INFO8010: Deep Learning based Chatbot

Julien De Cooman[1] and **Heddari Wafaa**[2]

[1] *julien.decooman@student.uliege.be (s113628)*

[2] *wafaa.heddari@student.uliege.be (s195807)*

I. INTRODUCTION

A Chatbot, that can also be referred as a conversational agent, is a software, based on Artificial Intelligence (AI) and Deep Learning techniques, able to simulate a conversation, to communicate and interact with human through text or even audio messages. This conversational agent is driven by Natural Language Processing (NLP) and Machine Learning (ML) to process an input data and be trained and prepared to deliver the appropriate answer to the user.

While creating a chatbot, two main options can be considered depending on the objectives. The first one is to design a Task-Oriented or simple chatbot. In this case, the chatbot is based on pre-written keywords and can handle common questions and simple interactions that do not require a variety of variables. It can be used to help users for searching specific information such as reservation process, hotel booking, etc. The second option concerns a more elaborated type of chatbot called Data-driven or Predictive chatbot which is able to learn, thanks to a model, to be more interactive and to deliver more specific answers than the previous type. Apple's Siri and and Amazon's Alexa are two famous examples of this type of chatbots.

In this project, we will be obviously focused on the second type of chatbot. We will use the open source "*Cornell Movie-Dialogs Corpus*" dataset, containing a collection of fictional conversations extracted from raw movies to feed and train a sequence-to-sequence model with the objective to deliver at the end a chatbot able to conduct a more or less consistent conversation.

II. RELATED PROJECT REVIEW

In the paper "A New Chatbot for Customer Service on Social Media"[1], the five authors *Anbang Xu*, *Zhe Liu*, *Yufan Guo*, *Vibha Sinha* and *Rama Akkiraju* explain how they attended to adress the challenges raised by the new trend of users to use social media to request and receive customer services. Indeed, by developping a new conversational system based on state-of-the-art deep learning techniques, their hope was to increase customer satisfaction and reduce the time and ressources involved by companies.

The first step of their work was the collection of data. For this purpose, 62 brands were selected, based on certain criteria, and about one million of conversion data linked to these companies were collected. Among them, 30K were used for evaluation, and the rest for the whole system development.

To create this system, several steps were performed.

- First, they cleaned the data by removing non-english requests, requests with images and @mentions.
- Second, they tokenized the data and built a vocabulary of the most frequent 100K words in the conversations.
- Third, they generated words-embedding features by using the word2vec neural network language model trained on the collected corpus. Each word in the vocabulary being then represented as a 640-dimension vector.
- Four, they trained the different networks. Indeed, the core of the system consists of two LSTM neural networks: one as an encoder that maps a variable-length input sequence to a fixed-length vector, and the other as a decoder that maps the vector to a variable-length output sequence. The input and output of LSTMs are vector representations of word sequences, with one word encoded or decoded at a time. Both LSTMs are trained jointly with 5 layers x 640 memory cells using stochastic gradient descent and gradient clipping.

Concerning the evaluation, the authors first performed a content analysis and discovered the presence of two types of request. The emotional requests, where users intend to express their emotions or opinions toward a brand without seeking specific solutions, and informational requests, which are sent by users to get information that may help them to solve their problems.

Then for each of these types of request, the system was first evaluate by humans according to three measures: appropriateness, empathy and helpfulness, and compared with actual human agent as well as standard information retrieval baseline (IR). The results showed that the Deep Learning system has a similar ability as actual agents to show empathy toward users in emotional situations. In addition to that, this one also outperformed IR in all three aspects of rating, especially when it comes to emotional requests. It has also been noticed that the performance of both DL system and IR dropped significantly when requests became informational, and that human agent performed better than others and equally well on different requests.

Finally, the researchers also used BLEU metric applied to 30K user requests to evaluate the DL system compared to IR. Again, the first one performed significantly better.

III. THEORETICAL BACKGROUND

A. Deep Learning

For the beginners, we can define Deep learning (DL) as a subset of machine learning where artificial neural networks, algorithms inspired by the human brain, learn from large amounts of data. These artificial networks are composed of multiple processing layers of neurons which receive and process information for the following layer.

The increase of data and the stronger computing power allowed capabilities of deep learning to grown in recent years bringing some significant progress in the field of, for instance, sound and image processing with facial recognition, text classification, etc.

Among the different type of neural network architectures, we can cite the Multilayer Perceptrons (the oldest architecture), the Convolutional Neural Networks, or the Recurrent Neural Networks which will be discussed deeper in the next point.

B. Recurrent Neural Network (RNN)

A Recurrent Neural Network (RNN) is a type of artificial neural network that can take as input a variable length sequence $x = (x_1, ..., x_n)$ and produce a sequence of hidden states $h = (h_1, ..., h_n)$ allowing the information of the previous time step to be also used as input for the next computation. It is widely used in the fields of natural language processing and speech recognition.

More formaly, RNNs (fig 1) process a sequence of input data $x \in S(\mathbb{R}^p)$ with variable length $T(x)$ and maintain a recurrent state h_t, that contains information about the history of all the past elements of the sequence, and that will be updated at each time step t by:

$$h_t = \phi(x_t, h_{t-1}; \theta) \tag{1}$$

where function $\phi : \mathbb{R}^p \times \mathbb{R}^q \longrightarrow \mathbb{R}^q$ (neural network parametrized by some parameters θ) and $h_0 \in \mathbb{R}^q$.

To be able to predict at each time step t the output from the recurrent state :

$$y_t = \psi(h_t; \theta) \tag{2}$$

where function $\psi : \mathbb{R}^q \longrightarrow \mathbb{R}^C$

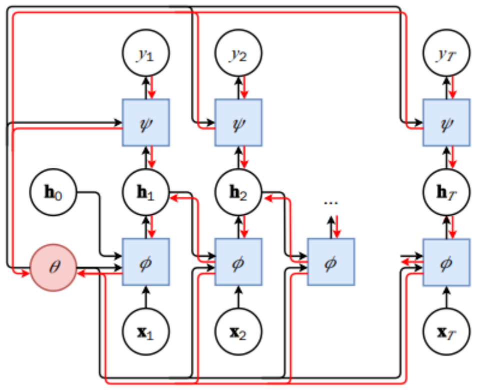


FIG. 1. Recurrent neural network, and the computation of the output y_t at each time step t

Unfortunately, because it suffers from vanishing gradient problems, RNN become hard to train and are most of the time replaced by Long-Short Tem Memory (LSTM) or Gated Recurrent Unit (GRU).

C. Sequence To Sequence (Seq2Seq)

The Sequence To Sequence model, also named Seq2Seq model, is an approach that uses Recurrent Neural Network techniques (more often LSTM or GRU to avoid vanishing gradient problem) to map an input and an output sequence of different lengths. This model is commonly used for language translation, speech translation, answering questions (like Chatbots) and many other applications linked to NLP.

As it can be seen in the fig 2 below, the seq2seq architecture is composed of three main parts: an encoder, an intermediate vector (or encoder vector) and a decoder.

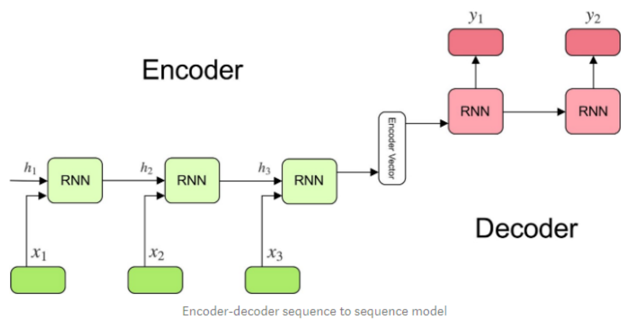


FIG. 2. Encoder-Decoder architecture of Sequence to Sequence model

The encoder part is composed of several RNN units that accept only one element of the input sequence (that is, a word x_t from the question), collects information and propagates it forward through the corresponding hidden state (h_t). At the end, the encoder produces a final hidden state, called intermediate vector, that gathers the information of all the input elements, and that will act as the initial hidden state of the decoding part. Finally, the decoder is also composed of a series of RNN units. Each of them, will accept an hidden state from the previous unit and produce its own hidden state as well as an output (y_t) corresponding to a word from the answer [2].

To better understand our implementation (described in section IV), the following theoretical points, inherent to the functioning of the seq2seq model, will be explained:

- Word embedding
- LSTM
- Drop out
- Bidirectional RNN
- One-hot encoding

1. *Word Embedding*

Word embedding is a technique of word representation as real-valued vectors in a predefined vector space that allows words with a similar meaning to have a similar representation. It is commonly used in Natural Language Processing (NLP) tasks thanks to the contextual similarity and the dimensional reduction it brings. The most famous word embedding models are the *Word2Vec* developed by google in 2013 and the Stanford's *Glove*.

In one hand, The Word2Vec consists in a simple feed forward neural network with one hidden layer that can use two different learning models to learn the word embedding: the "Continuous Bag of Words" (CBOW) or "Skip-Gram" Models. As illustrated in the fig 3, while the first approach learns the embedding by predicting the current word based on its context, the latter learns by predicting the surrounding words given a current word [3].

On the other hand, the basic idea behind the Glove word embedding is to derive the relationship between words from Global Statistics. First, a co-occurrence matrix, representing how frequently words are seen in some context, is built. Then, the matrix is factorise to obtain a lower-dimensional matrix where each rows correspond now to a vector representation for one word [4].

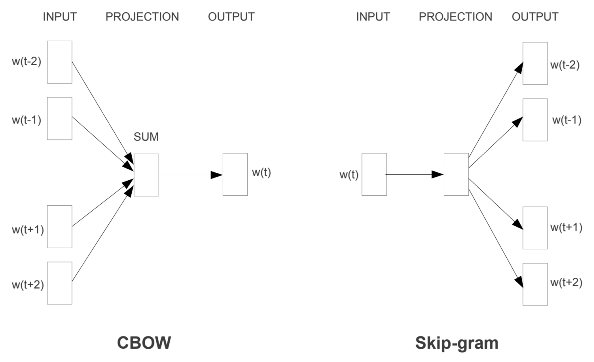


FIG. 3. Word2Vec training models

2. *Long Short Term Memory (LSTM)*

This kind of RNN network, whose cell structure is depicted in fig 4 below, has originally been designed to solve the vanishing gradient problem. It is composed of a memory cell (C_t) to carry information throughout the processing of the sequence, a forget gate (h_t) to decide which information has to be erase, an input modulation gate ($\tilde{C}_t$) to suggest new information, an input gate (i_t) to decide which part of the new information is important to keep and finally, an output gate (o_t) to determine the next hidden state [5].

The functioning in the LSTM cell is the following: at each time step, first, the previous hidden state (h_{t-1}) and the current input (x_t) are passed through different activation functions (*sigmoid* and *tanh*) to create the gates vectors. After that, the memory Cell (C_t) is successively multiplied by the forget vector ($f_t \in [0, 1]$) and add to, the product of the input vector ($i_t \in [0, 1]$) and the input modulation vector ($\tilde{C}_t \in [-1, 1]$). Finally the newly generated cell state is either used by the next LSTM cell or multiplied by the ouptut vector ($o_t \in [0, 1]$) to decide what information will be used by the next hidden state (h_t).

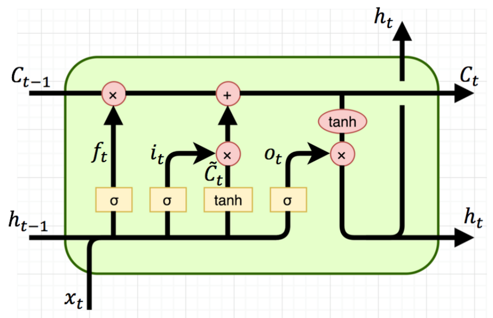


FIG. 4. LSTM structure [5]

More precisely, here is the formal definition of all the components of LSTM [5] :

$$
\begin{aligned}
i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \\
f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f) \\
o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\
\tilde{C}_t &= tanh(W_C[h_{t-1}, x_t] + b_C) \\
h_t &= o_t tanh(C_t) \\
C_t &= f_t.C_{t-1} + i_t.\tilde{C}_t
\end{aligned}
\tag{3}
$$

where W and b are learned weights and biases.

3. *Drop Out*

Dropout is a technique used to prevent overfitting by randomly ignoring some units (i.e neurons) of the neural network during the training phase. "This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass." [6]. More technically, at each training stage, individual nodes are dropped out of the network with a probability p, creating a reduced network as shown in figure 5. By doing that, the network becomes less sensitive to the specific weights of neurons which results in a network less likely to overfit the training data and more qualified to have a better generalisation.

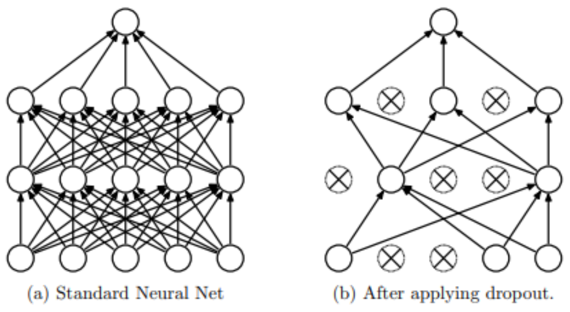


FIG. 5. Dropout Neural Net Model : (a) Standard Neural network with 2 hidden layers, (b) Thinned network after applying a dropout [6]

4. *Bidirectional RNN*

Bidirectional recurrent neural network (BRNN) is a type of RNN that connects two hidden layers running in opposite directions to the same single output. As it can be seen on fig 6, in addition to the common input X, each of them will respectively receive information from past or future states.

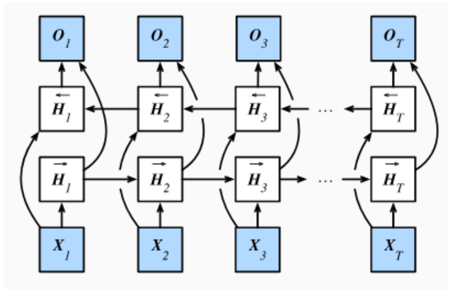


FIG. 6. Architecture of Bidirectional Recurrent Neural Network (BRNN)

Compared to RNN, Bidirectional RNN are trained to predict both the positive and the negative directions of time (forward and backward states) simultaneously. By employing two time directions, input data from the past and the future of the current time frame can be used to calculate the same output [7].

5. *One-Hot Encoding*

In the case of categorical data, categories have to be label/integer encoded to be read by neural networks. However, this type of encoding which consists of giving a unique number to each categories in the dataset suffers from the interpretation of a natural ordership by the model, which may result in poor performance or unexpected results (predictions halfway between categories) [8].

To solve this problem, One-Hot encoding can be applied to the integer representation. If so, each categories are now represented as a vector with a length equal to the number of categories in the dataset. This vector is filled by 0 at each positions except for the i^{th}, corresponding to the i^{th} category, that is set to 1 [9].

As an example, if we have three words "hello", "you" and "what", these one can be integer represented by the values 1, 2, 3, as shown in the table below.

Words	Categorical #
Hello	1
You	2
What	3

TABLE I. Label Encoding

And then be one-hot encoded into a binary vector.

	1	2	3
Hello	1	0	0
You	0	1	0
What	0	0	1

TABLE II. One-Hot encoding

IV. METHOD

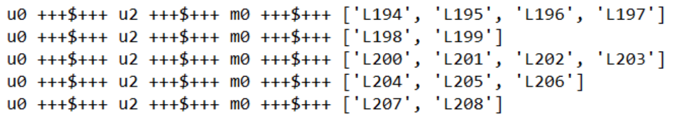
In this part of the report, the whole process behind the creation of the chatbot will be described. Basically, before starting the design of the model, the data was prepared, in a process called preprocessing, to allow it to be fed to the neural network. Then, the sequence-to-sequence model was implemented using Keras, an open-source neural-network library. Finally, the hyperparameters were set and the model trained.

A. First step: Preprocessing of data

As in our all machine learning project, the first fundamental step that takes place before everything else is the data preprocessing. This consists of preparing, transforming the data to be easily and efficiently used by a model.

In this case, the open source "*Cornell Movie-Dialogs Corpus*" database composed of two main files, *movie_conversations.txt* and *movie_lines.txt*, was used to train the chatbot.

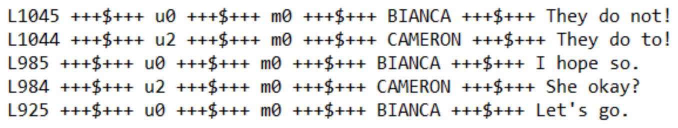
While the first file takes the form:



```
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L194', 'L195', 'L196', 'L197']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L198', 'L199']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L200', 'L201', 'L202', 'L203']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L204', 'L205', 'L206']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L207', 'L208']
```

FIG. 7. Conversation txt file overview

where $u0$ and $u2$ are the id of the movie characters implied in the conversation, $m0$ the movie id, and the elements in brackets, the sentence id's composing the conversation, the *movie_lines* text file gathers the sentence id, the character id, the movie id, the character's name and finally the sentence itself.



```
L1045 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ They do not!
L1044 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ They do to!
L985 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ I hope so.
L984 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ She okay?
L925 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ Let's go.
```

FIG. 8. Lines txt file overview

After importing these two files, both have been formatted to only keep the relevant information, that is the conversation in brackets, the sentences and the sentence's ids. Following this, all lines and their reply have been link to each other thanks to the conversation arrays, and two new datasets, containing respectively the questions and their corresponding answers, have been created.

After the creation of the new "questions" and "answers" datasets, the preprocessing goes through this process:

1. All lines were transformed to lower cases and cleaned from all possible useless punctuation's (e.g. ()+=-[]). In addition, the abbreviations were also replaced by their full form (e.g. *bout* → *about* or *wasn't* → *was not*) and extra spaces were removed.

2. A filter based on sentence's length has been applied to exclude all too long questions or answers, and facilitate the learning while limiting the memory resources. Note that the exclusion of a question automatically excludes the answer related to it, and inversely. The maximum length threshold was arbitrarily set at 15 words.

3. Dataset was shortened to approximately 50.000 pairs of question due to memory requirements and training time consumption.

4. Special tokens BOS (*Begin Of Sentence*) and EOS (*End Of Sentence*) were added respectively at the beginning and end of each answers. These tokens will be used to specify the start and the end of the sequences to the model.

5. Using the Tokenizer class from Keras, a vocabulary of the x most frequent words were created. Words not included in it were replaced by the special token UNK (*Unknown*). The value of x was determined by looking at the number of occurrences of each specific word in the dataset and by computing the percentage of words removed if specific words below a certain arbitrary threshold of occurrence were excluded. To avoid biases statistics, special tokens BOS and EOS were not taken into account.

6. Questions and answers were transformed into sequences of integer using the Keras function *text_to_sequence()*. For instance, the sentence "*did you see him*" became "[31, 4, 71, 52]".

7. To insure batches include sequences with the same length, special token PAD was added to each questions and answers to expand them up to the size of the longest sequence found in the datasets. To take a concrete example, if the maximum length is 5, a sequence such as "*fine thank you*" will be converted into "*fine thank you pad pad*". These newly formatted questions and answers correspond to the encoder and decoder inputs.

8. Decoder targets were created by removing the BOS token from the answers, by applying padding and by converting them into a one-hot vector (III C 5).

B. Second step: Seq2Seq Model

As mentioned in the theoretical background, sequence-to-sequence models are composed of three mains parts: an encoder, an intermediate vector and a decoder. In this section, it will thus be explained a bit deeper how each parts of the model are link to each others, their functioning as well as how the seq2seq model of this project has been implemented.

Note that its development was based on the interesting article ”A ten-minute introduction to sequence-to-sequence learning in Keras” from Keras blog which gives a good overview of how to use Keras to build a letter-level model [10].

1. Encoder

The encoder, whose goal is to encode input sequences into state vectors, first begin with an embedding layer (see III C 1 for more explanation), initialised with random weights, that will learn an embedding for all words in the training set.

Once feed with the encoder inputs, it produces dense vectors that will be given as input to the LSTM cells (described in section III C 2) belonging to a first layer. Each of them will in turn computes hidden and cell states ((h_i, c_i) whith h_0, c_0 initialised at zero) that will be transmitted as input to the next LSTM cell in term of time step but also to the direct LSTM cell in a second and last layer.

At the end, each layers produce final hidden and cell states forming the intermediate vector z^i.

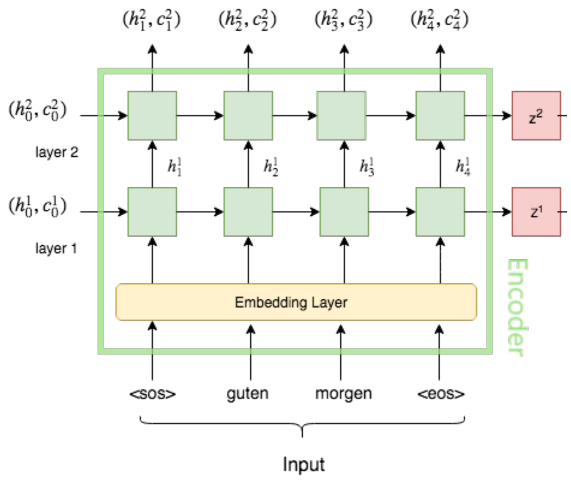


FIG. 9. Seq2Seq - Encoder

2. Decoder

For the decoder, the architecture is quite the same as the one of the encoder. It also consists of an embedding layer to transform sequences of integers into dense vectors, followed by two stacked LSTM layers producing the different states. The main differences compared to the encoder part come first from the use of the intermediate vector as the initial states for the decoder, but also from the addition of an output layer with Softmax activation to produce the probabilities of generating the different words.

The Softmax function which turns a vector of K values into a new one with K values ranged between 0 and 1 that sum to 1 (so that they can be interpreted as probabilities) can be computed as:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{4}$$

Where z_i is an element of the input vector that can take any real value [11].

Note that the functioning of the decoder will be different in the case of a training or inferring situation. Indeed, While in the latter, the output of each time step is given as input in the next one, in the training mode, the inputs of the decoder are the target sequences from the training set.

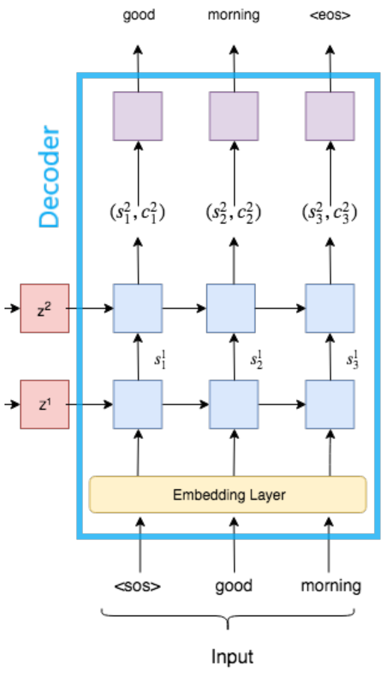


FIG. 10. Seq2Seq - Decoder

V. TRAINING & RESULTS

For the first milestone of the training process, it has been decided to train the model for 150 epochs with batches of size 32 and a splitting of the dataset between training and validation with a ratio of 0.2. Inside the network, the embedding dimension was set to 200, the number of unit in each LSTM to 256 and and a dropout of 0.3 (explained in section III C 3) was applied to each layers.

Concerning the optimization, RMSprop and Adam optimzer were tested, both with initial learning rate equals to 10^{-2} and in combination with the categorical cross entropy loss function which can be computed as:

$$Loss = -\sum_{i=1}^{M} y_i \cdot \log(p_i) \tag{5}$$

Where M denotes the number of classes, p_i the probability of the i^{th} category in the model output and y_i the corresponding target value (either 0 or 1).

The graph below (fig 11) represents the evolution of the training/validations loss and accuracy curves for both optimizers for the 150 epochs. Clearly, the Adam optimiser seemed to outperform RMSprop with the training loss decreasing faster and overall lower validation loss. Regarding this results and despite the clear overfitting present for both model, the Adam optimiser was prefered.

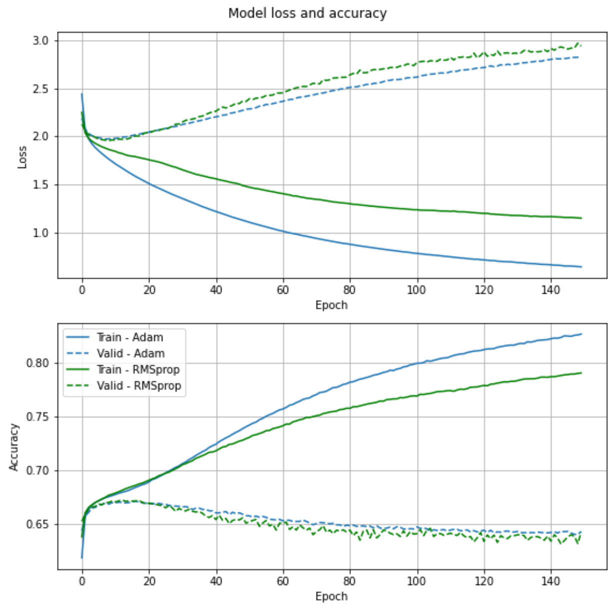


FIG. 11. Loss and accuracy evolution - RMSprop vs Adam

The second milestone of the training process concerned the use or not of a pre-trained embedding to potentially boost the performance of the model. To explore this, three new models were trained. One more with non pre-trained embedding but in a 100-dimensional version, and two with pre-trained Glove embedding of size 100 and 200.

In this case, looking at the learning curves (fig 12), conclusions were a bit more difficult to draw. Indeed, both 100-dimensional embedding had the same trend for training and validation loss, and although, in the case of the two 200-dimensional implementation, pre-trained Glove embedding had lower validation loss with practically identical training curve, it seemed not sufficient to conclude anything.

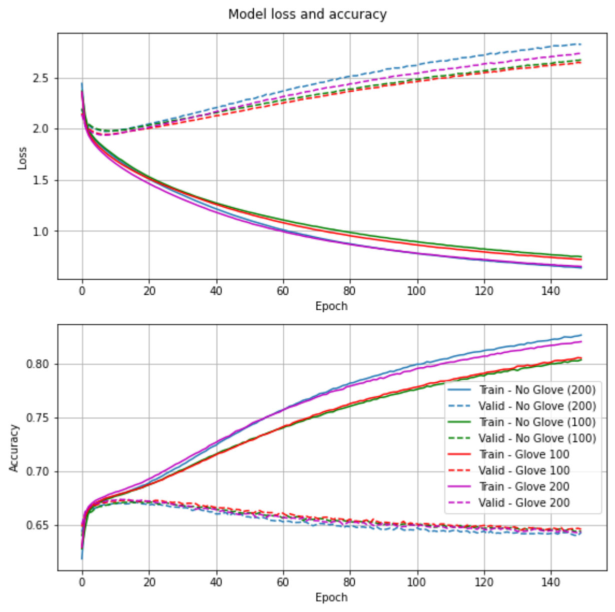


FIG. 12. Loss and accuracy evolution - Non-trained vs Pre-trained embeddings

Therefore, a qualitative evaluation of all four implementations was performed by asking some generic questions and comparing answers from all chatbot versions (see table III). Regarding the results, the pre-trained "Glove 200" embedding appeared to produce logical answers with more consistency and was definitively chosen for further experimentation's.

To try to improve again the performance of the chatbot, an increase of the number of layers to 3 and the transformation of the LSTM to bidirectional LSTM (described in section III C 4) have been experimented. After some testing with hyperparameters tuning (different dropout, learning rate decay and hidden dimension), no improvements were noticed and it has thus been decided to keep the previous version. *Of course, to prove that these modifications have been well tried, a second version of the chatbot with 3 layers and Bidirectional LSTM's will be added to the project for the sake of transparency.*

Following this, a number of attempt have also been made to reduce the overfitting of the model trained with Glove 200. Nevertheless, in spite of the efforts, the validation loss continued to increase quite rapidly during training (fig 13). One of the reason is probably the ambiguities existing in the "*Cornell Movie-Dialogs Corpus*". Indeed, this dataset contains movie dialogues with sometimes complex conversations with specific sense. Furthermore, one question can have very different suitable replies from the simple "Okay" to the long explanation, making the task more difficult for the chatbot.

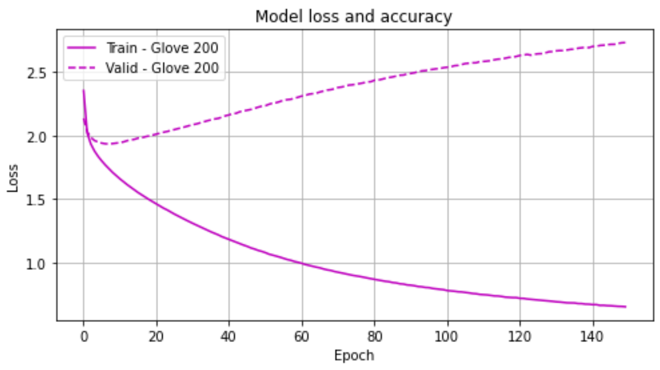


FIG. 13. Loss evolution - Glove 200 model

However, the model created in this project is one all the same able to conduct a more or less consistent conversation as it can be seen in the discussion example below:

HUMAN : hi

CHATBOT : hi

HUMAN : how are you ?

CHATBOT : head still secure to the neck

HUMAN : can i ask you a question ?

CHATBOT : uhhuh

HUMAN : are you real ?

CHATBOT : yes i am

HUMAN : are you sure ?

CHATBOT : pretty sure

HUMAN : then do you want to marry me ?

CHATBOT : what if i do become unk

HUMAN : don't worry ?

CHATBOT : why not

HUMAN : i love you

CHATBOT : i love you too

Of course, this conversation is a pretty good example and the chatbot is not always able to produce good results, sometimes giving totally incoherent answers:

HUMAN : hi

CHATBOT : hi

HUMAN : how are you doing ?

CHATBOT : fine just fine

HUMAN : have we met before ?

CHATBOT : of course you should just move out

HUMAN : no

CHATBOT : you have to add some water

HUMAN : why

CHATBOT : beacause you are dangerous

VI. DISCUSSION

In this project, the preprocessing of the data from the "*Cornell Movie-Dialogs Corpus*", the implementation of a seq2seq model with deep LSTM networks as well as the training and experiments conducted to produce a performing chatbot have been described. Moreover, as explained in the results, the goal of the project consisting of creating a chatbot able to hold a nearly human conversation have been achieved.

However, this project is far from perfection and multiple improvement can still be made. For instance, the next steps could be the implementation of Attention Mechanism to face difficulties encountered by seq2seq models when decoding long sentences, or the use of Bucketing to speed up the training. An other interesting experiment would also be to train the same model on a different dataset with more formal questions-answers pairs to have a more global idea of its performance.

VII. CODE STRUCTURE

The code is divided into three main parts:

- **hyperparameters.py**: contains all the hyperparameters to be tuned for training.
- **preprocessing_functions.py**: Contains all the functions used for the preprocessing of the data.
- **chatbot.py**: main code of the project that can be launched in training or talking mode.

To launch the project, use the command "***python chatbot.py***" with either the arguments "***-train***" or "***-talk***" depending on your need.

VIII. APPENDIX

	Non pre-trained (100)	Non pre-trained (200)	Glove 100	Glove 200
Hi	hi	hi	hi	hi
How are you doing ?	fine just fine	fine just fine i am not going to be back	fine i am fine	fine just fine
What is your name ?	jim unk	unk smith	unk smith	smith
Where are you from ?	unk unk unk	oklahoma city oklahoma	california san francisco	south america
How old are you ?	forty	$< blank >$	forty	forty
Are you a man/woman ?	i am not sure/yes i am	yes/if i asked you	no/no	no/no i told you
Are you a robot ?	i am a unk	yeah	no i am not positive	yes
What do you like to talk about ?	i am not sure	unk unk	you are very unhappy i am making a mistake marrying christine	nothing
How much is two plus two ?	biting you	i do not know	three months	fourteen years
Do you know my name	unk	no i am not	yes	no
Goodbye	yeah me and jake walk out	oh boy you are swell	call me later	call me later

TABLE III. Qualitative comparison of embedding

[1] Anbang Xu, Zhe Liu, Yufan Guo, Vibha Sinha, and Rama Akkiraju. ”a new chatbot for customer service on social media expression analysis.”, 2018.
[2] DeepAI. Bidirectional recurrent neural networks.
[3] Greg Corrado Jeffrey Dean Tomas Mikolov, Kai Chen. Efficient estimation of word representations in vector space, 2013.
[4] MLNerds.
[5] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network, 2018.
[6] Alex Krizhevsky Ilya Sutskever Ruslan Salakhutdinov Nitish Srivastava, Geoffrey Hinton. Dropout: A simple way to prevent neural networks from overfitting, 2014.
[7] Simeon Kostadinov. Understanding encoder-decoder sequence to sequence model, February 2019.
[8] Jason Brownlee. Why one-hot encode data in machine learning?, June 2020.
[9] Deep AI. One hot enconding.
[10] Francois Chollet. A ten-minute introduction to sequence-to-sequence learning in keras, September 2017.
[11] Thomas Wood. Softmax function.