# Natural Language Processing, Final Project Report

Dorjan Hitaj, Matricola 1740478

30 September 2017

## 1 Abstract

The term chat-bot refers to a software program that responds questions in natural language. Chat bots can be used for many purposes. They can be used in a business to aid users, provide them customer service, promote products and more. A specific kind of chat-bot which is known as ***Knowledge Bot***, is a type of bot that attempts to give answers to general questions about facts in specific domains. The goal of this project is about building such a bot.

**Keywords**: Chat-bot, Convolutional Neural Network, Part of speech tagging, Babelnet, Babel-domain, Wordnet, Sentence similarity, Mongo DB, Word vector.

## 2 Introduction

In this project we are required to build a **Knowledge Bot**. The bot should be able to "understand" and answer user questions about which it has information, and also be able to expand its knowledge by asking the users for information about concepts it has few or no information at all in its knowledge base. The first interaction is known as **Querying interaction** and the latter is known as **Enriching interaction**. The initial data for the bot will be a set of question-answer pairs classified in 16 specific relations that are stored in a remote server. To tackle this problem I have used neural networks for some tasks and heuristics for some others, which are explained in more details in following sections.

In the **Querying** mode I have used a *convolutional*[1] neural network to define the relation in the user's question. I have used part of speech tagging to extract concepts from the user's question. For an efficient and fast searching in the knowledge base I have used a Mongo Db instance, which I have indexed to make possible performing full text search on it, thus increasing the chances of getting an answer if it is there. After searching, the returned results are filtered by the relation predicted and then a similarity coefficient is calculated among the user's question and the filtered results, by means of wordnet[2]. In the end the **answer** of the question with the highest similarity coefficient is returned to the user.

In the **Enriching** mode, a concept is picked in the domain chosen, then it's lemma is obtained via Babelnet and the knowledge base is searched to check the amount of information present for that concept. Depending on the amount of information, a question is generated and presented to the user for him to answer. These two directions of interaction and their implementation is explained in the following sections.

## 3 Querying Interaction

### 3.1 Overview

The **Querying interaction** involves the user that makes a question to the **Bot** and the Bot trying to answer it.

The answer to users question is done by:

Predicting the **RELATION** to which the users question belongs. After predicting the relation the question is pos tagged and from it the noun concepts are extracted. The concepts of interest are the concepts which fall in these pos tags [**"NN", "NNP", "NNS"**]. These concepts serve as seeds in a full text search on a highly indexed Mongo DB which is populated with the most recent Knowledge Base server dump. The results are filtered by using the relation predicted earlier. Then the remaining results (the questions) are matched with the users question by computing the similarity between each filtered result and the users question. After that the answer of the question with the highest similarity coefficient with the users input question is returned to the user. Similarity is computed by following [2] and its implementation in [3]

### 3.2 Relation Prediction

Relation prediction is performed by training a a Convolutional neural network. Convolutional neural networks trained on top of pre-trained word vectors like **Googles word2vec** or **Twitters Glove** have achieved very good results, as shown in [4]. Convolutional networks were designed for computer vision but have been shown to be very effective in natural language processing tasks like sentence classification.

By following the model set up presented in [4], the model trained on the dataset of the **Knowledge Base** server performed very well on the relation prediction task by achieving an accuracy of above 93-94%

#### 3.2.1 Data preprocessing

The CNN is trained on top of Google word vectors obtained from an unsupervised neural language model. The vectors are trained by (Mikolov et al)[5] These vectors are very powerful, and have shown to be universal feature extractors'[4]. The model trained on top of these vectors performs very good with very small hyperparameter tuning.

To create the input of the neural network I take each **question** in the training set, which consists of around 60% of the dataset provided in the Knowledge Base server(due to memory limits on my machine), and split it by spaces. The each token of the sentence is converted into a word vector representation and this word vector representations are added to a list which is padded into a length of 15. This consists of the **X_train** dataset. For the **Y_train** dataset I have a list of all the 16 relations that the sentences belong to. I append to a list the index of the relation corresponding to the sentence that is being processed at that moment in time. In the end I use **keras np_utils.to_categorical** to convert all the Y_train in a one-hot encoding, meaning that, if relation PLACE has index 2 in the lists of relations then it will be represented as a list of all zeros and a 1 in the third position of that list(array indexes start at 0) .

#### 3.2.2 The Model Infrastructure

The model is composed of several layers. Initially a normal **Input** layer is added and directly after it, I have added a **Dropout** layer of 50%. Then there are 2 blocks of 3 types of layers consisting of a **Convolutional1D** layer followed by a **MaxPooling1D** and then a **Flatten** layer. The Convolutional layers have **relu** as activator and both have **strides** of 1. After the Convolutional block I have added another **Dropout** layer which is followed by a **Dense** layer with 50 hidden dimensions. In the end I add another **Dense** layer with **16 units**, which represent the number of *relations* we have, and a **softmax** activation function which will give a probability distribution over the 16 relations we want to predict. The model is trained over 10 epochs with **categorical_crossentroy** as loss function and **Adadelta** optimizer. The data is feed to model into batches of 64. Most of the hyper parameters used were suggested in [4] as they have performed extensive **Grid-Searching** in various datasets for sentence classification.

### 3.3 Querying the KBS

#### 3.3.1 Search Keywords

The Model trained above is loaded on start-up of the Knowledge Bot. The question asked by the user is prepared in the input format as the model requires and afterwards it is given to the model for prediction. The question is processed further more to get the possible concepts for which the user is asking about in that predicted relation. To get the candidate concepts I perform **POS**(part of speech)-tagging by using the **nltk** library. From those pos-tagged tokens I get the **["NN", "NNP", "NNS"]** because almost all the time the concept for which the user is asking about, falls into that part of speech category. With that list of nouns I query the local indexed copy of the Knowledge base server.

#### 3.3.2 Optimizing Search

The filtering is assumed to be done also by domain initially, but since the domain fields in the knowledge base are all empty it is currently not needed in the Quering interaction but is predicted to perform in the future when the **KBS** question answer pairs will have also the domain in it.

To perform a fast and efficient search I have inserted every record of the Knowledge base server in a local **Mongo DB**, and I have indexed it by many fields where the concept for which the user is asking for might be found. MongoDB provides text indexes to support text search queries on string content. Text indexes can include any field whose value is a string or an array of string elements[6]. I have indexed the database by **question, c1, relation**. The search is performed most of the time in less than 1 second. In this way the search is faster and the Knowledge Base server is not contacted when not needed to avoid wasting time by performing requests to it. Of course the newly added records in the Knowledge base server needs to be fetched in order to improve the model periodically, for which I have implemented a script that can be run as a **cron-job**, lets say once per day at midnight to fetch the new records in the Knowledge Base and add them to the local, indexed Mongo DB instance, by keeping track of the current state of my local db with the last id of the Knowledge base record that I have stored and query the server only for the records after that last index by using the servers REST endpoint.

### 3.4 Answer selection

The result set returned by the query performed to the Mongo BD is further filtered according to the predicted **relation** for the users question. The search space is narrowed even more in this step. The possible remaining candidates are matched with the users question by computing the **symmetric similarity**[2][3] between the users question and the questions in the filtered dataset. In the end the row with the highest **symmetric coefficient** is chosen and that answer is returned to the user.

# 4 Enriching Interaction

## 4.1 Overview

In the enriching interaction the bot attempts to gain knowledge from the user for the concepts for which there is no information in Knowledge base or there are very few triples associated to it.

The flow goes like this: I pick a random babelnet id in the chosen domain using the provided file *babeldomains_babelnet.txt*, and then get it's lemma from Babelnet. Further I check how much information I have for it in the server and by this the relation of the question to be made is decided. Finally I ask the built question to the user, get the answer and enrich the Knowledge base server.

## 4.2 Concept Selection

I have mapped all the *babelnet ids to domain* file in a dictionary of lists where the key is the domain name. With the user chosen domain, I get a random babelnet id in the list of babelnet ids for that domain and make a request to the babelnet server to get the lemma associated to that babelnet id. Since there can be more than one sense for the babelnet id, I get the first sense. After that I query the KBS(my Mongo DB instance) for that concept. If in the result set there is no record containing the concept, then it is a very good candidate for which to ask question about using any of the relations mapped to the domain, else if there is information about that concept in the server I check the relations for which there is no information and pick one of them. If there is none of above cases, thus the concept has triples for all relations, I pick another babelnet id and do the same checks to it until a suitable concept is finally chosen.

## 4.3 Question Generation

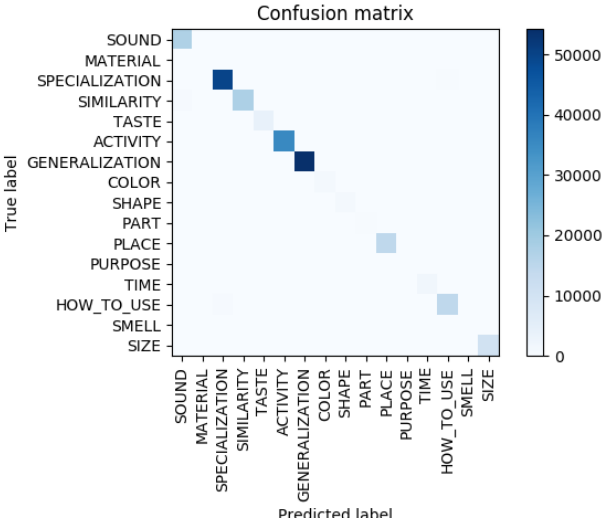
As the concept is chosen, the relation and the question pattern is chosen. If it is the case where no record containing the concept is found on the database, I pick one of the relations in the domain randomly, since each information we can gather about this certain concept would be of interest thus enriching the Knowledge base. Then I choose one question pattern among the patterns mapped to chosen relation and substitute the concept in the question pattern. As the question is built it is sent to the user and bot is waiting for the answer. In the other case where there is some record about the chosen concept in the database, I pick a relation out of unused relations of the particular domain and proceed similarly to build the question.

## 4.4 Answer Parsing

As the bot gets the answer from the user, some processing is done before adding the item to the Knowledge base. What we actually have are: question, answer, c1, domain, relation; and what we are missing is c2. To get c2 is kind of complicated because we don't have any information about the format of the user's answer, so this may lead to picking the wrong token as c2 from the answer. To overcome this, I have written some rules depending on the relation of the question. For example, when the question has the relation ACTIVITY, HOW_TO_USE or PURPOSE the bot expects the c2 to be the verb in the answer. When the relation is SOUND,TASTE, COLOR, SHAPE,SMELL or SIZE the bot expects c2 to be an adjective or a noun verb while for the MATERIAL, SPECIALIZATION, SIMILARITY, GENERALIZATION, PART and PLACE, c2 is expected to be a noun. As for TIME c2 can be a number (year, date). To write this rules the pos tags of the answer are considered and the most likely c2 is chosen. It is worth mentioning that those rules do not always work since they are general and I recall here that we have no clue how the user can answer. For example: "What do you do with a book?" and the user answers like: "You read a book". We see that the **c2** is a VERB most of the time in questions about ACTIVITY. This heuristic performs very good in properly extracting the concept for standard and regular answers.

# 5 Chat-bot Performance

## 5.1 Relation Classifier




The performance of relation classifier is quite good such as there are only a few confusions among classes. Actually due to the small number of confusions they are not visible in the heatmap shown above. But going through the results of the predicted relations, I have noticed that the most confused relations are SPECIALIZATION and GENERALIZATION. This confusion is normal due to the fact that during the information extraction task which was used to fill the Knowledge base, these two relations were quite confused and lots of question patterns used for one are also used for the other. However this confusion is not of high importance since

it does not happen often as seen in the testing of the model.

There are also a few confusions between ACTIVITY and HOW_TO_USE.

### 5.2 Querying Interaction performance

The Knowledge Bot performs well in this interaction by correctly predicting the questions relation most of the time. In cooperation with a full **indexed** search and the **similarity** matching, most of the time the user gets the information correctly in case that the information is in the database. But it is worth mentioning that the answer is highly depended on the information in the knowledge base. It has occurred that the answer returned is non-sense or even wrong, but after checking the knowledge base information for that concept it resulted that the data in the KBS are actually wrong.

For example, when asking *"Where is Rome located?"*, the answer is *"Adams County"*, which is not correct. The relation classifier and the concept extractor have both worked perfectly in this case by classifying the question in the **PLACE**-relation as it truly is, and also extracting **Rome** as the concept the user is asking about. But as mentioned above the answer instead of Italy, is Adams County in the knowledge base, while the Knowledge Bot is completely correct in the way it tackles the problem.

### 5.3 Enriching Interaction Performance

The bot performs very well in this selection since the list of concepts in the provided babel domains is large. The search in the database is straight forward and also the filtering of the relations in which the bot has the least information. The tricky part would be the users answer, especially finding the **c2**-concept. For this I tried to design parsing rules by using heuristics, pos tagging and even babelnet as explained above but we can not always control the way the user is answering a question by using the general rules.

## 6 Conclusions

In this project we had to develop a Knowledge Bot. The Bot should be able to answer user questions and also make questions to increase its own knowledge. The Bot would use as its source of knowledge a set of question answer pairs that were extracted previously. The **Quering interaction** which is composed of the user asking a question and the bot trying to find out the best possible answer regarding to users question was achieved by initially categorizing the users question into one of 16 predefined **relations**, and then extracting from the question the possible concepts that the user might be asking about. With those concepts and the predicted relation, a indexed Mongo Db instance is queried and filtered. Afterwards the filtered result set is matched with the users question by computing the similarity coefficient. In the end the users answer is the answer of the question that had the highest similarity coefficient with the users answer. This way of tackling the problem performs very good by giving all the time the best possible result, ofcourse taking into account if there is knowledge about that concept in the knowledge base. To be noted in this case is that sometimes when asking, the bot‘s answer might seem wrong but as observed the dataset in which the bot searches for an answer has some incorrect question answer pairs like for example **"Where is Rome located?"** which has the answer **"Adams County"** which is not true but as far as the querying algorithm is concerned that is a correct execution.

The **Enriching interaction** which includes asking the user for unknown information is done by selecting a concept and a relation and composing a question about it. The the users answer is taken and the **concept2** is extracted by using heuristics related to the part of speech expected for the concept in the users answer should be. Ofcourse this method of extracting **concept2** does not work all the time because there is no way to predict all the users answers but for standard answers it extracts the proper concept.

## 7 Bot

The bot is a telegram bot and can be contacted under the name **@CarbonBot**[7]

## References


[1] WILDML. Convolutional neural networks for nlp.

[2] Corley Courtney Mihalcea, Rada and Carlo Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. 2006.

[3] nlpforhackers.io. Computing sentence similarity using wordnet.

[4] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[5] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[6] Mongo DB Inc. Mongo database.

[7] Telegram. Telegram bots.