly generate SQL queries from natural language (Giordani [11], 2012; Popescu et al., 2003 [16]; Cho, 2014 [8]). But this approach makes strong assumptions on the structure of queries, involves lexical matching between tokens in the query against the table and column names, and they can only generate a subset of SQL queries. Such techniques were widely followed before 2010.

There are systems that commonly use pattern matching, intermediate representations of the query, grammar-based techniques (Pazos Rangel et al., 2013). There are recent works that incorporate user feedback and active learning to improve accuracy (Li and Jagadish, 2014).

Another area that is much similar to our task is code generation. This involves parsing natural language queries into program code in programming language such as Python (Bahdanau et al., 2014 [5]; Su et al., 2016 [18]; Auli et al., 2013; [3] Gao et al, 2013 [10]). Our task is similar to that of Ling et al, but the only difference is we are trying to convert a Natural Language Query into a SQL query. Our method of generating a SQL query from natural language can be modeled as a sequence to sequence model. Much of the previous work in this research focused on a single database with a fixed schema for training and testing and hence it cannot be generalized to a new database. We are going to model a similar approach for Automatic Terminal Information Service (AITS) database.

For our analysis, we consider the AITS dataset, standardizing it to have a consistent SQL style. ATIS User questions for a flight-booking task, manually annotated. We use the modified SQL from Iyer et al. (2017) [13], which follows the data split from the logical form version. The dataset has only four thousand records and hence a data augmentation strategy must be used. A paraphrasing strategy is used for augmentation, similar to that of Artzi and Zettlemoyer, 2013 [2].

The task of converting natural language queries into SQL query can be viewed as a sub-task of semantic parsing, and many such the text to logical query problem has been studied for many years (Warren and Pereira, 1982; Popescu et al., 2003, 2004; Li et al., 2006; Giordani and Moschitti, 2012; Wang et al., 2017b). The methods that are created by Database community (Li and Jagadish, 2014;

Yaghmazadeh et al., 2017) involve user interactions with the systems and more hand crafting. In this work, we focus on recent neural network based approaches (Goodfellow et al., 2014 [12]; Cho et al., 2014 [9]; Wang et al., 2014 [19]; Yin et al., 2016; Iyer et al., 2017 [13]). Sarhan and Amany (2009) [17] introduce an end to end approach to converting text to logical forms.

The end to end deep learning approach (Iyer et al. 2017) [13] to this problem involves using various recurrent units that can remember the state like LSTM and GRU. The approach to create SQL queries from text resembles the strategy used for Neural Machine Translation which translates one natural language to another. In contrast, we translate a natural language input into SQL query in order to implement text to SQL for NLIDB. Hence, we used a neural sequence-to-sequence model to directly generate SQL queries from natural language queries. Our work is built on models that show excellent result on Neural Machine Translation (Bahdanau et al., 2014) [6] and end to end natural language generation (Cho et al., 2014). [9]

3. Design of Experiments and Algorithms

Our training dataset is split into two files:

1. train.nl- It has the input natural language queries.
2. train.sql- It has the SQL query corresponding to the Natural language query in train.nl.

The dataset has 4379 rows.

3.1 Data Processing

The data processing of our input is done two fold.

1. Paraphrasing - There are many ways to represent a sentence or a query. Our aim is to recreate the meaning of the queries using different words but at the same time keeping the essential attributes or database object names intact.
2. Template creation - The SQL data has only SELECT query but these SELECT query has its own pattern. We identified 9 unique patterns in the data set, these pattern retains only the SQL keywords. Some have a basic structure with only *SELECT, DISTINCT, FROM* and *WHERE*. Where as other queries may have a *GROUP BY* clause or few other queries may have sub-queries. Some of the commonly used keywords in the queries are:

```
SELECT
DISTINCT
FROM
WHERE
GROUP
BY
HAVING
```

3. Clause Attribute Classification- In this step, we classify the Natural language query into different clauses in a SQL query.Explained as below:
Select clause: variables used after the select keyword in the SQL query.
Group by clause: variables used after the ”group by” keyword in the SQL query.
Having clause: variables used after having keyword in the SQL query.
Where attribute clause:variables used after where keyword in the SQL query.
Where value clause:variables used after where keyword and ”=” in the SQL. query.That captures the values for filtering the database.

3.1.1 Paraphrasing

An input natural language query can be paraphrased to different sentences keeping the meaning intact. This way our model will have more data and train better. After paraphrasing it has 41222 data points.

In order to paraphrase, we use the wordnet synsets to replace the words.

We use the dependency parser to find if the word is a compound word (compound),open clausal complement (xcomp), noun object(nobj), object of preposition (pobj) or a direct object(dobj) to avoid paraphrasing it.

- Compound word (compound) - It can be hyphenated word. It could be a phrase modifying the head of a noun phrase, a quantifier phrase or a prepositional phrase.
- Noun object (nobj) - It is simply a noun object
- Object of preposition (pobj)- It is simply a noun phrase that modifies the head of prepositional phrase.
- Direct object (dobj)- It is noun phrase that is accusative object of direct transitive verb.

For the sentence *"list all the flights that arrive at general mitchell international from various cities"*, below we can see the word and its corresponding Dependency Parse Tree tag and POS tag:

```
list          - 'ROOT' , 'VERB'
all           - 'predet, 'ADJ'
the           - 'det'  , 'DET'
flights       - 'dobj' , 'NOUN'
that          - 'nsubj', 'ADJ'
arrive        - 'relcl', 'VERB'
at            - 'prep' , 'ADP'
general       - 'amod' , 'ADJ'
mitchell      - 'amod' , 'NOUN'
international - 'pobj' , 'ADJ'
from          - 'prep' , 'ADP'
various       - 'amod' , 'ADJ'
cities        - 'pobj' , 'NOUN'
```

Here the words list and arrive are the only words that are paraphrased. We find the synsets from wordnet. Synsets are nothing but set of words that are similar in meaning. They can be utilized to obtain the synonyms for the words. For example, below are the synonyms for the word *list* are

```
Synset('list.n.01')
Synset('name.v.02')
Synset('number.v.03')
```

The synonyms for the word *come* are

```
Synset('come.v.01')
Synset('arrive.v.01')
Synset('reach.v.02')
```

Below is the output after paraphrasing the sentence:

Given-"list all the flights that arrive at general mitchell international from various cities" New sentences created after paraphrasing:

1. name all the flights that arrive at general mitchell international from various cities
2. list all the flights that get at general mitchell international from various cities
3. list all the flights that come at general mitchell international from various cities
4. list all the flights that come_up at general mitchell international from various cities
5. name all the flights that get at general mitchell international from various cities
6. name all the flights that come at general mitchell international from various cities
7. name all the flights that come_up at general mitchell international from various cities.

Below is the process followed to achieve this: The first word of any sentence is always paraphrased. If it is not the first word we do the following:

- If it is a noun we do not paraphrase it. The nouns could be potential candidates for becoming a database object- namely table name, column name etc. Hence we avoid losing it in order to accomodate it in our output sequence.
- There is no gain in paraphrasing stop words, hence we exclude that as well
- If the dependency parse tree yields the tag nobj,pobj or dobj it is not paraphrased
- If the dependency parse tree yields the tag compound or xcomp then we exclude it in order to ensure that we do not paraphrase part of a phrase or word.
- If the word clears all the above conditions, then we use the word along with its pos tag to obtain its synsets from wordnet.
- We use only the first three synsets.
- We make sure that each paraphrased word occurs in combination with each of the other for a given sentence.

3.1.2 Template Creation

Training SQL data set is used for creating the templates. The templates are created by retaining all the SQL keywords like *SELECT, DISTINCT, IN, FROM, GROUP, BY, HAVING, WHERE and JOIN* and discarding all other variables. We identified 9 different templates. The following are some of the templates

```
['SELECT', 'DISTINCT', 'FROM',
'WHERE']

['SELECT', 'DISTINCT', 'COUNT',
'FROM', 'WHERE', 'GROUP', 'BY']

['SELECT', 'DISTINCT', 'FROM',
'WHERE',(
'SELECT', 'MIN', 'FROM', 'WHERE'),
('SELECT', 'MAX', 'FROM', 'WHERE'
)]
```

The first template above has a basic structure with only SELECT, DISTINCT, FROM and WHERE clause, whereas the second template has a GROUP BY clause and the third template has sub queries.

3.1.3 Clause Attribute Classification

Attribute classification is extracted for the data by extracting the position of words after the different clauses (SELECT, GROUP BY, WHERE etc.) And these words are matched with that of words in the Natural language query and then tagged the each word respectively. This is repeated for all the records in the paraphrased training data set. The output of Natural language query after the attribute classification for the input sentence *"list all the flights that arrive at general mitchell international from various cities"* is shown in Table 1.

3.2 Methodology

3.2.1 Sequence to Sequence Models

The Natural Language Interface for Database can be modelled as a sequence-to-sequence model. A sequence-to-sequence model will always try to map an input sequence of fixed length with an output sequence of fixed length. The length of the input sequence and output sequence can be different. But all the input sequences should be of the same length for a deep learning approach using LSTM to train. Hence for modelling such sequence to sequence architecture we must bring all the input sequence to a fixed length. This fixed length is either the maximum length of the input sequence or any arbitrary value between the median length of the input sequence and the max length. Once we did padding for both the input and the output sequence, we can build a sequence-to-sequence architecture on the data. Here in this project, we considered the two of the most commonly used sequence to sequence architectures namely an Encoder-Decoder Architecture and an Attention-Based Architecture.

Encoder-Decoder Architecture: The Encoder Decoder Architecture consists of two parts, namely an Encoder and a decoder. Encoder mainly consists of a sequence of LSTM or GRUs, in our case we used LSTM. Each LSTM unit accepts a word or a single element from the encoder input sequence and it collects information for that particular word/element and propagates the information forward.

In our text to SQL translation problem, the encoder input sequence is nothing but the collection of words in our Natural Language Query and we can represent each word as x_t where 't' is the temporal step of the word. Here in an encoder rather than returning an output for each LSTM unit we instead only the final hidden state. This final hidden state produced from the encoder model is referred to as a context vector or an Encoder Vector. This vector encapsulates all the information from the input sequence and this is used as a context on which the decoder makes its predictions.

The decoder model again consists of several LSTM units which take as input a word or a single element from the decoder input sequence. Here, the decoder input sequence is the collection of words in our SQL query. We add a SOS tag at the beginning of each decoder input sequence since our decoder model needs to predict the first word of the output sequence only from the context vector and no previous output. For this purpose, we append SOS tag as the first element of the decoder input sequence. The output from our first LSTM unit in

the decoder is passed as input to the next unit and so on. This way the decoder can output a vector that maximizes the probability given the encoder vector. Each element in the output is labelled as y_t where 't' is the timestep. A softmax is used on the decoder output to find the actual word and the final output sequence is a collection of words for the SQL query.

We also tried building a character level model where we feed in individual characters as input rather than using word tokens, but provided the long length of Natural Language and SQL queries which must be fed as input to encoder and decoder respectively and it is difficult for LSTM units to remember such long term dependencies and longer temporal sequence of character level model requires more computation power, hence it is costlier to train such character level model. In our paper, we going to discuss only about the word level model.

Attention-Based Architecture: The main drawback of Encoder – Decoder Architecture is that the decoder receives only the encoder vector i.e., only the last encoder hidden state. This encoder vector is expected to summarize the entire input sequence and for long input sentences, we expect the decoder to create the output just based on one vector sequence. If the given input sentence is too large this will it harder for the decoder to predict the output sentence given the input sentence. For the purpose of overcoming this difficulty, we are using Attention-Based architecture.

An Attention-Based Model consists of three main parts namely, an Encoder, a Decoder and an Attention Layer. The Attention layer is considered as an interface between Encoder and the Decoder that provides each unit in the decoder with the information from the encoder hidden state. There are two different types of Attention namely, Global Attention and Local Attention. In Global Attention mechanism, each and every decoder units listen to all the encoder hidden states whereas, in a Local Attention mechanism the decoder units will listen to only a subset of encoder states. For the task, we used the Global Attention mechanism since SQL queries follow a fixed structure and partial listening of input won't work in our case as it works for Neural Machine Learning. The Encoder and Decoder part of the Attention model is roughly similar to that of the previous approach. But here instead of returning only the final hidden state vector, the hidden state of each encoder LSTM units is returned as output and this is fed into the Attention Unit.

Attention model works by computing a set of attention weights α(t, t'), this denotes how much attention weight should be given to word t' in natural language query for generating word t in SQL. Similarly we have T_x weights one for each element in the input sequence. This weights together give us the context vector that is fed as the input to corresponding decoder LSTM unit. The Attention Unit takes as input both the encoder hidden states and the previous output from the decoder and returns the context, which is fed as input to each decoder LSTM nodes. Hence this LSTM is referred to as Post Attention LSTM. This way the decoder can output a vector that maximizes the probability given the context vector. Each element in the output is labelled as y_t where 't' is the time step and the above steps are repeated T_y times with a new context vector for the current time step and the output of previous decoder as input. A Softmax is used on the decoder output to find the actual word and the final output sequence is a collection of words for the SQL query.

As mentioned in the Bahdanau et. al (2015), we used a bidirectional LSTM for encoder where the last hidden state of the backward LSTM in the encoder is fed as the first hidden state for the decoder. The input to the next decoder step is the concatenation of output from the previous step and the context vector. This way the decoder can output a vector that maximizes the probability given the context vector and the previous output. Each element in the output is labelled as y_t where 't' is the time step. A softmax is used on the decoder output to find the actual word and the final output sequence is a collection of words for the SQL query. In this paper, We are using a greedy approach to find the best possible output sequence, this could be improved significantly if we use a Beam search approach in finding the most probable sequence.

We also tried using two stacks of LSTM for the decoder, but this doesn't change the performance of the model. Hence we used only one level of Post Attention LSTM.

3.2.2 Probabilistic model

Classification Features[7]

Token-based Features: These features are based on learning tokens in a given sentence. If the token captures any symbols and these features capture the aggregates. We also took lower case form of a token as a feature for uniform learning. *Grammatical Features:* POS tags of tokens and grammatical relations (e.g. nsubj, dobj) of a token with other tokens in the sentence were considered. These features were obtained using the nltk in python Contextual Features : Tokens prior and following (local context) the current token was also considered as features. And their respective POS tag are considered. [15] *Other Features:* IsAttribute captures is token is an attribute that means if it is tagged to either select, from, where, group by or having clause. We tag it 1 if it is an attribute otherwise 0.

Model Selection

We used the conditional random fields in our scenario.We have explored all other probabilistic model and finally choose the conditional random fields because of its added advantages over Hidden Markov model and Maximum entropy Markov models. Each of these models are discussed and reasoned out that Conditional random is best choice is this task.

Conditional Random Fields and Maximum entropy Markov models. are a type of discriminative classifier which models the decision boundary between the different classes where as HMM is generative models which revolves around how the data is generated and then classification is done.

Hidden Markov Models: Hidden Markov Models is based on the Naive Bayes, which says the following:

$$p(y|x) = \frac{p(y)p(x|y)}{p(x)}$$

So, for prediction an outcome given x based on naive independence assumption we get

$$P(y|x) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1 ... x_n)}$$

HMM's are generative models which calculates the joint probability and are based on naive Bayes that the labels are independent of each other. The parameters are typically trained to maximize the joint likelihood which captured the probability over observation and label sequence. So, the model needs to enumerate all the possible observation. HMM is only dependent on every state and its corresponding observed label. So, this model wouldn't fit for the case where it should capture multiple interacting features or long-range dependencies of the observations. Due to this difficulty the conditional models are considered as alternatives. Conditional model specifies the probabilities of possible label sequences given an observation sequence. Furthermore, the conditional probability of the label sequence will rely on arbitrary, independent options of the observation sequence while not forcing the model to account for the distribution of these dependencies.

Maximum entropy Markov models: Maximum entropy Markov models (MEMMs) are conditional probabilistic sequence models that overcomes the limitations of that of HMM's.

From the Figure 1 in the appendix, we can observe that MEMMs are directed graphs unlike that of conditional random field. And we can observe that the directions are flowing from labels to that of input variable and flow is just opposite to that of Hidden Markov models.Suppose in the Figure 1 the contest size is 3 then Y2 depends on Y1 and the X1, X2, X3. And the conditional distribution would factor like logistic regression. And the conditional distribution equation is as follows:

$$P(\mathbf{y}_{1:n}|\mathbf{x}_{1:n}) = \prod_{i=1}^{n} P(y_i|y_{i-1}, \mathbf{x}_{1:n})$$

But the MEMMS has the label bias problem, that is observations far away don't impact the early predictions. For example, P(Y2/X) is only depend on the X1, X2, X3. As far as we could possibly know, CRFs are the main model class that does this in a simply probabilistic setting, with ensured global maximum likelihood convergence.

Conditional Random Fields: *Definition of CRF*. Let G = (V, E) be a graph such that Y = (Yv) vV , so that Y is indexed by the vertices of G. Then (X, Y) is a conditional random field in case, when conditioned on X, the random variables Yv obey the Markov property with respect to the graph:

P(Yv / X, Yw, w $\neq v$) = $P(Yv/X, Yw, w \sim v)$ where w $\sim v$ means that w and v are neighbors in G.[**?**]

CRF is a random field globally conditioned on the observation X. In CRFs, our input data is sequential, and we must take previous context into account when making predictions on a data point. To model this behavior, we will use Feature Functions, that will have multiple input values, which are going to be: Function of set of input vectors X, the position i of data point we are predicting, the label at i-1 and i. The purpose of the feature function is to express characteristic of the sequence that the data point represents. Each feature function is based on label of the prior word and present one. To build the conditional field we assign the weights to each feature function (in this case it is represented by lambda)[14]

And the lambda is parameter which we are trying to optimize. Probability distribution of conditional random field is as follows:

$$P(y, X, \lambda) = \frac{1}{Z(X)} exp\{\sum_{i=1}^{n} \sum_{j} \lambda_j f_i(X, i, y_{i-1}, y_i)\}$$

$$\text{Where: } Z(x) = \sum_{y' \in y} \sum_{i=1}^{n} \sum_{j} \lambda_j f_i(X, i, y'_{i-1}, y'_i)$$

Lambda is estimated by maximizing the likelihood. So if we consider the negative log on likelihood then it becomes minimization problem, and it order to that we can take the partial derivation of negative log likelihood with respective lambda. Once we get the derivation, we can use the gradient descent method to update the values of the lambda. The Following are the mathematical equations used.

$$L(y, X, \lambda) = -log\{\prod_{k=1}^{m} P(y^k|x^k, \lambda)\}$$

$$= -\sum_{k=1}^{m} log[\frac{1}{Z(x_m)} exp\{\sum_{i=1}^{n} \sum_{j} \lambda_j f_j(X^m, i, y^k_{i-1}, y^k_i)]$$

Negative Log Likelihood of the CRF Probability Distribution

Model Implementation

Step1: As the dataset has about 4k records, 0.6 of records are considered as training set and rest the testing set

Step2: As the template assignment is the multi classification problem, so natural language query data is trained on both Multinomial logistic regression model(MLR) and Support Vector Machine(SVM) where templates are passed as labels. And upon the testing, we observed the Support Vector Machine(SVM) outperforms the Multinomial logistic regression model(MLR).

Step3. Creating the features for each token in a given natural language query and then training on Conditional Random Fields with these features and clause attributes as labels. For implementation of conditional random field, the scikit learn package in python is used and parameters are optimized by grid search approach.

Step 4: Once the training is performed, for any

given new query it chooses template from the predictions by trained Support Vector Machine model and then finds the clause attributes from the trained conditional field model.
Step 5: Finally the clause attributes are inserted in the template based on clause matching. And also the text in natural language query is mapped to attributes, so that it can be inserted as the variables in the SQL output. For example: cities as *city code*, airport as *airport code* etc.

4. Results and Analysis

4.1 Probabilistic Model

4.1.1 Results from Multiclass classification for template selection:

```
Accuracies:
SVM:99.4%
MLR: 98.7%
```

Based on the accuracy's from the above results, we can observe that Support Vector Machine is performing better over the Multinomial logistic regression. So we stick to SVM model for template selection.

4.1.2 Results from Conditional random fields:

Refer the confusion matrix for the Clause Attribute assignment using Conditional random field in Appendix section (Table2). From the table we can observed that the F1 score for all the clause except the select is clause is good enough where as F1 score for select clause is 0.62

4.1.3 Output SQL query:

Input natural language query is :*"what is airline ff"*
Template selected using the SVM model is :

```
['SELECT', 'DISTINCT',
'FROM', 'WHERE']
```

Output of Conditional random field for the input natural language query:

```
['O', 'O',
'where attribute clause',
'where value clause']
```

SQL Query:

```
SELECT DISTINCT airline_code
FROM airline WHERE airline_code =ff
```

We can observe that for the given natural language query, the model is producing the relevant code. And it is also observed that, the model is working well with the simple template, but not working well on the complex templates which includes the sub queries.

4.2 Sequence to sequence models

The sequence to sequence models generated SQL query that greatly corresponds to the input Natural Language query. Both the encoder-decoder model and the attention based model recognized the relation names and the column names for the respective query as shown below.

Expected Output Query

```
SELECT DISTINCT flight.flight_id
FROM flight WHERE (
flight.from_airport IN (
SELECT
airport_service.airport_code
FROM airport_service WHERE
airport_service.city_code
IN ( SELECT city.city_code FROM
city WHERE city.city_name =
'DALLAS' )) AND
flight.to_airport IN ( SELECT
airport_service.airport_code
FROM airport_service WHERE
airport_service.city_code IN
( SELECT city.city_code FROM city
WHERE city.city_name = 'PHOENIX'
)) )
```

Encoder-Decoder Output Query

```
SELECT DISTINCT flight_1.flight_id
FROM flight flight_1 ,
```

```
airport_service airport_service_1 ,
city city_1 , airport_service
airport_service_2 , city city_2
WHERE flight_1.from_airport =
airport_service_1.airport_code AND
airport_service_1.city_code =
city_1.city_code AND
city_1.city_name = 'BOSTON'
AND flight_1.to_airport =
airport_service_2.airport_code
AND airport_service_2.city_code =
city_2.city_code AND
city_2.city_name = 'PHOENIX'
<EOS>
```

Here, the first query is the expected SQL query and the second one is the output generated by the encoder-decoder model, though the actual and output query looks different in structure, the relations considered by the predicted output is exactly the same as the one mentioned in actual query. To evaluate the performance of the above model we used 'accuracy' metric provided with keras, even though, this metric will over estimate the performance of our models. This accuracy metric check if an element in the output query is present in the input decoder query. We used this as our metric since it is easy to evaluate but this is not the best metric as it won't consider for syntactic and semantic structure of the SQL. Since, the strategy we used is similar to that of machine translation we considered using Bleu Score, but it won't work as unlike Machine Translation, the text to SQL translation requires creating queries that follows a fixed structure and hence bleu score won't be a good metric. The best metric for this work is by considering a metric that measures whether the SQL query is executable without any syntax errors and it returns the desired records when run on a SQL compiler.

The output of the Attention Based model is similar to that of encoder-decoder model, the query below is SQL query generated by Attention Model for the above query.

Encoder-Decoder Output Query

```
SELECT DISTINCT flight_1.flight_id
FROM flight flight_1 , city
airport_service_1 , city
airport_service_1 ,
airport_service airport_service_2 ,
city city_2 WHERE
flight_1.from_airport = AND AND
airport_service_1.city_code = =
= = = ) AND AND = = AND AND = )
```

The queries generated by both the encoder - decoder model and the attention based model is SQL like, but the encoder-decoder model generate more relevant query compared to that of the Attention based model. The relations and the object generated by the encoder-decoder are more relevant. For example, the encoder decoder model generates the following relation and objects flight flight_1 , airport_service airport_service_1 , city city_1 , airport _service airport_service_2 and city city_2 which are aptly named and numbered. Also, the encoder-decoder model learned to join the multiple relation based on the correct key columns. For example, in the following WHERE condition the flight and airport relations are joined by correctly by the columns from_airport and airport_code as both the columns as the airport code.

```
flight_1.from_airport =
airport_service_1.airport_code
```

And similarly, it joins the relations airport_service and city by the common column city_code.

```
airport_service_1.city_code =
city_1.city_code
```

The main drawback of the query generated by Encoder-Decoder model is even though the query generated is syntactically relevant it failed to capture the template structure of the actual query. This could be achieved by adding another LSTM layer as a parallel channel that classifies the template of the query and then using the class of the template as the part of encoder LSTM's input. We are unable to achieve it practically due to computational power limitations. One another limitation is that it fails to model the variable names in the query

based on the variable name in natural language input. This could be done by adding Named Entity Recognition (NER) tags for Date and Cities in place of the actual entities in the natural language query. But we are unable to capture it as the input natural language query were in lower case and the NER tagger in both nltk and spacy will not be able to tag the entities that are in lower case. If the above two limitations are fixed, the performance of the model can be increased even further.

5. Conclusion

Natural Language Interface for Database is the need of the day to bridge the gap between technical and non-technical users of the database system. Our approach in creating a text to SQL system for NLIDB uses sequence to sequence architecture like encoder-decoder and attention-based architecture and a probabilistic approach using conditional random field (CRF) along with paraphrasing. The end to end deep learning approaches creates a better translation of Natural Language Queries to SQL queries compared to that of conditional random field. This is because the CFR model works well for shorter queries but not for complex queries, since it is difficult to capture long term dependencies in such approach.

The efficiency of sequence to sequence methods can be increased by using a beam search rather than the greedy approach that we implemented for finding the best possible sequence. Also a better translation can be achieved by masking the entities to their type rather than using the entities itself directly. There are many other handcrafting techniques that could be used to check if it increases the accuracy of the translation. We can also take advantage of availability of number of database sources and a thriving population of data base users, we can make use of such resources to manually tag the database queries and to create Natural Language and SQL query pairs using active learning. This will generate a lot of labelled data that will greatly aid in creating powerful deep learning models.

Appendix

Figure1:Maximum entropy Markov models

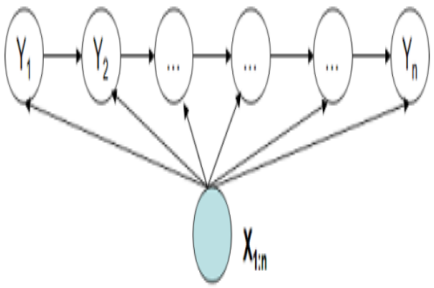

Table 1. Attribute Classification

Token	Attribute	Tag
list	0	O
all	0	O
the	0	O
flights	0	SELECT
that	0	O
arrive	0	O
at	0	O
general	0	O
mitchell	0	WHERE
international	0	O
from	0	O
various	0	O
cities	0	WHERE

Table2 :Confusion Matrix from CRF models

Clause	Precision	Recall	F1 score
Attribute			
0	1.00	1.00	1.00
Select clause	0.69	0.561	0.621
Group-by clause:	1.00	0.86	0.92
Where attribute clause	0.89	0.96	0.93
Where value clause	0.97	0.92	0.95

Acknowledgments

We are very grateful to Professor Damir Cavar for imparting valuable Natural Language Processing Concepts throughout the semester. The concepts such as Recurrent Neural Networks, LSTM were the foundations to models like Encoder-Decoder and Attention Based Neural Network which have been implemented in our project. We also want to extend a special thanks to Atreyee Mukherjee for her valuable feedback on our assignments which greatly helped us.

References

[1] Yoav Artzi, Kenton Lee, and Luke Zettlemoyer. Broad-coverage ccg semantic parsing with amr. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1699–1710, 2015.

[2] Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1:49–62, 2013.

[3] Michael Auli, Michel Galley, Chris Quirk, and Geoffrey Zweig. Joint language and translation modeling with recurrent neural networks. 2013.

[4] Amittai Axelrod, Xiaodong He, and Jianfeng Gao. Domain adaptation via pseudo in-domain data selection. In *Proceedings of the conference on empirical methods in natural language processing*, pages 355–362. Association for Computational Linguistics, 2011.

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

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[7] Raffaella Bernardi and Manuel Kirschner. Context modeling for iqa: the role of tasks and entities. In *Coling 2008: Proceedings of the workshop on Knowledge and Reasoning for Answering Questions*, pages 25–32. Association for Computational Linguistics, 2008.

[8] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[9] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[10] Jianfeng Gao, Xiaodong He, Wen-tau Yih, and Li Deng. Learning semantic representations for the phrase translation model. *arXiv preprint arXiv:1312.0482*, 2013.

[11] Alessandra Giordani. Mapping natural language into sql in a nlidb. In *International Conference on Application of Natural Language to Information Systems*, pages 367–371. Springer, 2008.

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[13] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. Learning a neural semantic parser from user feedback. *arXiv preprint arXiv:1704.08760*, 2017.

[14] Andrew McCallum and Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 188–191. Association for Computational Linguistics, 2003.

[15] Ashish Palakurthi, SM Ruthu, Arjun Akula, and Radhika Mamidi. Classification of attributes in a natural language query into different sql clauses. In *Proceedings of the International Conference Recent Advances in Natural Language Processing*, pages 497–506, 2015.

[16] Ana-Maria Popescu, Oren Etzioni, and Henry 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.

[17] Amany Sarhan. A proposed architecture for dynamically built nlidb systems. *International Journal of Knowledge-based and Intelligent Engineering Systems*, 13(2):59–70, 2009.

[18] Jian Su, Kevin Duh, and Xavier Carreras. Proceedings of the 2016 conference on empirical methods in natural language processing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016.

[19] Adrienne Wang, Tom Kwiatkowski, and Luke Zettlemoyer. Morpho-syntactic lexical generalization for ccg semantic parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1284–1295, 2014.

[20] Sida I Wang, Percy Liang, and Christopher D Manning. Learning language games through interaction. *arXiv preprint arXiv:1606.02447*, 2016.