QuestionAnswer Chatbot

Cristian Di Pietrantonio

September 16, 2017

1 Introduction

The importance of intelligent and autonomous question answering systems has grown rapidly in applications like user support, where (possibly many) users ask questions about products and an answer is required in short time. In this cases, human operators are not enough, so here it comes the necessity of automatic answering systems.

The present work introduces a simple version of the above-mentioned type of systems. It uses Telegram, a popular and powerful messaging service, as interface to users. The bot is designed to ask and answer questions about entities with the help of a knowledge base, which links known entities according to a predefined set of binary relations.

2 System model and architecture

In this section, the system model and the environment in which it lives are presented. There are four systems that together provide the question answering service: the *Knowledge Base Server* or *KBS* for short, the *Telegram* service, *BabelFy* and the *QAChatbot* system. The latter is the one developed, whereas the first two are existing services. Nevertheless, let's review some of their characteristics.

The KBS can be abstracted as a dataset of question-answer pairs. Each pair comes with additional information (other than the question and answer strings) such as the entities that are mentioned in the exchange and the relation between them. The chatbot has to use this information to answer future questions. Users can use the Telegram messaging service to interact with the system, exchanging messages with the system's *bot*. Telegram allows to create accounts (bots) and using them programmatically. The advantage of using Telegram is that we don't have to implement a human interface for people to interact with the chatbot. The only (easy) effort is to use Telegram API to let the system exchange messages through a bot. BabelFy is a system capable of disambiguating sentences; it will be used to identify entities addessed in conversations. Figure 1 depicts what has been described until now.

The QAChatbot system is a rather complex on its own, presenting several modules. It follows a brief high level introduction to the ones that define the system's structure and behavior. The *Chatbot* module handles the interaction with a user, and it is the one that sends and receives messages, decides the next action to take influencing the conversation flow. When an intelligent evaluation is required (e.g. to answer a question) the *Brain* module is used. It relies on machine learning models and heuristics to allow the chatbot to understand and reply in a consistent way to the user's input. The chatbot has access to data through the *DataAccessManager* module. It handles both the interactions with the Knowledge Base System and the local representation of the dataset, which is called *Knowledge Graph.* The knowledge is used when looking for an answer, to register one, or to formulate a question.

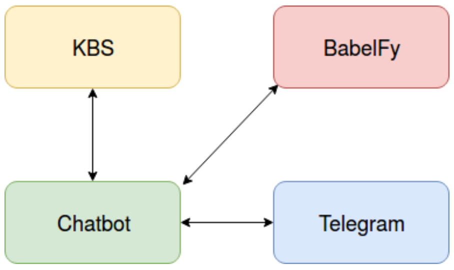

Figure 1: The system environment.

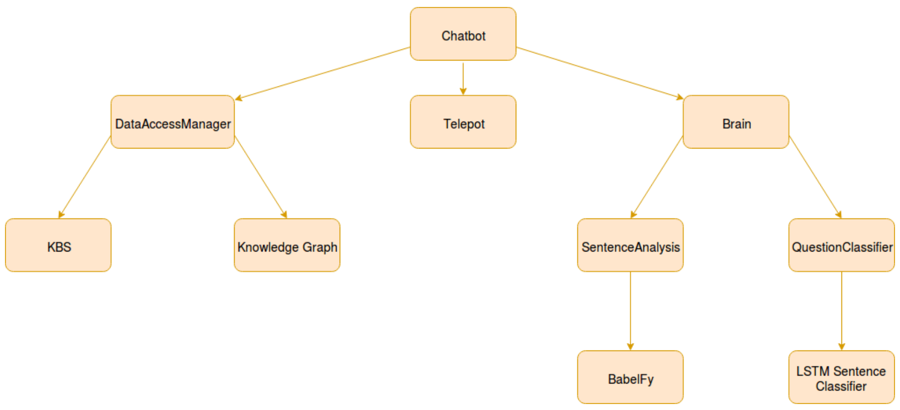

Figure 2: The chatbot modules.

2.1 Conversation flow

The Chatbot class handles the conversation with a user. Each user has an associated Chatbot instance created by the telepot API when they first contact the bot. The structure and the flow of the conversation is modeled as a finite state automaton. In the initial state, the bot is waiting for a user to make the first contact; when it happens, the bot invites the user to choose a domain and transits to the DOMAIN state. The user replies, and the bot checks that the selected domain exists. The next state is DIRECTION, where the system must determine whether the user wants to ask a question or to let the bot doing so. The interlocutor can use a set of keywords to let the bot start asking, otherwise the input coming from the user is considered a question. After the bot formulates a question it enters in WAITING_FOR_ANSWER state; when the answer comes, it is registered in the knowledge base and the exchange is completed. Likewise, when it is the bot that must answer, it replies with the computed answer to the user. At the end, the bot asks the user who is going to formulate a question in the next exchange.

2.2 The knowledge graph

The system creates a local copy of the Knowledge Base in order to optimize its use. In particular, the knowledge is modeled through a directed graph where each node is an entity or a concept (identified as babelnet ids), and there is and edge between two entities or concepts for each relation that binds them (multiple edges between two nodes makes this object an extension of a graph). Each edge is annotated with the KBS entry that led to the edge creation. It takes (semi) constant time access a specific node's outgoing and incoming edges, thus this representation is convenient when looking for a relevant concept (answer) given another concept and a relation (question).

2.3 Implementation notes

In this paragraph, some details of the implementation are discussed. The system must handle multiple conversations at the same time; for this reason, it is realized as a multiprocess and multithread application. The telepot API automatically allocates a thread for each conversation, instantiating the Chatbot class. Every Chatbot class instance must use the Brain module to perform question answering using, among other tools, a neural network model. It is infeasible in terms of memory and performances to load a model for every conversation. Instead, a dedicated process, reachable through the network, loads the model once and replies to the prediction requests coming from different Chatbot instances. Every other operation, e.g. data lookup, is done on the conversation thread, since they do not alter performances (at least in this basic implementation).

3 Querying

During the *Querying phase*, a user sends a message to the bot asking some kind of question about a concept. Upon receiving a question, the chatbot calls the "answer_question" function defined in the *Brain* module. The function performs the following steps:

1. requests a prediction to the remote *Question Classifier* server. It hosts a LSTM model that associates to the input question one of the predefined relations expressed in the project requirements.

2. Extracts the entities that can be identified in the question. Two type of questions are supported: the ones that ask if there exists a particular relation between two entities (e.g. Is Rome in Italy?) and the ones that ask the concept associated to a given entity through a given relation (extracted from the question, e.g. Where is Rome?). Entities extraction is done using both BabelFy and a POS tagger: among all the BabelFy annotations, the system takes the one with a "NOUN", "PROPN" or "NUM" tag.

3. Queries the Knowledge Graph in search for the entity (or entities) and its outgoing edges annotated with the relation at hand. For example, if the question to be answered is "Where is Rome?", the systems look for the "Rome" node and outgoing edges from that node associated to KBS entries having PLACE as relation. Returns all the entries that satisfy these conditions.

4. Each entry has an associated question, that is, where that entry comes from. The system compares the current question to the questions retrieved from the Knowledge Base to find the most similar one. A function as been defined such that it takes two sentence parse trees and outputs the degree of similarity that ranges from 0 to 1.

5. Returns the answer to the most similar question.

In the following subsections, the Question Classifier and the sentence similarity function are investigated more in depth.

3.1 The Question Classifier

To answer a question it is necessary to extract the relation which the inquired entity is supposed to take part to. To achieve this, a LSTM neural network is trained to associate a relation to a sequence of vectors representing the question. In particular, the key factor is the representation of the input question as a sequence. In a fist attempt, distributed word vectors were used to code words but the resulting accuracy was incredibly low ($\approx 24\%$). Next, the simpler bag of words approach has been chosen (i.e. associate to each word an unique index in a vector), but being careful on which words to retain and which to discard as OOV (out of vocabulary) words.

For every relation there is a set of words that are more frequent than others in the associated questions. It is reasonable to assume that more frequent words characterize a class of questions (e.g. the "Where" word is characteristic of the PLACE relation). It follows that the bag of words technique can be applied using as vocabulary only the frequent words of every class of questions; a sequence of words is translated in a sequence of one hot vectors, while OOV words are represented using the zero vector. Lastly, it remains to define what frequent means. A word w with frequency $f(w)$ is frequent if

$$f(w) \geq avg + \frac{1}{2} std,$$

that is, $f(w)$ must be greater than the average frequency of words in that class plus half of the standard deviation. This method has been tested by dividing the knowledge base evenly in two parts: training and test set. It has proved to be an effective one, with an accuracy of 96.9% and a F1 score of 85.9%.

3.2 Sentence similarity

In order to search the Knowledge Base for an answer that might have already been given, it is useful to define a function that computes the degree of similarity between two sentences. One way of doing it is by comparing their syntactical structure, other than the presence of the same words. Consider the roots of the dependency trees of the sentences. The amount of the same type of dependencies with their children, in the same order, plus the presence of the same words, should contribute in the degree of their similarity. Repeat this comparison recursively and in parallel using the input trees and what comes out is the Algorithm 1.

The algorithm returns a natural number n that can be greater than one. To normalize the value, the maximum possible value m is needed; it is given by simply computing the function using as parameters two pointers to the bigger tree (the longer sentence). The normalized value is $\frac{n}{m}$. Two equal sentences will result in a similarity index of 1.

4 Enriching

In the Enriching modality the bot asks question to users in order to increase its knowledge. The following steps are performed:

1. First of all, a subject must be picked such that it belongs to the domain that has been chosen by the user; if there are no known entities in that domain, a subject is picked randomly among all the nodes (entities) in the Knowledge Graph. We are given the following mappings that we can use to accomplish this task: entities to domains, domains to relations and relations to question patterns.

```
Input: root1, root2
$v1 \leftarrow 0$
$v2 \leftarrow 0$
if $root1.lemma = root2.lemma$ then
    $v1 \leftarrow 1$
end
if $root1.dep = root2.dep$ then
    $v2 \leftarrow 1$
end
$value \leftarrow v1 + v2$
for $child1$, $child2$ in $node1.children$, $node2.children$ do
    if $child1.dep = child2.dep$ then
        $value \leftarrow value + 1$
    end
    $value \leftarrow value$ + sentence_similarity($child1$, $child2$);
end
return $value$
```

Algorithm 1: sentence similarity.

2. Next, the system selects a relation such that the subject's node lacks of an edge associated to that relation in the Knowledge Graph; if the domain is present, only the compatible relations are considered.
3. Now that the bot has a subject entity and a relation to ask about, a question is generated using the available question patterns.
4. Finally, the user's response is analyzed using BabelFy and a dependency parser in order to extract the most likely entity to be the answer, that is, a BabelFy annotation which is a "direct object" or "object of preprosition" and that is different from the entities in the question. The data gathered so far (question, answer, entities extracted) compose a new data entry that is sent to the Knowledge Base Server. After having sent the data to the server, the system fetches new entries (the one just inserted but also the ones coming from other bots) and updates the Knowledge Graph.

5 Conclusion

The work presented implements a question answering chatbot that uses a knowledge base to provide a response the user. Both machine learning and heuristics are used, since different tasks have better results with one technique rather than the other; for example, when selecting the most similar question in the querying modality, an heuristics is used instead of a machine learning model. The reason why is that it would have been more computational expensive to learn structural differences between sentences with neural networks rather than working with that structure explicitly. Instead, in the relation prediction task, a recurrent neural network was the perfect fit, being a classification problem.

6 Notes

Attempts to do a proper quantitative evaluations were done, but with no luck. A lot of entries are simply wrong (ask for example "Where is Italy?" or "Where is Rome?"). Furthermore, the system relies heavly on BabelFy to perform the correct identification of entities. Sometimes this does not happen (e.g. sometimes compound entities are split) leading to a mismatch to what is in the Knowledge Graph and what is retrieved by analyzing the current question of a user. More example of entries that lead to these problems can be found in the file “problems.txt”.