DataChat_

Urjit Patel, Vincent Chabot
Center for Data Science - New York University
{up276, vec241}@nyu.edu

***Abstract*—Chatbots, programs which are able to interact with humans in natural language and and potentially perform actions to meet user's request (like booking a flight), are considered as the future of our interactions with the digital world in which we live. End-to-end chatbots are constituted of several building blocks and one of them is the Natural Language Understanding (NLU) block, that allow the program to understand the user's request. Our work focuses on developing the NLU block of a Question Answering chatbot and more precisely, of a end-to-end chatbot developed by Zelros (zelros.com) aiming at building conversational BI.**

I. Chatbot: Context & Motivations

We used to interact with computers via command line, which was later on replaced by the graphical user interface and point-and-click mouse. Smartphones and touchscreens changed those interactions into swipes and taps, but yet it requires downloading many apps and still takes time to have to switch from one application to another. Think about planning a weekend out of the city : you need first to check airbnb, open google to compare the flights, and you'll also have to juggle between Uber Yelp and other Trip Advisor to find what to do on site. Chatbots, programs which are able to interact with humans in natural language and and potentially perform actions to meet user's request (like booking a flight), are currently considered as the future our interactions with machines.

But although there is currently a big hype around chatbots and good reasons to believe that they will become the way we'll interact with our world in the future, we are still very far from chatbots being able to handle full conversations as humans would do and perform all the tasks we would like them to perform. Even if chatbots are generating big hopes and are evolving quickly, they can still only perform limited task and need to be task specific for the moment in order to perform well. Building an effective is still a big challenge and a lot of research is currently ongoing on managing language interactions with a human, i.e Natural Language Understanding (NLU) / Natural Language Processing (NLP).

II. Project Overview

A. Building the NLU block for a conversational BI chatbot

Our work focuses on developing the NLU block of a Question Answering chatbot and more precisely, of a end-to-end chatbot developed by Zelros (zelros.com) aiming at building conversational BI, that is to say a natural way for people to interact with companies data. Indeed, data sharing and data accessing is a real pain point within companies. It usually requires to use SQL-like query, open often out of date source files, or even worse, email exchanges between business users and data owners. Solving the conversational BI challenge would for instance allow a sales manager to ask to the chatbot questions such as "what is the average sales amount per vendor for last month" and get the answer instantly without having to open any data files, write in some SQL request or ask his data analyst and then have to wait for the answer.

B. NLU technical problem formulation

We formulate here the technical task we have to solve. Given a database to which we want the chatbot to be able to answer questions about, unique questions are generated ('What is the number of users', 'what is the number of cars', ...). We call these questions 'database questions'. How the questions are generated is out of the scope of our work. **The NLU technical task to which we have to answer is therefore the following : Given a user question, find the database question, if any, that is semantically speaking the same question.** Hence, the formulation of the questions should not influence the results and only the semantic is important. If the user question is out of bound, i.e. has no semantic equivalent in the database questions, we should return 'fallback'. Furthermore, we also allow ourselves to ask the user to clarify his question, shall the system detect that more than one database question potentially matches. We should then return the potential matching database questions and asking her/him to chose the correct one.

The general framework is summarized in fig. 1.

III. Project Framework

A. Introduction: Challenges & Related Work

The first challenge we encountered was to find the data to build our model. One one hand, we had no user company data, and on the other hand, there was no publicly available data closely matching our use case, that is to say : 1) very short questions (few words), and not always well formulated since users often ask questions by just throwing out key words such as for Google request, and 2) questions grouped by classes (many classes of few instances), one class being all the possible formulations of one question. The second challenge we encountered was the lack of litterature related to

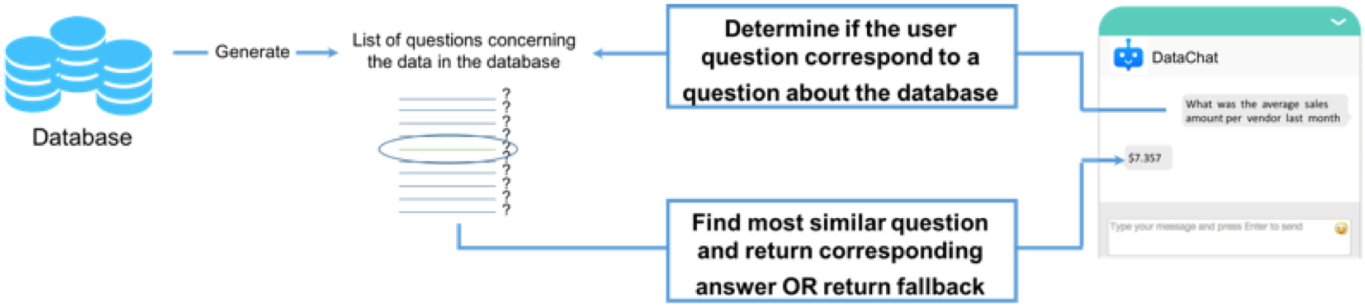

Fig. 1. ChatBot Global Framework

our task (finding semantic similarity between short questions). Semantic similarity is usually tackled at the word embedding level (word2vec), or for longer sentence / documents, but is harder on questions or very short amount of not always well structured text.

It was really hard to apply Deep Learning approach due to scarcity of data matching our use case. Instead, we decided to implement a similarity framework using pre-trained words / sentence embeddings. Our framework is made of 3 blocks:

- From pre-trained words/sentence embeddings, build a good semantic, vectorized representation to represent questions.
- Measure the similarity between two questions vectorized representations
- Potentially be able to return multiple answers if it is not clear which data base question the user is referring too. We also want our system to be able to detect if a user question does not have any equivalent in the database, that we will later refer to as 'fallback'.

B. *embeddings*

We use 2 types of pre-trained embeddings. Whenever we need to use word embeddings we use word2vec Google vectors pre-trained on Google News articles. Word2vec was first introduced by [1]. As for sentence embeddings, we use Skip-Thought pre-trained vectors introduced by [2]. The fact that we lacked data made it not relevant to train our own word2vec embeddings, and training on any dataset would just not allow to outperform results obtained by simply taking these pre-trained embeddings. The reason for choosing word2vec as word embeddings is because these embeddings are specifically built based on semantic. With word2vec, 2 synonyms will have a similar vector representation (and therefore a cosine distance close to 1).

C. *3 Similarity Frameworks Experimented*

We've implemented three different similarity frameworks that we describe here and discuss results in the Experimental Evaluation section.

Similarity framework #1: The first similarity framework we implemented is the following : we represent the questions as vectors using Continuous Bag of Words (CBoW) introduced by [1], and we compute the semantic similarity between two questions by computing the cosine distance between the two vectors. CBoW is just the fact to take for a sentence embedding the average of the words embeddings in the sentence, thus the bag of word denomination. And Continuous because sentence embeddings lie in the continuous space of the word embeddings. The intuition for using a simple model as CBoW is that as word2vec embeddings can be considered as a semantic representation of the words, averaging over the words of a sentence will give an embedding carrying the meaning of the sentence. We can then find the similarity between two questions by simply computing the cosine distance between the two questions vectors. Fig. 2 summarizes our framework.

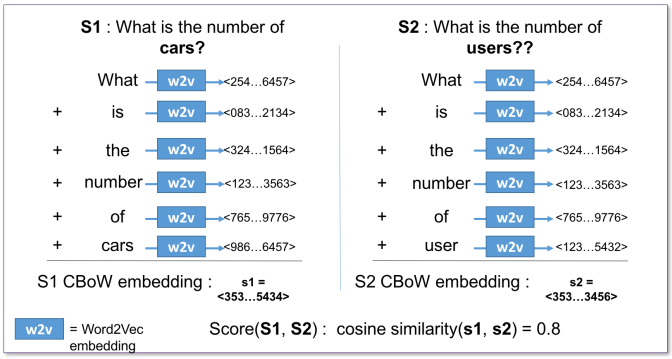

Fig. 2. Similarity framework #1

An improvement of this framework is to replace the average we perform in CBoW by a weighted average. Indeed, not all are not as important in a question and many words can be considered as noise, such as words used for formulation. For instance, 'what is the number of cars in my database?' and 'what is the number of users in my database?' will have a very high cosine similarity even if they are two different questions. To solve this, we compute CBoW using a weighted average using the Inverse Document Frequency of the words (the Idf of the Tf-Idf) as weights. We compute those Idfs on a large database of documents we collected on a relevant field.

Therefore, the embedding E of a questions q is computed as follow :

$$E(q) = \frac{1}{\#ofwords} \sum_{word \in q} Idf(word).w2v(word)$$

Where $w2v(word)$ represent the word2vec embedding of word.

This framework allows to reach satisfactory results but nevertheless, even when using the Idf-weighted average, it still does not allow to discriminate very well when some questions are semantically somehow different but similarly formulated questions. Fig. 3 illustrates this problem.

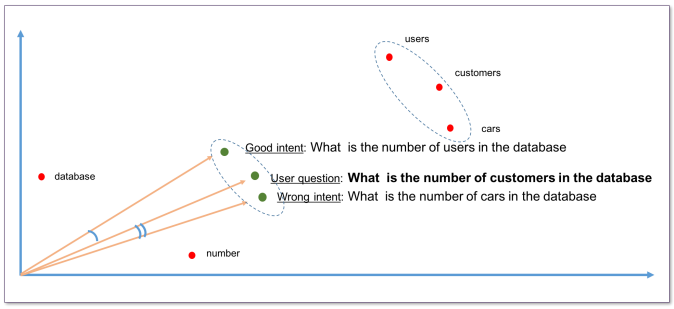

Fig. 3. CBoW questions embeddings projected on a 2D plan using PCA

Similarity framework #2: In a second framework, we experiment with sentence2vec embeddings, and more particularly with Skip-thought vectors introduced by [2]. The reason we chose these embeddings is because they are trained with the same idea as word2vec embeddings: the dataset is made of triplet of sentences (s1, s2, s3), and by taking s2 as input, the objective is to predict s1 and s3. Nevertheless, simply computing the cosine distance between skip-thought question embeddings does not give any satisfactory results. The first reason we can think of is that skip-thought is not specifically trained to embed semantic, and thus skip-thoughts vectors probably carry as much about structure, grammar and so on that semantic. To solve this issue, replace the dot product between two questions q_1 and q_2 by a bilinear product $q_1 W q_2$, where W is a matrix we learn and hope it will enable to "filter" the dimensions of skip-thoughts vector that carry semantic meaning (if any !). We initialize W as the identity matrix. Therefore, before training, $q_1 W q_2$ is just the dot product. We use mean-square error + L2 regularization to learn W. The framework is presented on fig. 4.

Similarity framework #3: Lastly, we implemented a more elaborated framework based on the similarity framework #1 using word2vec embeddings, and try to find a better way than averaging. Following the idea that many words carry much more structure/formulation information than semantic information, and therefore are noise to our model, we first filter questions using Part of Speech (POS) words tags. For instance, we could decide to keep only nouns, verbs and adjectives from a questions, as we could argue these words convey most of the semantic in a sentence. Second, for each filtered word in

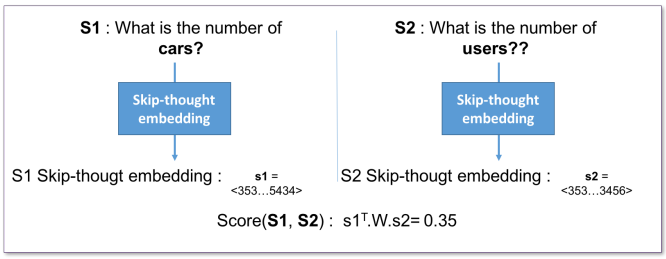

Fig. 4. Similarity framework #2

q_1, we compute the cosine distance with the filtered words in q_2 and keep the maximum of these cosine distances. The intuition here is that if a concept (noun, verb, quantifier, ...) is there in the 2 sentences, a contribution of 1 should be added to the final similarity between q_1 and q_2, otherwise this word should not bring any contribution to the final score. Word2vec is particularly useful there since if two synonyms are used, they would still result in a high pairwise cosine similarity. Then, we simply take as the similarity between q_1 and q_2 the average of all the maximum pairwise word similarities. Nevertheless, we observe that cosine similarities between different words are sometimes non negligible and can go up to .3 - .4, when synonyms can go down as low as .6 - .7. This can affect the results and therefore, in order to solve this, we introduce a non linear filter with the objective to push words pariwise cosine similarities below 0.5 towards 0 and those above 0.5 towards 1. The filer we built is shown on fig. 5 and has the following equation:

$$y = 1/(1 + np.exp(-m * (x - 0.5)))$$

where m is a tunable hyperparameter (we usually chose m = 30)

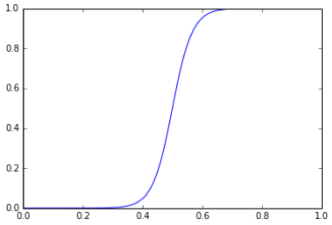

Fig. 5. Non linear filter used to improve difference between words pairwise similarities greater and lower than 0.5

The final score between two questions is therefore the following:

$$Score(q_1, q_2) = \frac{1}{\#ofwords} \sum_{w_i \in q_1} \max_{w_j \in q_2} filter\big(cossim(w_i, w_j)\big)$$

We synthetise our final framework in fig. 6.

D. *Adding multichoice and fallback framework*

Finally, we want our system to be able to return multiple choice if it is not clear which question the user refers to, or also

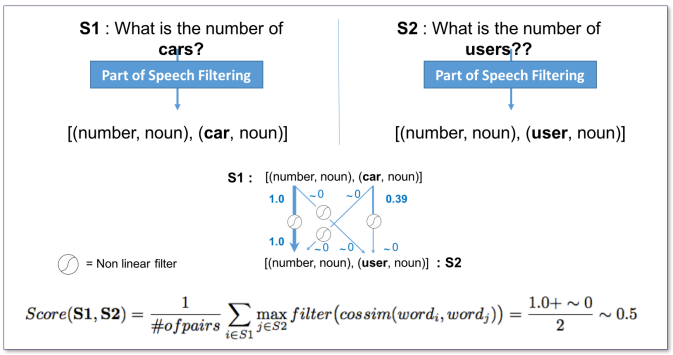

Fig. 6. Similarity framework #3

be able to determine when a user question has no equivalent in the database. In order to do this, we use margin sampling: we compute the distances between the user question and all the questions in the database (using any method 1, 2 or 3). We use a softmax to have those distances sum to 1 to get a distribution. We then use the the pseudo-distance to compute the distance to the vectors of the basis of the n-modal distributions (and then we select the basis vector with the highest component):

- The mono-modal distribution (we pick the answer with the highest probability)
- The bi-modal distribution (we ask whether the user wants the first or second most probable question)
- ... and so on up to the uniform distribution

We decide to define fallback when we have more than 3 possible answers. The pseudo distance has the following form:

$$\mathrm{d}(\vec{x}, \vec{y}) = \left\| \vec{x} - \frac{\vec{x} \cdot \vec{y}}{\|\vec{y}\|} \vec{y} \right\|$$

Of course, allowing ourselves to return multiple answers significantly allows to improve our results.

IV. Experimental Evaluation

A. Data

As mentioned earlier, we built a similarity framework using pre-trained embeddings, hence there was no training involved and data only mattered for computing accuracy. Therefore, we built a dataset of 30 pairs of questions (Database question, User question), where User question is simply a reformulation of the Database question. Of course, the more the questions, the more difficult to recover the true one. But 30 questions is about the typical number of questions per database the company we are working for encounters. In addition of this data set we built, we downloaded questions from the Yahoo finance question data set, in order to compute Idf for the first similarity framework.

B. Methodology

We simply use accuracy as our evaluation metric. We take as input one of the 30 user questions, and return 1 when our system was able to identify the true Database question (meaning we would be able to return the user the corresponding answer), 0 otherwise, and we average over these scores.

C. Results

For our first similarity framework with regular average, we achieved 0.63 of accuracy on the dataset containing the 30 user questions. When taking the Idf-weighted average, we slightly improved up to 0.66. We suspect the little improvement to be because of the fact that we compute the Idfs on a dataset which is not our working data set (we used Yahoo finance question data set). We highly recommand recomputing Idfs using user questions once more data is collected.

For our second similarity framework, using raw pre-trained skip-thought vector did not allow to recover any of the database questions. This is because of the nature of our questions (very short) and because skip-thought vectors are not specifically trained to be used for semantic tasks. We then implement the bilinear similarity product where we learn W. Of course, we know having only 30 true examples will not allow us to learn a matrix of size 3800x3800 (skip-thought vectors dimension is 3800). But we are able to show that this framework is actually able to learn. The learning curve is presented on fig. 7. This shows that once enough user data collected for training, this could be a good lead to explore.

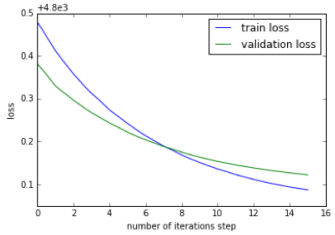

Fig. 7. Learning the projection matrix

For our third similarity framework, our system achieved 0.77 which was the best accuracy we got by returning only single answers.

Finally, when allowing ourselves to return multiple answers (up to 3, predicting fallback otherwise). If the system has a doubt, it should return up to maximum three choices to the user, and if the true database question is in this proposal set, we still count true. With this, we were able to achieve 0.86 accuracy, our best score overall.

V. Conclusion & Future Work

To conclude with, we were able to build a system that achieves **0.86 accuracy** performance with no data to allow any learning. We consider this framework as achieving good results given the lack of data. Of course, 0.86 accuracy is not enough for a question answering machine, as getting one to two wrong answers ever ten questions can get very annoying. But this could be a good way implement a first version of a bot and start collecting data that would then allow to take a Machine Learning / Deep Learning approach.

This is why we think that future work could be done on the following axes:

- Words embedding : train own word2vec embeddings using company data
- Try learning properly the matrix in the similarity framework # 2 with appropriate amout of data
- Explore Deep Learning approaches. There are many available models that could work like RNN models (LSTM, GRU, ...) with any kind of attentions, Neural Tree Indexers, ...

VI. Acknowledgement

First, we want to thank our advisor Pr. Silva his supervision. We would also like to thank all the persons that spend time discussing our ideas throughout our project, including Tian Wang and Pr. Bowman. Finally, we would like to specifically thank Zelros for proposing this project and for their always very insightful ideas and feedbacks.

References

[1] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[2] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302, 2015.