

Natural Language Processing

Final Project: The Knowledge Bot

Michele Cipriano

September 30, 2017

1 INTRODUCTION

The development of a question-answer system has always been a challenging topic in artificial intelligence starting from the early developments of computer science with the Turing test. During last years these kind of systems have seen an increasing interest from top companies, which are mainly developing personal assistants that can help users in basic tasks or answer to questions with a various range of complexity, from “What’s the weather today?” to “What’s the integral of $f(x)$?”. This possibilities are due to developments in natural language processing and to the availability of GPUs that can speed up the learning process of neural networks which are more and more used in various tasks of NLP.

The aim of this project is to develop a bot that can answer to questions and can improve its behaviour by asking new questions to the users and learning by their answers. To achieve this, a dataset containing a large amount of question-answer pairs (1.1M) has been used. The whole project has been developed in Python, using Keras and PyTorch to simplify the learning process for the models used by the bot. It can answer and ask questions communicating with a Knowledge Base Server, which is collecting all the old and the new data, and it works using Telegram API, which makes it easy to send and receive messages from whatever device. The behaviour of the bot strictly follows a workflow which will be described in depth in section 2.

The main part of the workflow is the querying phase. Here three different methods have been developed and compared. First, an ad hoc algorithm has been developed without using any learning algorithm to classify relations or to extract concepts from the sentences, this method works fine only if a certain set of patterns is being followed, otherwise it fails. To improve it, a relation classifier and a concept extractor have been developed to search for the subject and the object of the questions. This method made the bot much more flexible, allowing it to answer to a larger amount of questions, but it’s not always able to extract con-

cepts correctly. Nowadays state-of-the-art models are using variations of Seq2Seq [1]; in this project a basic end-to-end model based on Seq2Seq has been developed and studied to test how well this kind of method performs on the knowledge base used. Unfortunately, even if the network is able to understand some patterns, it's not able to answer to the questions. An in-depth analysis of this model is done in section 5.

The report starts with the description of the workflow and then analyzes in depth each step of it, from the three different methods used in the querying phase (sections 3 to 5) to the enriching phase (section 6). An important role is played by the use of BabelNet and BabelFy services to disambiguate the concepts of the knowledge base during both the querying and the enriching phase.

2 WORKFLOW

The bot manages different conversations independently. Before starting a conversation the user has to send a message to the bot, the bot then asks for the BabelNet domain and after the answer has arrived it chooses randomly to answer or to ask a question. These two tasks are called querying and enriching. In the querying phase the user asks a question which is analyzed by the bot that gives an answer. In the enriching phase the bot asks for a question and it adds the answer received by the user to the knowledge base in order to improve its future answers.

In this project three different methods have been studied for the querying phase. The first method doesn't make any use of learning algorithms, it's strongly tied to the question patterns and it uses them to extract concepts from the questions. The second method uses learning algorithms; here two different neural networks have been developed. The first one is a relation classifier, which maps a question to one of the 16 relations used in the knowledge base, while the second is a concept extractor, which maps each word of the question to one of the two concepts or any other element that could be included in the question.

In the enriching phase the bot generates a question by exploiting question patterns, a set of concepts depending on the domain chosen by the user and the relation associated to it which is chosen randomly. Once the answer has been received, the bot analyzes it and creates a tuple that will be added to the knowledge base by sending it to the knowledge base server via an HTTP request.

3 QUERYING: WITHOUT LEARNING ALGORITHMS

Probably the easiest way to answer to questions is to exploit the question patterns. In fact, if the question asked by the user has the same structure of one of the questions contained in the set of the question patterns (e.g. "Where is Rome?" has the same structure of the pattern "Where is X?") it becomes easier to extract the concepts from the questions ("Rome" in the case of the previous example). The problem is thus reduced to match the received question to one of the question patterns.

In the developed algorithm, the question patterns are first ordered depending on the Levenshtein distance between the question and each pattern, and are then compared with the

question to see if there is a match. If a match is found, the concepts are searched in the knowledge base. If they are found in a tuple, the answer of the tuple will be send to the user as the answer.

Here's a list of examples showing how this answer generator works:

User: *Is Teluk Ramunia in Johor?*

Bot: *Yes*

User: *Where is photoinitiator used?*

Bot: *It can be used to initialise radical polymerization e.g. in the preparation of acrylate polymers*

User: *What is the Colosseum?*

Bot: *I don't understand.*

Note that in the last example the bot can't answer to the question because the pattern "What is the X?" is not included in the question patterns. This is the main problem of this method.

4 QUERYING: USING LEARNING ALGORITHMS

The problem of doing the querying step exploiting question patterns, as seen previously, is that questions are strictly tied to them. Thus, if there is a typographical error in the question or, in general, if the question doesn't have the structure of any of the questions of the existing patterns the algorithm is not going to work well. Moreover the knowledge base is built to make use of relations and the method described earlier doesn't make any use of it.

Probably the first method the comes to mind when trying to improve a naïve algorithm as the previous one is to use neural network. In particular it's possible to use a neural network to classify the sentence to the relation that is used in it and a neural network to map each word of the sentence to a concept. Once the relation has been defined and the concepts have been extracted, the problem is reduced to a search algorithm. In fact, at this step, the answer can be determined by searching the relation and the concept in the knowledge base.

The relation classifier is composed of an embedding layer initialized with a vocabulary of 138K words directly created from the knowledge base, the word embeddings are initialized with Word2Vec. The embedding layer can be trained like the other layers in order to learn the embedding of the words that were not found in Word2Vec. This layer is followed by a BiLSTM of 250 hidden nodes and a Softmax layer of 16 nodes corresponding to the relations of the knowledge base. This models has been trained for 5 epochs using RMSprop reaching an accuracy of **98.781%** on the test set. Note that the accuracy is computed on a vocabulary of 138K words, which is a subset of all the words included in the knowledge base. All the missing words are represented with the special token <UNK>.

The structure of the concept extractor is similar to the structure of the relation classifier, with the exception of the output layer (a Softmax with 7 nodes) and the fact that here each word, not the whole sentence is mapped to a value. For what regarding the output layer, each word can be a token of a concept (initial token, final token or both initial and final tokens) or another word. Moreover each word can be part of concept 1 or concept 2. The idea of the network is thus to distinguish the two concept and to differentiate between the words

of the two concepts and the other words. The network could have been made of only three outputs (concept 1, concept 2, other), but on preliminary tests the network confused to map correctly words like “the” or “a”, that could easily be part of a concept or another element of the sentence, and it didn’t work well on concepts of multiple words. Before the training of the network, since the knowledge base is noisy, is necessary to remove bad tuples. Moreover some of the concepts are defined only with a BabelNet ID, thus a query to BabelNet it’s needed in order to get the correct word. Since querying BabelNet for each concept is slow, a cache containing pairs of BabelNet IDs and words has been developed in order to speed up this process. This model, as the previous one, has been trained for 5 epochs using RMSprop reaching an accuracy of **94.273%**. Notice, again, that the accuracy is computed on the same vocabulary as before.

Here’s a list of examples showing how the relation classifier and the concept extractor work:

User: *What is the Colosseum?*

Bot: *amptitheatre*

Relation: *GENERALIZATION*

Concepts: *Colosseum*

User: *Where is Dinsdale placed?*

Bot: *Darlington*

Relation: *PLACE*

Concepts: *Dinsdale*

User: *Is siege of Buda (1541) included in Kingdom of Norway?*

Bot: *no*

Relation: *PART*

Concepts: *Kingdom, None*

User: *How can I use hydroiodic acid?*

Bot: *It can be used to convert the starting material (methanol*

Relation: *HOW_TO_USE*

Concepts: *hydroiodic*

User: *Can be Knowledge used as as a resource to render services for customers?*

Bot: *Yes*

Relation: *HOW_TO_USE*

Concepts: *render, to*

As it is possible to see from the examples, even if the bot answers correctly to some of the question it’s not always because the extracted concepts are correct. In fact, while the relation classifier seems to work well in practice, the concept extractor easily makes mistakes; this happens in particular when there are two concepts in the question. This is mainly due to the fact the noise in the knowledge base, where most of the tuples are either incorrect or are defined with wrong concepts.

5 QUERYING: SEQ2SEQ

In the last few years developments on machine translation (MT), in particular neural machine translation (NMT), led to the study of a model called Seq2Seq [1], which is basically

an encoder-decoder [2]. Variations of Seq2Seq have been used not only to improve MT, but also to develop end-to-end approaches for question-answering systems [3]. This method led to the development of two important datasets called SQuAD (Stanford Question Answering Dataset) [4] and MS MARCO [5]. All top models are variations of Seq2Seq, which suggests that structure of this type are worth studying for question-answering systems.

In this project a simple Seq2Seq model has been developed and tested on the knowledge base. The whole model has been developed in PyTorch, which makes the management of the network easier than Keras thanks to dynamic computational graphs. The interesting thing about Seq2Seq is, as said before, that it's an end-to-end approach, thus it's not necessary to perform multiple steps or develop a pipeline, the network generates the answer directly from a question.

The Seq2Seq model is made of two parts: an encoder and a decoder. The aim of the encoder is to map a variable length sequence (the question) to a fixed length vector (also called context vector), while the aim of the decoder is to map a fixed length vector to a variable length sequence (the answer). Both the encoder and the decoder have an embedding layer that maps a word to its context vector. These layers are initialized with a vocabulary of 30K words and the embeddings are coming from Word2Vec as the other networks. The embedding layers are followed by 2 recurrent layers, here GRU are used instead of LSTM to make computations slightly faster without losing long term dependencies. GRU layers are made of 512 hidden units. The decoder has a Softmax layer as output in order to compute the probability of a word to be in a specific part of the sentence.

The model has been trained using RMSprop for 2 epochs. The input of the encoder and the decoder are the source and the target sentences, which are made of indices corresponding to the indices of the words in the vocabulary used. To try to increase the amount of short term dependencies the source sentences have been reversed, as done in [1]. At the end of each sentence a special token <EOS> has been appended. To speed up the training phase a set of four buckets has been used in order to reduce the amount of padding in the sentences. Moreover a large batch of 128 elements have been used to save time on tensor computations. The model has been trained for a little bit more than two days on a NVIDIA K80 using Google Compute Engine. Of course using such a large batch quickly increases the amount of memory that the GPU needs to do the computations on tensors. This affects negatively the number of hidden nodes, hidden layers, bidirectionality and the size of the vocabulary that must be smaller. Nevertheless the use of large batches was necessary because of a limited amount of credits on Google Cloud Engine.

The models reached an accuracy of **33.271%** on the test set. Consider, however, that the accuracy has been computed on a dataset created with a vocabulary of 30K words. In fact, as it is possible to see from the following examples, the bot is not able to answer to the questions:

User: *What is the Colosseum?*

Bot: Lydenburg honeycomb divert México divert clarify reconstruct TL Governments Govern-
ments Governments Governments Governments Governments Governments Govern-
ments Governments Governments Governments Governments Governments Govern-
ments Governments Governments Governments Governments Governments

User: *Is Takamatsu Airport in Takamatsu?*

Bot: *zinc Charles*

User: *Is Xinjiang a type of autonomous region?*

Bot: *Anchorage squarees Panipat Oxnard Oxnard Oxnard Anchorage ruggedes Anchorage Georgetown Governments squarees Tourists talk cancer Tourists Pretoria overcome squarees recognize Governments squarees deny deny squarees recognize Governments deny squarees*

User: *Where is Liverpool?*

Bot: *Milford cutting transmit repel repel rabbits Governments Monday repel repel microorganisms Governments Monday Governments Monday transmit repel medicine cancer repel microorganisms Governments Monday Governments Carmel repel microorganisms Governments Monday*

User: *Is Roshven in Scotland?*

Bot: *México Cubuk Charles Charles Charles Tigrinya Gazipasa transport repel refugees repel repel compress At repel magnetic Kizilcahamam modify modify*

Even if this network doesn't work it's important to notice some behaviours. The first one is that for most of the answer the network has not learned to give them a short length; in fact, the greatest part had to be cut. Most of the questions start with a place, this could be due to the fact that the relation place is the most common relation in the knowledge base and that places are often used in other relations as well (e.g. for the answer "Is Xinjiang a type of autonomous region?" "Anchorage" is municipality in Alaska).

The main reason why the network was not able to learn the answers is that the tuples of the knowledge base are noisy, most of them have, in fact, either wrong answers or a bad structure of the sentence. Moreover, due to a general unavailability of GPUs it was not possible to train the network for more than two days. This negatively affected the amount of the hidden units, which is generally larger and the size of the vocabulary used, which for this knowledge base is much bigger. This network is, by the way, a basic Seq2Seq; since the introduction of the latter a lot of progress has been made, from the introduction of an attention mechanism, which increases the accuracy of the network [6], to a better management of the vocabulary, which could help to better handle all the words of the knowledge base [7]. To conclude, these kind of networks are often trained for more than a week, also on multiple GPUs [8].

All these development makes Seq2Seq worth studying, in particular because it's an end-to-end approach and it's easy to use once the training has finished. Of course, a well-finished dataset could easily help the network to reach higher accuracy faster, in particular if a teacher forcing method is used for the decoder input.

6 ENRICHING

In the enriching phase the bot generates a question choosing the relation and the concepts randomly depending on the domain chosen by the user. Once the answer has been received the bot analyzes it and it sends a tuple to the knowledge base server in order to improve its behaviour in the future.

Each BabelNet domain is associated with a set of relations, each question pattern is associated to a relation and each concept is associated to a BabelNet domain. With all of this it's possible to generate a question with ease. The developed algorithm chooses a relation and a concept randomly depending on the BabelNet domain chosen by the user at the beginning

of the conversation. Then a question is chosen randomly from a set of question patterns depending on the relation of the previous step. Each question can include one or two concepts in its structure, i.e. the questions can be of the form “Where is X?” or “Is X in Y?”. In the latter case another concept needs to be extracted from a set of concepts (again, depending on the BabelNet domain) in order to complete the generation of the question. At this point the algorithm substitutes “X” and “Y” with the corresponding concepts creating the question, which is sent to the user.

Once the user has answered to the question, if the latter contains only one concept, another concept needs to be extracted from the answer in order to send the correct tuple to the server. Here, a neural network, different from the one used in 4, is used to extract the concept. This neural network maps each word of the sentence to a concept or another element. In particular each word can be the first word of the concept, the last word of the concept, the concept itself or any another kind of word. Before sending the tuple to the server, the service BabelyFy is used to disambiguate the just found concept in order to send the correct BabelNet ID. Of course this last step is not needed if the question already contains two concepts.

Before training the network, each tuple of the knowledge base is preprocessed in order to remove malformed tuples and to convert BabelNet IDs to words. Here a cache of BabelNet IDs is used in order to reduce to amount of requests to the server and speed up the preprocessing phase.

The neural network has been developed using Keras. It is composed by an embedding layer (initialized with an embedding matrix created from a vocabulary of 138K words directly extracted from the knowledge base and Word2Vec), a BiLSTM of 250 hidden cells and a Softmax layer with four outputs corresponding to the classes described above. The optimizer used is RMSprop and the network has been trained for 5 epochs reaching an accuracy of **99.104%** on the test set. Note, again, that the accuracy is computed on a vocabulary of 138K words.

Here’s a list of examples showing how the question generator and the concept extractor work:

Bot: *What shape does Marasmius have?*

User: *It has the shape of a mushroom*

Concept extracted: *mushroom::bn:00056435n*

Bot: *On which date was Cobwebs to Catch Flies held?*

User: *It was not held, it was written in 1783*

Concept extracted: *1783*

Bot: *What is Istanbul Samandira Army Air Base?*

User: *It’s a military base*

Concept extracted: *military base*

7 CONCLUSION

Developing a good question-answering system is a hard task, in particular when the dataset is noisy and there is no availability of GPUs. This project compares different approaches to solve this problem. Developing naïve algorithms is easy, but comes at a price: the amount of questions that it supports is limited. Methods based on neural networks are somehow able to

generalize the problem but doesn't work well if the data is noisy. The papers regarding state-of-the-art methods on SQuAD and MS MARCO make Seq2Seq approaches both promising and interesting to study, but when the amount of data becomes huge it becomes very hard to handle all the information with a single network, making pipelines still a better choice for the development of good question-answering systems.

APPENDIX: IMPLEMENTATION

The whole project has been developed in Python, using Keras built on top of Tensorflow and PyTorch. All the models have been trained splitting the data into a training set (60%), a validation set (20%) and a test set (20%) after the preprocessing. Here's a list of the files included in the project with a brief description of each one:

- **AnswerGenerator.py**: this file contains the implementation of the algorithm described in section 3.
- **BabelNetCache.py**: a class that improves the performances of the bot by reducing the amount of requests to BabelNet, this is possible because of a cache file that stores pairs of BabelNet IDs and concepts.
- **bot.py**: this is the main file of the project, it implements the workflow and uses all the models generated during the training phase, managing the communication with the Knowledge Base Server and the communication with the users using Telegram APIs.
- **download_kb.py**: this script downloads the whole knowledge base from the knowledge base server saving all the data on a single JSON file.
- **KnowledgeBase.py**: an auxiliary class used to load the knowledge base from a JSON file and to search a tuple containing a certain relation and the two concepts of the tuple itself, this last step is particularly useful to generate the answer once the relation and the concepts have been extracted from the question asked by the user.
- **patterns.py**: this script collects all the pattern files into a single file removing bad patterns, it is supposed to be used after having removed malformed files and a double check by hand of the final file is needed to have a correct set of patterns.
- **query_kb.py**: this script could be used to test the bot by randomly selecting a tuple from the knowledge base or by searching a sentence between the questions and the answers of it.
- **QuestionGenerator.py**: the algorithm described in section 6, where questions are generated during the enriching phase.
- **QuestionPatterns.py**: an auxiliary class that manages the question patterns storing them by relation.

- **seq2seq.Seq2Seq.py**: the neural network (described in section 5) which implements Seq2Seq using PyTorch.
- **seq2seq.utils.py**: a set of auxiliary functions used to train and evaluate Seq2Seq network.
- **train.py**: this file is responsible for the training of all the models used by the bot, it pre-processes the knowledge base for each model by removing useless tuples and by creating the input and the outputs used by the neural networks. Both Keras and PyTorch are used here.
- **utils.py**: a set of functions used to communicate with BabelNet and BabelFy and process outputs and inputs of the neural networks.
- **vocabulary_from_kb.py**: an auxiliary script used to build a vocabulary depending on the frequency of the words found in the questions and the answers of the knowledge base.
- **Vocabulary.py**: an auxiliary class used to manage a vocabulary file plus special symbols like <PAD>, <UNK>, <GO>, <EOS>.
- **Word2Vec.py**: an auxiliary class which is used to create embedding matrices exploiting the Word2Vec model made available by Google [9].

REFERENCES

- [1] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to Sequence Learning with Neural Networks,” in *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NIPS’14, (Cambridge, MA, USA), pp. 3104–3112, MIT Press, 2014.
- [2] K. Cho, B. van Merriënboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1724–1734, Association for Computational Linguistics, Oct. 2014.
- [3] O. Vinyals and Q. V. Le, “A Neural Conversational Model.,” *CoRR*, vol. abs/1506.05869, 2015.
- [4] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, “SQuAD: 100,000+ Questions for Machine Comprehension of Text,” in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Association for Computational Linguistics, Oct. 2016.
- [5] T. Nguyen, M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, and L. Deng, “MS MARCO: A Human Generated MACHine Reading COMprehension Dataset,” in *Proceedings of the Workshop on Cognitive Computation: Integrating neural and symbolic approaches 2016 co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016)*, Barcelona, Spain, December 9, 2016., 2016.
- [6] M.-T. Luong, H. Pham, and C. D. Manning, “Effective Approaches to Attention-based Neural Machine Translation.,” *CoRR*, vol. abs/1508.04025, 2015.
- [7] S. Jean, K. Cho, R. Memisevic, and Y. Bengio, “On Using Very Large Target Vocabulary for Neural Machine Translation,” in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, (Beijing, China), pp. 1–10, Association for Computational Linguistics, July 2015.
- [8] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, “Google’s neural machine translation system: Bridging the gap between human and machine translation,” *CoRR*, vol. abs/1609.08144, 2016.
- [9] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” *CoRR*, vol. abs/1301.3781, 2013.