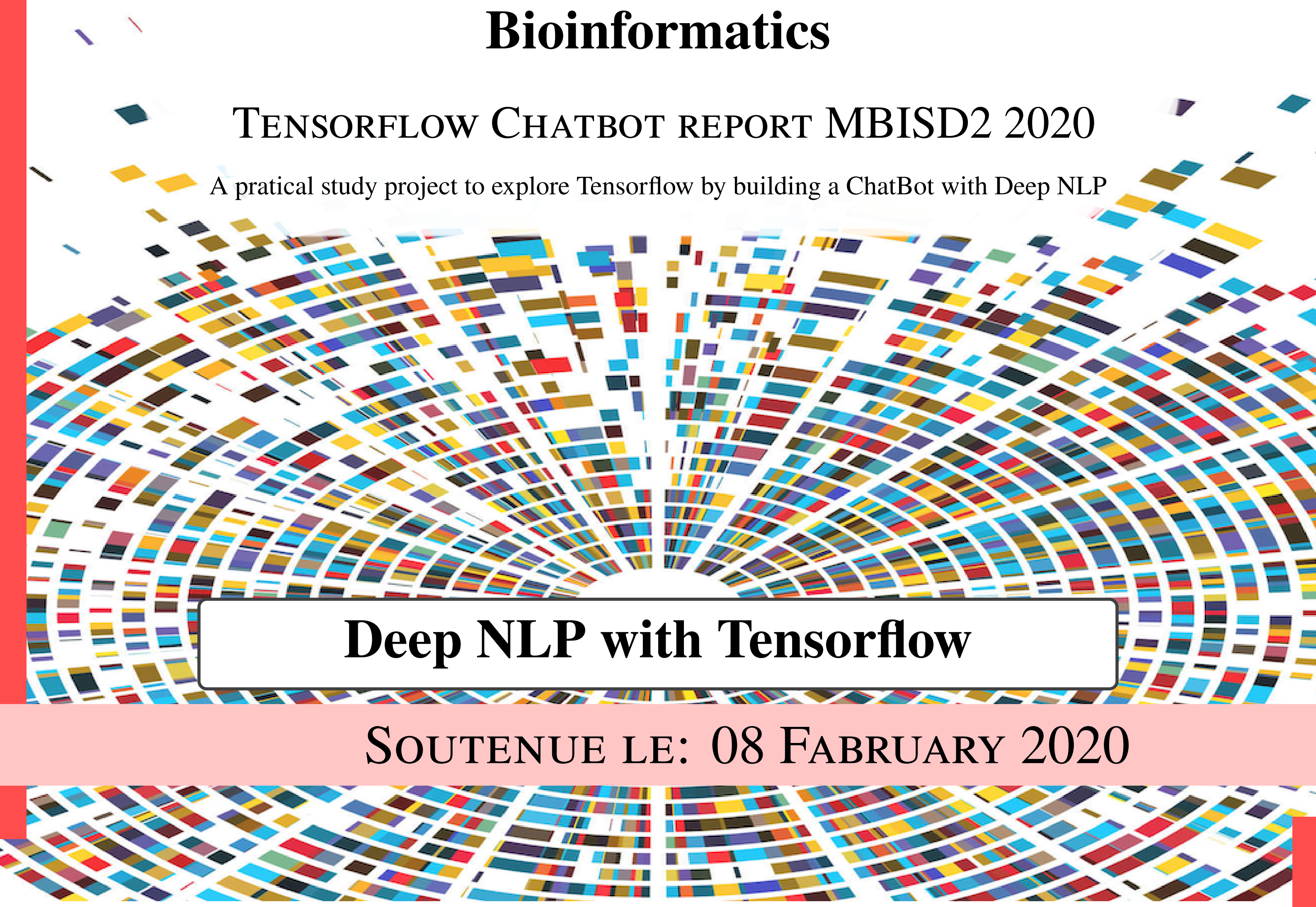
Master of Data Science and Bioinformatics

Tensorflow Chatbot report MBISD2 2020

A pratical study project to explore Tensorflow by building a ChatBot with Deep NLP



Deep NLP with Tensorflow

Soutenue le: 08 Fabruary 2020

Realised By:

BEN DAOUIA Ahmed

BERHILA Youness

ELCORBANI Brahim

Supervised By **Pr. BADIR Hassan**

Contents

List of Figures

Abstract

Conversational agents or chatbots (short for chat robot) are a branch of Natural Language Processing (NLP) that has arisen a lot of interest nowadays due to the extent number of applications in company services such as customer support or automatized FAQS and personal asistent services, for instance Siri or Cortana. There are three types: rule-based models, retrieval-based models and generative-based models. The difference between them is the freedom they have at the time of generating an answer given a question. The chatbot models usually used in public services are rule-based or retrieval-based given the need to guarantee quality and adecuate answers to users. But these models can handle only conversations aligned with their previous written answers and, therefore, conversations can sometimes sound artificial if it goes out of the topic. Generative-based models can handle better an open conversation which makes them a more generalizable approach. Promising results have been achieved in generative-based models by applying neural machine translation techniques with the recurrent encoder/decoder architecture. In this project is implemented, compared and analyzed two generative models that constitute the state of the art in neural machine translation applied to chatbots. One model is based on recurrence with attention and the other is based exclusively in attention. Additionally, the model based exclusively on recurrence has been used as a reference. Experiments show that, as in translation, an architecture based only in attention mechanisms obtains better results than the recurrence based models.

Chapter 1 Introduction

Chapter Contents

1.1 Conversational agent or chatbot

A conversational agent or chatbot is a language recognition system able to maintain a conversation with an user using a question/answer protocol. The communication can be done by audio or text media. As far as concerned in this project, we are going to focus on the textual models. The first chatbot models were rule-based, for instance ELIZA, PARRY1 and A.L.I.C.E2 . These models require a programmer to write some rules and patterns beforehand for the analysis and decomposition of a sentence and then, create an answer by the combination of a template and keywords. Thanks to the latest advances in machine learning and more specifically in artificial neural networks, it is possible to create chatbot models that do not longer require previous written rules. Instead, given a set of examples, the chatbot learns the pattern inherent to the samples. There are two different approaches depending on thefreedom they have at the time of generating an answer: retrieval-based and generative-based.

1.2 The retrieval-based systems

The retrieval-based systems determine which answer, from a set of answers previously written, is the most appropriate given a sentence/question as an input. These models are quite useful when the target domain is limited (e.g. a model trained only for sport or medicine conversations) and the chatbot is not allowed to commit grammatical or semantic errors during its service, for instance in FAQs3 and costumer support services. The problem is that they barely handle unseen questions and became impractical in open domains (e.g. general knowledge).

1.3 Generative models

Generative models, on the other hand, are trained to generate data as a response word by word. Nevertheless, not having rules implies that they have to learn to build sentences during their training. For that reason, they are more complex and harder to train than the retrieval-based systems. Usually generative models are prone to commit grammatical and semantic errors but on the other hand, they better handle new data and can answer with more natural sentences. What makes them an interesting approach is that they are an advance towards what is known as Strong Artificial Intelligence4 (Strong AI), is the system itself who analyze, compute and build an answer thanks to an autonomous learning without any human intervention.

1.4 Encoder/decoder

A great step forward to the area of generative-based chatbots was the implementation of a model using an encoder/decoder architecture with recurrent neural networks known as sequence to sequence (Seq2Seq) [12] used in translation. This project is motivated by the good results shown in the experiment, which achieve a new state of the art in the generative-based chatbot area.

In this internship I am interested in finding a solution for this issue in order to rewrite obsolete queries in connected environments and provide a query rewriter that handles query obsolescence and keeps up with the environment dynamicity.

1.5 Requirements and specifications

As one of the main languages used in machine learning nowadays, this project has been developed completely in Python 3.5.3. using the open-source software library for machine learning TensorFlow 5 for both implementation and training of models. for the execution environment I used Google colab interface because it gives a better GPU solution for the execution of big amount of data.

Chapter 2 Methodology and concept

Chapter Contents

2.1 Machine Learning

Machine Learning is a field of study of AI that studies and develop techniques capable to learn tasks as classification or regression from a data set. There are different algorithms without being any of them, in general, better among the others (No Free Lunch Theorem) 1 . The suitability of one algorithm in particular, depends exclusively on the nature and type of the problem addressed. The aim of a learning algorithm is to estimate the behaviour of a training set by the identification of their inherent pattern. Once accomplished, it must be capable to perform tasks as classification or regression given unseen samples.

2.2 Natural Language Processing - NLP

Natural Language Processing (NLP) is a research area of Artificial Intelligence (AI) which focus in the study and development of systems that allows the communication between a person and a machine through natural language. Chatbots belong to the area of NLP given the importance of their ability to understand natural language and know how to extract relevant information from it. Both models, retrieval-based and generative-based must be able to identify some information from the input sentence in order to pick or create an answer.

2.3 Biological Neuron

The learning algorithm at which focuses this project, is inspired by the biological neurons of the brain. Neurons are a type of cell from the nervous system composed by a cell body called soma, some input signal branches called dendrites and a single output signal branch called axon.

Axons split in their extremities into different sub-branches called tellodendrites. Connection between tellodendrites of a neuron and dendrites (or directly the soma) of another is performed by the synaptic terminal, which is a small structure that contains neurotransmitter molecules responsible for the exchange of the nervous signal (also known as synapses). Neurons, emit electrical impulses by the axon if the amount of electrical excitation received by the dendrites exceeds a threshold.

2.3.1 Artificial Neural Networks

A neural network is a type of machine learning algorithm that is inspired by the behaviour of biological neurons in the brain. It consists of a group of basic units called artificial neurons

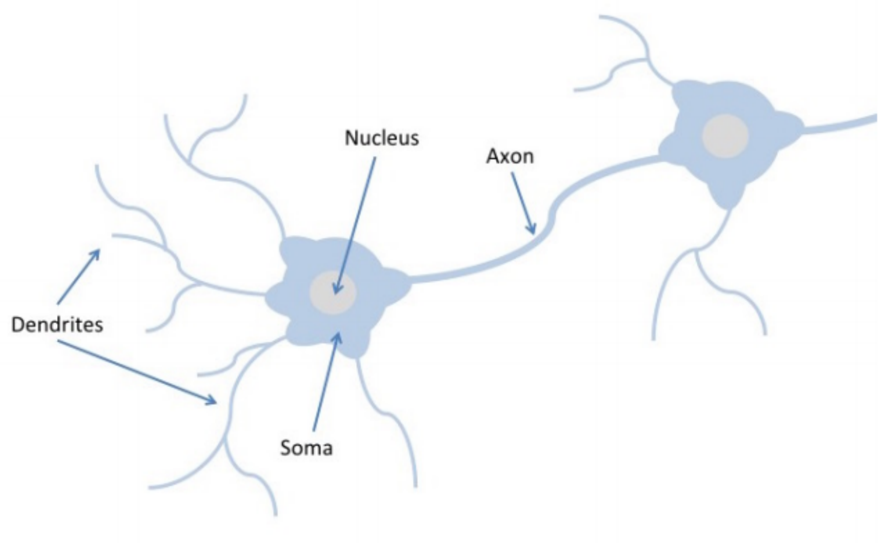


Figure 2.1: Structure of a neuron cell

(AN) or perceptron which are connected among them composing a complex network. They can compute an output given input data by decomposing it in different representations in order to identify different characteristics. The first model of AN was proposed by neurophysiologist Warren McCulloch and mathematician Walter Pitts in 1943 . The proposed model is a simple mathematical approximation of the operation of a biological neuron, capable to compute basic operations as identity function, AND, OR and NOR. Many other models have been proposed since then, but the most simple AN architecture is the perceptron which was proposed by Frank Rosenblatt in 1957. Whilst the AN model proposed by McCulloch and Pitts used binary values, the perceptron can operate with any numbers. The algorithm computes an activation function over the weighted sum of the input values. Additionally, in order to give one extra degree of freedom, a bias is added as shown in the following equation:

$$output = f(\sum_i x_i * w_i + w_0)$$

Figure 2.2

where wi and wO are the weights and bias respectively. The optimal values of these parameters are computed using gradient descent techniques starting from a labeled training data set2 . Gradient Descent is an iterative optimization algorithm used to find the global minimum of a function. Activation functions are continuous and differentiable non-linear functions. They are required to be smooth in order to be able to learn from gradient descent techniques.

The non-linearity is an important condition that ensures a non linear discriminant expression

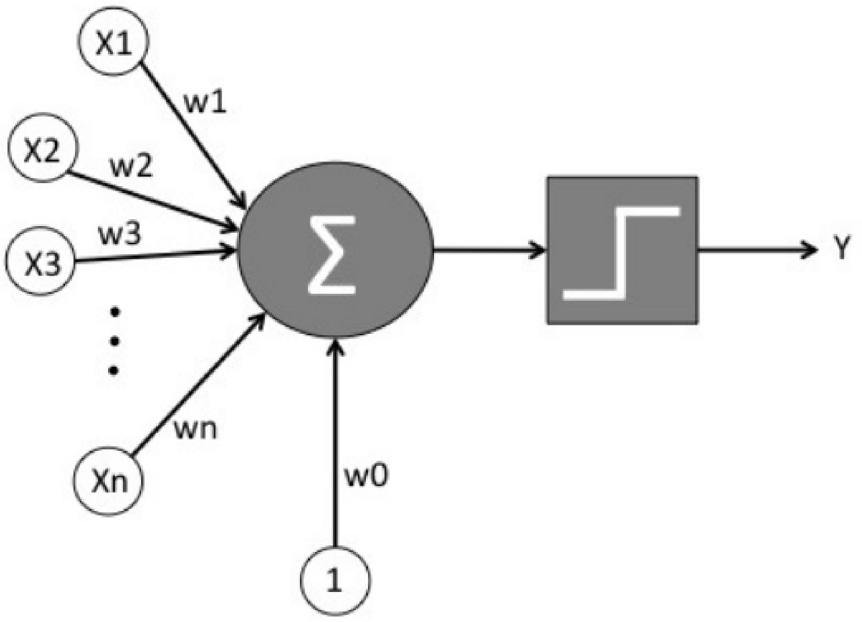


Figure 2.3: Structure of a perceptron

at the output of the neural network, on the contrary a multilayer and single layer network perform alike. Output values are binary, if the weighted sum exceeds a threshold imposed by the activation function (originally the Heaviside step function), then the output is activated on the other hand the output value is deactivated.

The solution is a multilayer perceptron (MLP) which is a network composed by multiple layers of perceptrons (Figure 2.4). The basic structure of a MLP is composed by an input layer where all data is fed to the network, one or more hidden layers for multiple representations of data and characteristic identification and finally an output layer. The output layer can use a different activation function depending on the nature of the task. For classification, the output layer uses a softmax function that represents, for each target class, a probability of success.

2.3.2 Recurrent Neural Networks - RNN

Two of the models covered by this project use a special type of artificial neural network called recurrent neural network (RNN). RNN have the ability to retain information from previous data as a temporal memory. They can be viewed as a sequence of concatenations of the same unit like a chain, where each one computes an output given an input and the information provided by the last network.

Due to its temporal memory, they are quite useful for sequential data were each element of the sequence is related to the the others. Nevertheless, RNN can only retain recent information

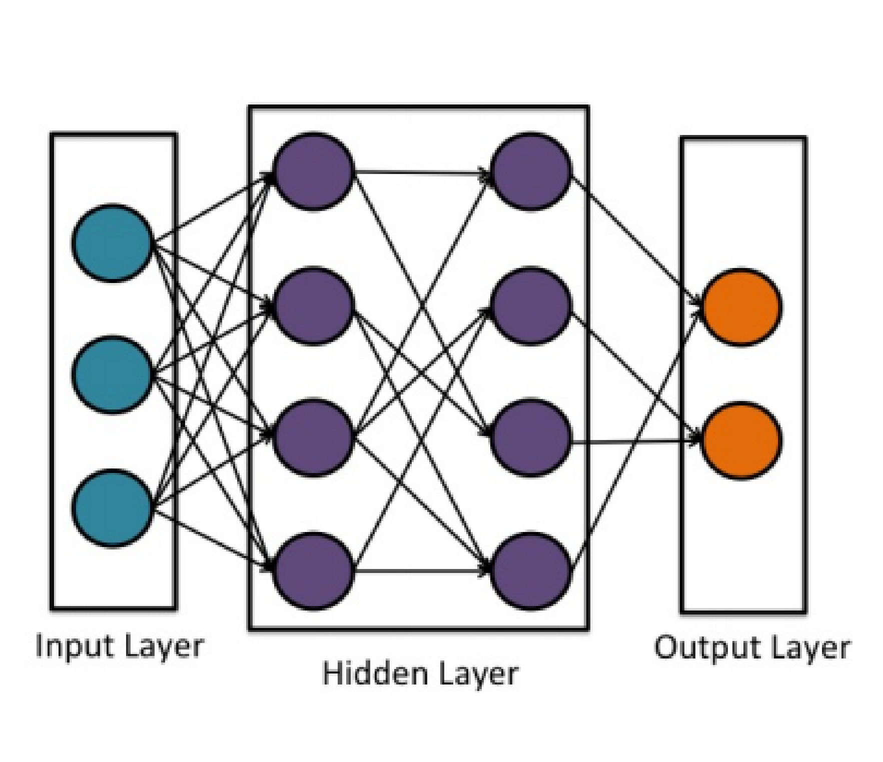


Figure 2.4: Structure of a multilayer perceptron

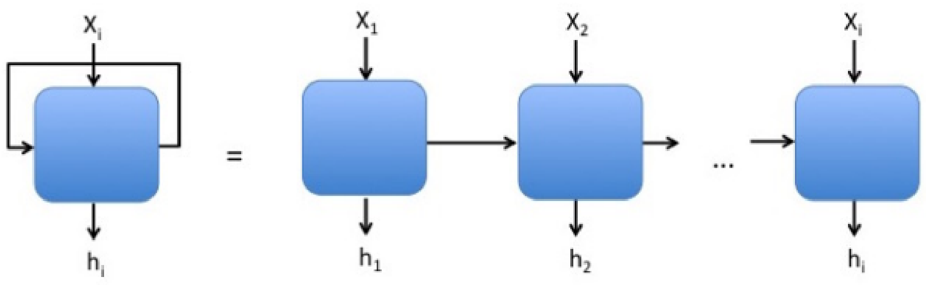


Figure 2.5: Diagram of a recurrent neural network

from a sequence, which means that they can only perform correctly when the element to be processed is near at the sequence to the relevant information.

2.4 RNN Encoder/Decoder architecture (Seq2Seq)

A very successful implementation of Encoder/Decoder architecture for NLP tasks (specially in neural machine translation) is the RNN Encoder/Decoder also known as Sequence to Sequence (Seq2Seq). The encoder and decoder are recurrent neural networks, which allows the model to be fed with variable-length input sentences. Given an input sentence, the encoder iteratively computes for each word a hidden state vector using the word and previous hidden state of the RNN. Once the whole sentence has been analyzed, the relevant information of the input sentence is contained in the last hidden state of the RNN, known as context or thought vector. The decoder computes, word by word, an output in the original representation space using the information contained in the context vector and previous decoded words. The architecture implementation can vary depending on the type of RNN cell used (genuine RNN cell, a LSTM cell or a GRU cell), number of cells per layer or the number of hidden layers among other parameters. Figure 3.1 shows a diagram of the sequence to sequence architecture.

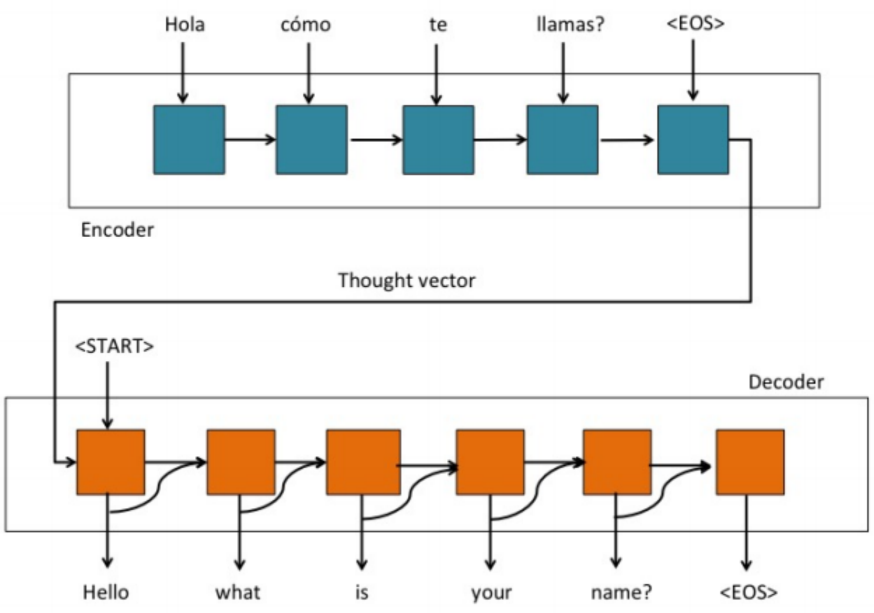


Figure 2.6: Diagram of the sequence to sequence architecture

As the size of the sentence increases, it is needed to encode a large quantity of information into a fixed-length vector, so some of it is lost at the encoding process resulting in a poor performance of the chatbot.

Chapter 3 Implementation

Chapter Contents

3.1 Introduction

In this project all code has been written in Python using Tensor Flow libraries. For the implementation of the baseline model and the Seq2Seq + Attention model, it has been used code written by Google Brain resident Etienne Pot1 in TensorFlow available at his GitHub page2 . The software consists of a simple implementation of the Seq2Seq architecture plus an interface with a wide range of adjustable parameters and tools. On the other hand, the transformer model has been created using TensorFlow's library Tensor2Tensor, which was published by researchers from as an open-source code for the recreation of their model.

3.2 Data bases

Along the project, two data sets have been used:

3.2.1 Questions

3.2.2 Answers

3.3 Data preparation

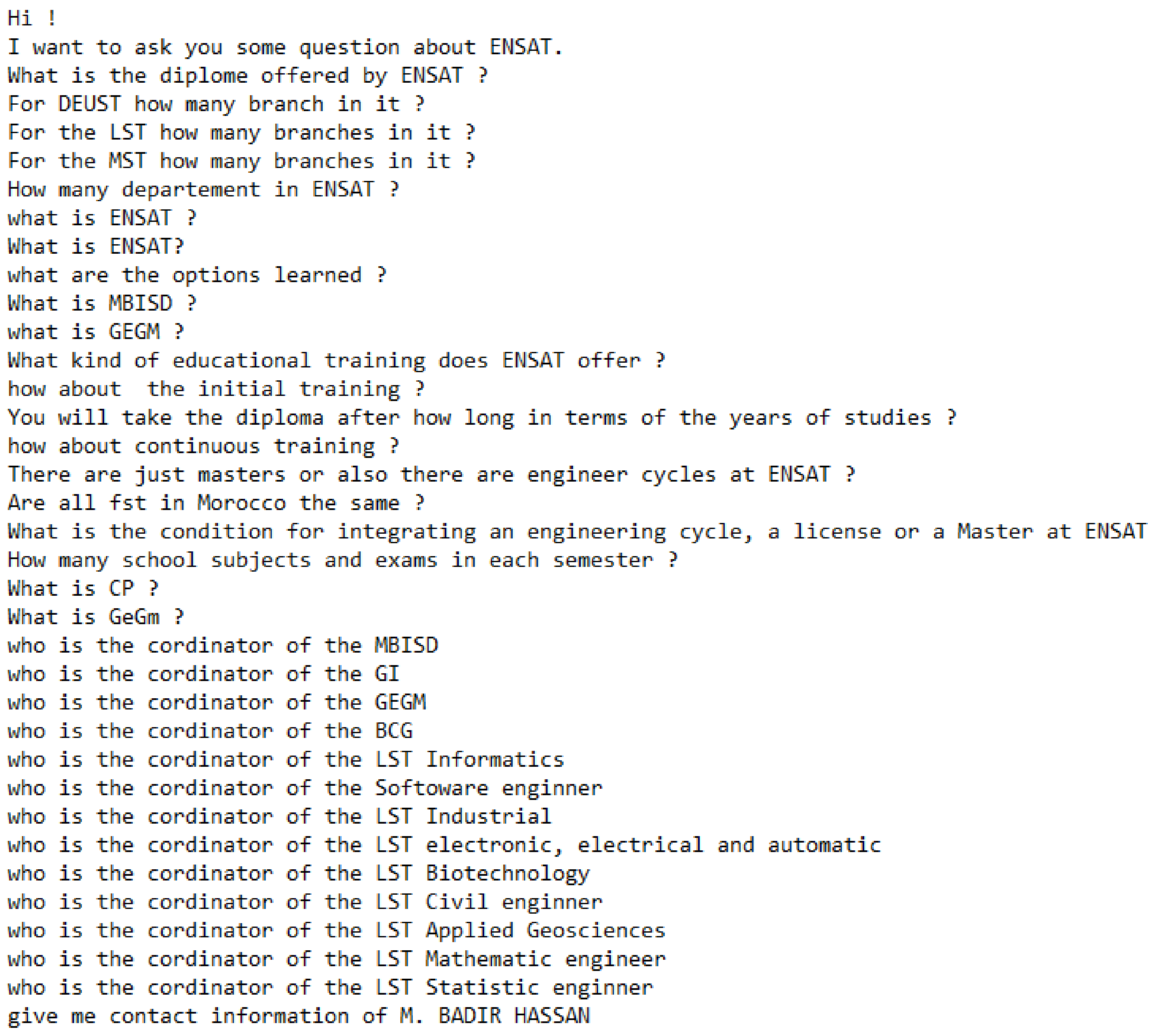
For the preparation of the data, we tried to collect a set of data from our classmates by creating a form that allows to do a question answer about Ensat and exactly IT department.

3.4 Data Preprocessing

First, we import the dataset, then we create a dictionary that maps each row to its identifier, after that we create a list of all conversations to get questions and answers separately. We simplify and clean up the text using regular expressions:

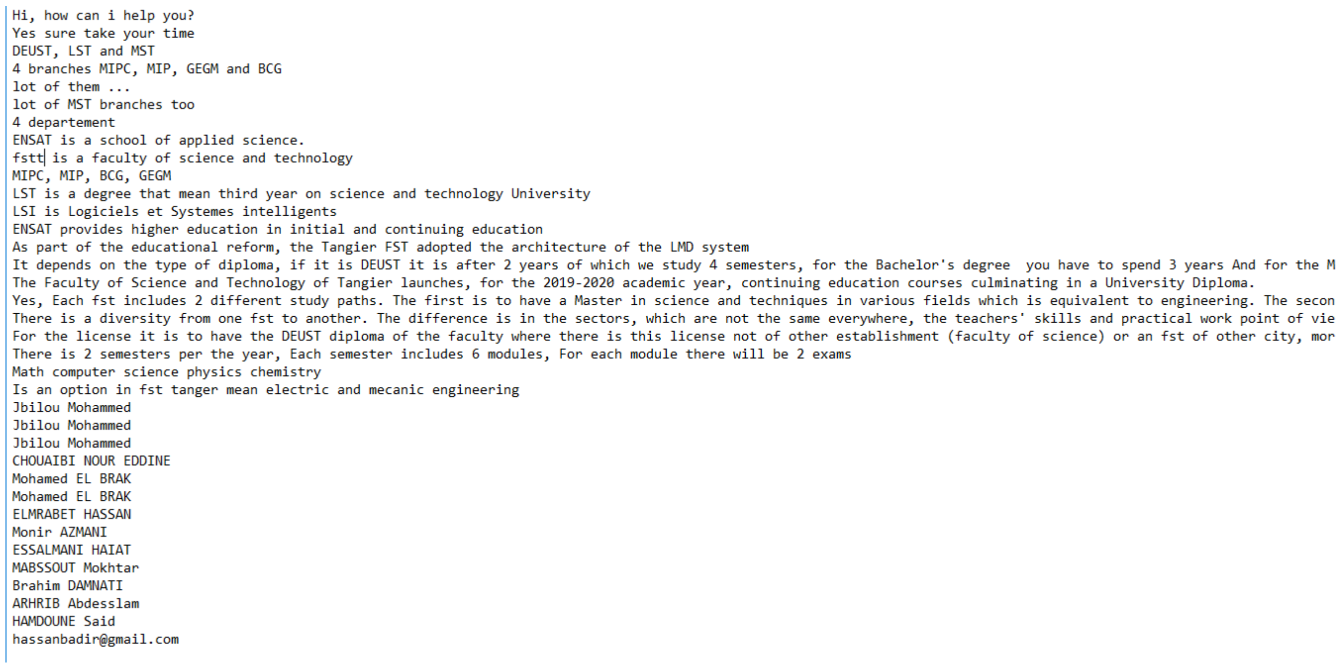
- Cleaning questions.
- Cleanup of responses.
- Filtering questions and answers that are too short or too long.

We create a dictionary that maps each word to its number of occurrences and we create two dictionaries that map the words in the questions and the answers to a single integer. We add the last tokens to the two dictionaries. Then we create an inverse dictionary of the "answerswordstoint" dictionary. We add the EOS token at the end of each response and we translate all the questions and answers in full and replace all the words that have been filtered by the OUT token. Finally,



```
Hi !
I want to ask you some question about ENSAT.
What is the diplome offered by ENSAT ?
For DEUST how many branch in it ?
For the LST how many branches in it ?
For the MST how many branches in it ?
How many departement in ENSAT ?
what is ENSAT ?
What is ENSAT?
what are the options learned ?
What is MBISD ?
what is GEGM ?
What kind of educational training does ENSAT offer ?
how about  the initial training ?
You will take the diploma after how long in terms of the years of studies ?
how about continuous training ?
There are just masters or also there are engineer cycles at ENSAT ?
Are all fst in Morocco the same ?
What is the condition for integrating an engineering cycle, a license or a Master at ENSAT
How many school subjects and exams in each semester ?
What is CP ?
What is GeGm ?
who is the cordinator of the MBISD
who is the cordinator of the GI
who is the cordinator of the GEGM
who is the cordinator of the BCG
who is the cordinator of the LST Informatics
who is the cordinator of the Softoware enginner
who is the cordinator of the LST Industrial
who is the cordinator of the LST electronic, electrical and automatic
who is the cordinator of the LST Biotechnology
who is the cordinator of the LST Civil enginner
who is the cordinator of the LST Applied Geosciences
who is the cordinator of the LST Mathematic engineer
who is the cordinator of the LST Statistic enginner
give me contact information of M. BADIR HASSAN
```

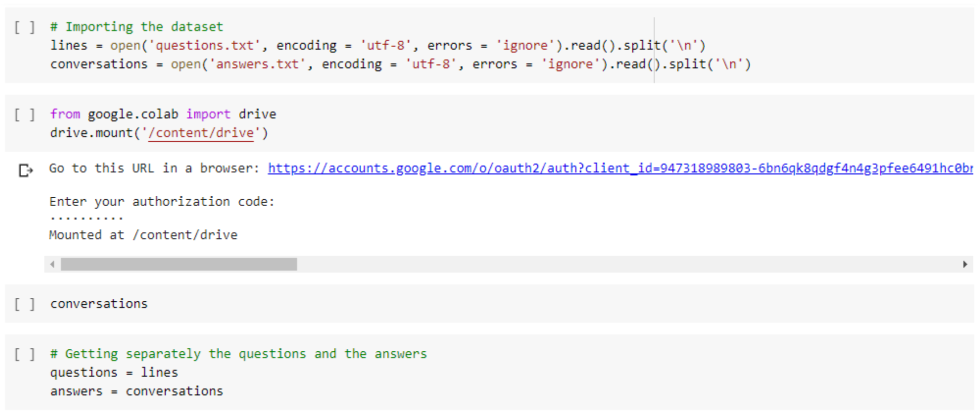
Figure 3.1: Questions data set



```
Hi, how can i help you?
Yes sure take your time
DEUST, LST and MST
4 branches MIPC, MIP, GEGM and BCG
lot of them ...
lot of MST branches too
4 departement
ENSAT is a school of applied science.
fstt is a faculty of science and technology
MIPC, MIP, BCG, GEGM
LST is a degree that mean third year on science and technology University
LSI is Logiciels et Systemes intelligents
ENSAT provides higher education in initial and continuing education
As part of the educational reform, the Tangier FST adopted the architecture of the LMD system
It depends on the type of diploma, if it is DEUST it is after 2 years of which we study 4 semesters, for the Bachelor's degree  you have to spend 3 years And for the M
The Faculty of Science and Technology of Tangier launches, for the 2019-2020 academic year, continuing education courses culminating in a University Diploma.
Yes, Each fst includes 2 different study paths. The first is to have a Master in science and techniques in various fields which is equivalent to engineering. The secon
There is a diversity from one fst to another. The difference is in the sectors, which are not the same everywhere, the teachers' skills and practical work point of vie
For the license it is to have the DEUST diploma of the faculty where there is this license not of other establishment (faculty of science) or an fst of other city, mor
There is 2 semesters per the year, Each semester includes 6 modules, For each module there will be 2 exams
Math computer science physics chemistry
Is an option in fst tanger mean electric and mecanic engineering
Jbilou Mohammed
Jbilou Mohammed
Jbilou Mohammed
CHOUAIBI NOUR EDDINE
Mohamed EL BRAK
Mohamed EL BRAK
ELMRABET HASSAN
Monir AZMANI
ESSALMANI HAIAT
MABSSOUT Mokhtar
Brahim DAMNATI
ARHRIB Abdesslam
HAMDOUNE Said
hassanbadir@gmail.com
```

Figure 3.2: Answers data set

we sort the questions and answers according to the length of the questions.



```
[ ] # Importing the dataset
    lines = open('questions.txt', encoding = 'utf-8', errors = 'ignore').read().split('\n')
    conversations = open('answers.txt', encoding = 'utf-8', errors = 'ignore').read().split('\n')

[ ] from google.colab import drive
    drive.mount('/content/drive')

    Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0br

    Enter your authorization code:
    ··········
    Mounted at /content/drive

[ ] conversations

[ ] # Getting separately the questions and the answers
    questions = lines
    answers = conversations
```

Figure 3.3: Data preprocessing

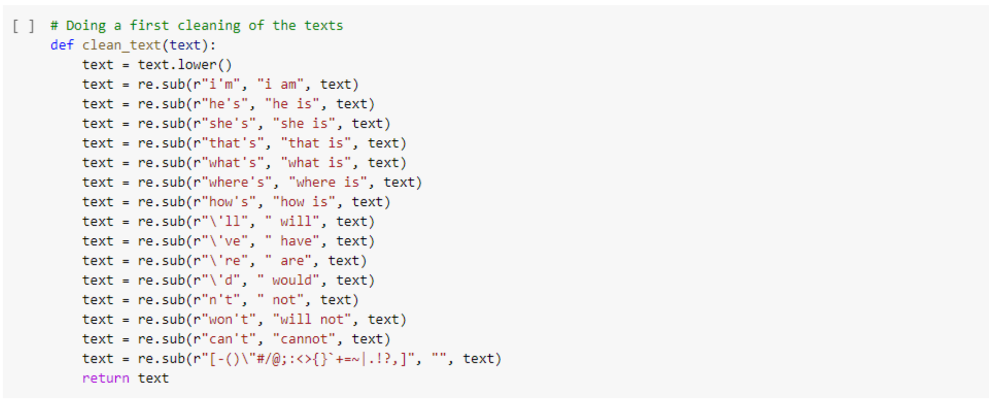
3.5 Data cleaning

3.6 Construction of Seq2Seq model

3.6.1 Creation of placeholders for entries and targets:

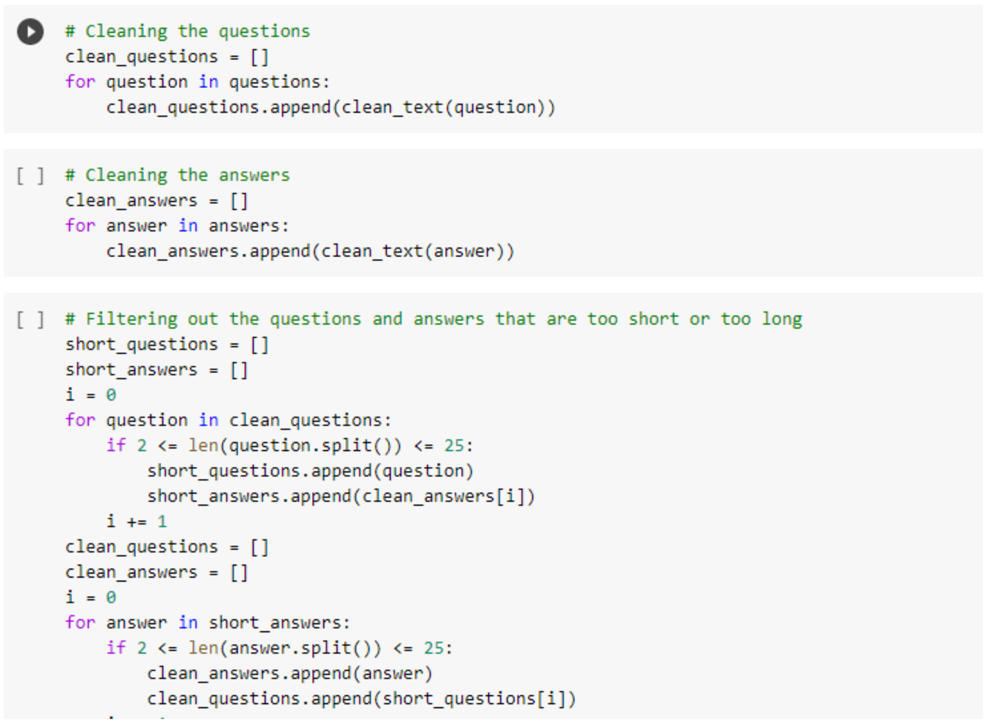
We create reserved spaces for the entrances of our models.

- tf.placeholder accepts 3 parameters (data type, input data dimensions, input name) Input data dimensions must be two-dimensional (in tf.placeholder) because neural networks can only accept grouped entries, as opposed to a single entry. Thus, we add 1 dimension



```
[ ] # Doing a first cleaning of the texts
    def clean_text(text):
        text = text.lower()
        text = re.sub(r"i'm", "i am", text)
        text = re.sub(r"he's", "he is", text)
        text = re.sub(r"she's", "she is", text)
        text = re.sub(r"that's", "that is", text)
        text = re.sub(r"what's", "what is", text)
        text = re.sub(r"where's", "where is", text)
        text = re.sub(r"how's", "how is", text)
        text = re.sub(r"\'ll", " will", text)
        text = re.sub(r"\'ve", " have", text)
        text = re.sub(r"\'re", " are", text)
        text = re.sub(r"\'d", " would", text)
        text = re.sub(r"n't", " not", text)
        text = re.sub(r"won't", "will not", text)
        text = re.sub(r"can't", "cannot", text)
        text = re.sub(r"[-()\"#/@;:<>{}`+=~|.!?,]", "", text)
        return text
```

Figure 3.4: Cleaning the input of answers

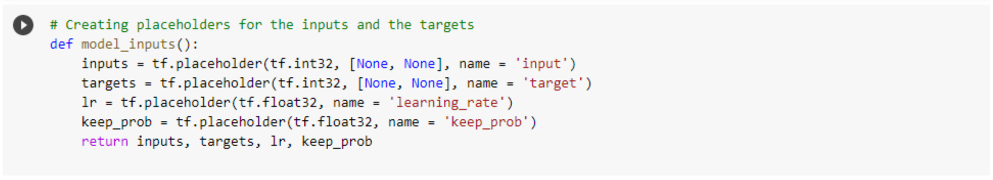


```
# Cleaning the questions
clean_questions = []
for question in questions:
    clean_questions.append(clean_text(question))
```

```
[ ] # Cleaning the answers
    clean_answers = []
    for answer in answers:
        clean_answers.append(clean_text(answer))
```

```
[ ] # Filtering out the questions and answers that are too short or too long
    short_questions = []
    short_answers = []
    i = 0
    for question in clean_questions:
        if 2 <= len(question.split()) <= 25:
            short_questions.append(question)
            short_answers.append(clean_answers[i])
        i += 1
    clean_questions = []
    clean_answers = []
    i = 0
    for answer in short_answers:
        if 2 <= len(answer.split()) <= 25:
            clean_answers.append(answer)
            clean_questions.append(short_questions[i])
```

Figure 3.5: Filtering the input of answers



```
# Creating placeholders for the inputs and the targets
def model_inputs():
    inputs = tf.placeholder(tf.int32, [None, None], name = 'input')
    targets = tf.placeholder(tf.int32, [None, None], name = 'target')
    lr = tf.placeholder(tf.float32, name = 'learning_rate')
    keep_prob = tf.placeholder(tf.float32, name = 'keep_prob')
    return inputs, targets, lr, keep_prob
```

Figure 3.6: Python code for Creation of placeholders for entries and targets

corresponding to the lot.

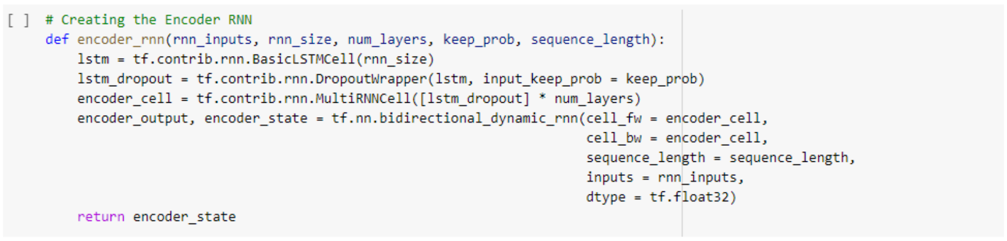
- keepProb is a hyperparameter used to control the dropout rate. Dropout rate is the rate of the neurons you choose to crush during the single iteration of training.

3.6.2 Target preprocessing:

The decoder only accepts a certain format of targets (targets must be in batches) so preprocessing is needed, we need to remove the last column from the dictionary before adding <SOS> to process target functions keep length of maximum sequence because after that we do a concatenation to add an <SOS> token at the start of the sequence, so we need to remove the last token before so that the length of the sequence does not exceed the maximum sequence length.

3.6.3 Creation of the RNN Encoder:

The encoder is a recurrent neural network. It takes as input the embeddings of our words and its output is the thought vector that the decoder will use. LSTM is a technique used for regularization in neural networks, it helps to learn and prevent overfitting with data. Overfitting occurs when a model learns the details and noise in the training data as it negatively affects the performance of the model on the new data. Encodercell: the cell inside the RNN encoder that contains the stacked LSTM layers on which we apply dropout improve accuracy Encoderstate: output returned by the RNN encoder, just after the last fully connected layer ...

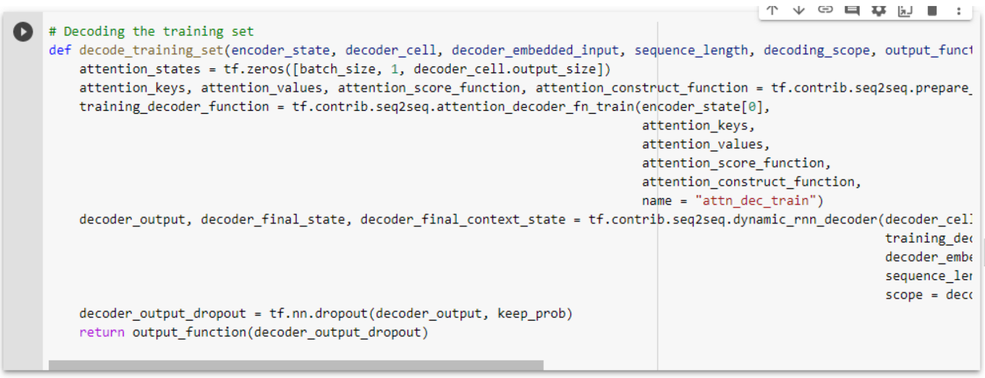


```
[ ] # Creating the Encoder RNN
    def encoder_rnn(rnn_inputs, rnn_size, num_layers, keep_prob, sequence_length):
        lstm = tf.contrib.rnn.BasicLSTMCell(rnn_size)
        lstm_dropout = tf.contrib.rnn.DropoutWrapper(lstm, input_keep_prob = keep_prob)
        encoder_cell = tf.contrib.rnn.MultiRNNCell([lstm_dropout] * num_layers)
        encoder_output, encoder_state = tf.nn.bidirectional_dynamic_rnn(cell_fw = encoder_cell,
                                                                        cell_bw = encoder_cell,
                                                                        sequence_length = sequence_length,
                                                                        inputs = rnn_inputs,
                                                                        dtype = tf.float32)
        return encoder_state
```

Figure 3.7: Python code for Creation of the RNN Encoder

3.6.4 Decoding the training set:

We need the decoding training set to decode the coded questions and answers of the training set. So this function decodes the observations of the training set and also returns the output of the decoder at the same time, which was for the observation of the training set which is an observation happening in the neural network for update the weights and thus update the chatbot's ability to speak like a human. Decoding the test / validation set: We need the decoding test set to decode the answers to the coded questions from the validation set, or just new predictions that are not used in training anyway. The validation set is the set that we make during the training Creation of the RNN Decoder:



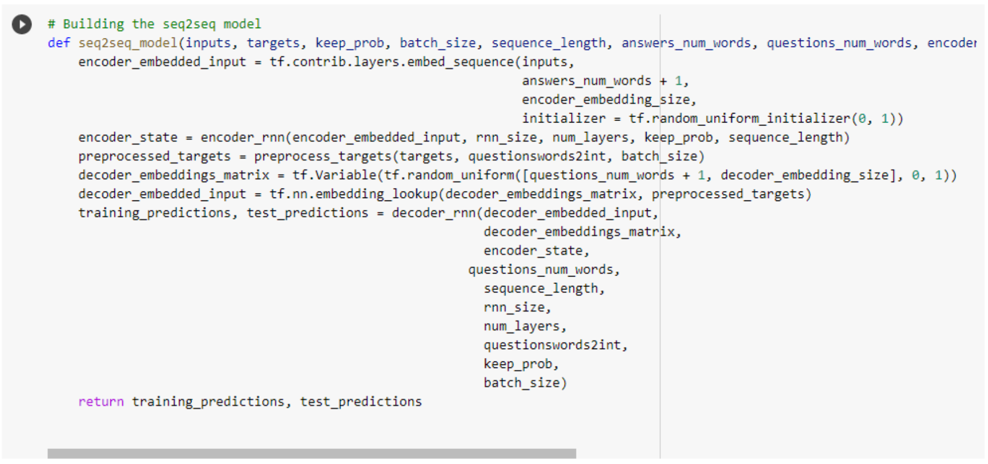
```python
# Decoding the training set
def decode_training_set(encoder_state, decoder_cell, decoder_embedded_input, sequence_length, decoding_scope, output_funct
    attention_states = tf.zeros([batch_size, 1, decoder_cell.output_size])
    attention_keys, attention_values, attention_score_function, attention_construct_function = tf.contrib.seq2seq.prepare_
    training_decoder_function = tf.contrib.seq2seq.attention_decoder_fn_train(encoder_state[0],
                                                                              attention_keys,
                                                                              attention_values,
                                                                              attention_score_function,
                                                                              attention_construct_function,
                                                                              name = "attn_dec_train")
    decoder_output, decoder_final_state, decoder_final_context_state = tf.contrib.seq2seq.dynamic_rnn_decoder(decoder_cel
                                                                                                              training_de
                                                                                                              decoder_embe
                                                                                                              sequence_le
                                                                                                              scope = dec

    decoder_output_dropout = tf.nn.dropout(decoder_output, keep_prob)
    return output_function(decoder_output_dropout)
```

Figure 3.8: Python code for decoding the training set

Once the decoder receives the last hidden state from the encoder and the decoding token, it begins to decode the input word word by word. The decoder must be powered at its output at the previous time step to produce the next output.

3.6.5 Building the seq2seq model:



```python
# Building the seq2seq model
def seq2seq_model(inputs, targets, keep_prob, batch_size, sequence_length, answers_num_words, questions_num_words, encoder
    encoder_embedded_input = tf.contrib.layers.embed_sequence(inputs,
                                                              answers_num_words + 1,
                                                              encoder_embedding_size,
                                                              initializer = tf.random_uniform_initializer(0, 1))
    encoder_state = encoder_rnn(encoder_embedded_input, rnn_size, num_layers, keep_prob, sequence_length)
    preprocessed_targets = preprocess_targets(targets, questionswords2int, batch_size)
    decoder_embeddings_matrix = tf.Variable(tf.random_uniform([questions_num_words + 1, decoder_embedding_size], 0, 1))
    decoder_embedded_input = tf.nn.embedding_lookup(decoder_embeddings_matrix, preprocessed_targets)
    training_predictions, test_predictions = decoder_rnn(decoder_embedded_input,
                                                         decoder_embeddings_matrix,
                                                         encoder_state,
                                                       questions_num_words,
                                                         sequence_length,
                                                         rnn_size,
                                                         num_layers,
                                                         questionswords2int,
                                                         keep_prob,
                                                         batch_size)
    return training_predictions, test_predictions
```

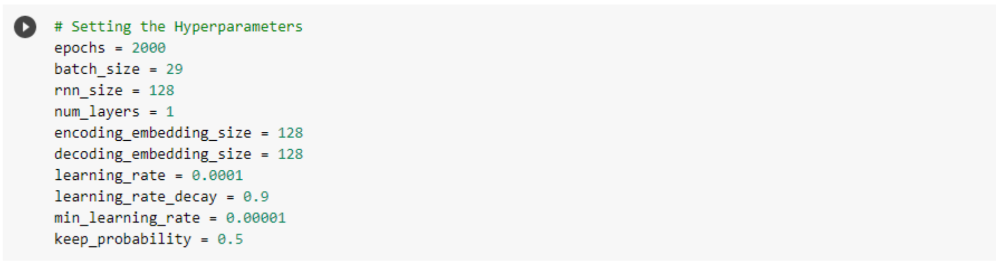
Figure 3.9: Python code for Buildng the seq2seq model

This function returns the training predictions and the test predictions but we want them to be returned at the same time. So, we are going to assemble an encoder that returns the encoder state and a decoder that returns training and testing predictions. But we also need an encoder because in order for the decoder to return test and training predictions it has to take the encoder state returned by the encoder and that is why we need these 2 networks in this function.

3.7 Training the SEQ2SEQ Model

3.7.1 Regulation of Hyperparameters:

Epochs: An epoch is a whole process to get the batches of inputs into the neural network and then propagate them inside the encoders to get the encoder state and to propagate the encoder states with the targets to the inside the RNN decoder to get the final output. First, the final output results then the final responses predicted by the chatbot, then the propagation of the loss generated by the output and the target in the neural network and updating the weight of the chatbot's best ability to work as a human. An epoch is basically an entire iteration of training. NumLayers: number of layers you have in RNN encoder and decoder EncodingEmbeddingSize: it is the number of columns in the integration matrix which is the number of columns that you want to have as value of embeddings where in the matrix each row corresponds to each token in the whole question of the corpus. LearningRate: it should not be too high or too low. If it is too high the model will learn too quickly and therefore will not learn to speak correctly and if it is low the model will take a long time to learn correctly. Learning rate decay: it means by what percentage the learning rate is reduced during training iterations because we want to start with a certain learning rate but we want to apply the decay on the training iterations to reduce the learning rate. learning rate so that it can learn more in depth the logic of human conversations and in our case, or general the correlations found in the dataset. keepProb: used to apply the stall to improve model accuracy.



```
# Setting the Hyperparameters
epochs = 2000
batch_size = 29
rnn_size = 128
num_layers = 1
encoding_embedding_size = 128
decoding_embedding_size = 128
learning_rate = 0.0001
learning_rate_decay = 0.9
min_learning_rate = 0.00001
keep_probability = 0.5
```

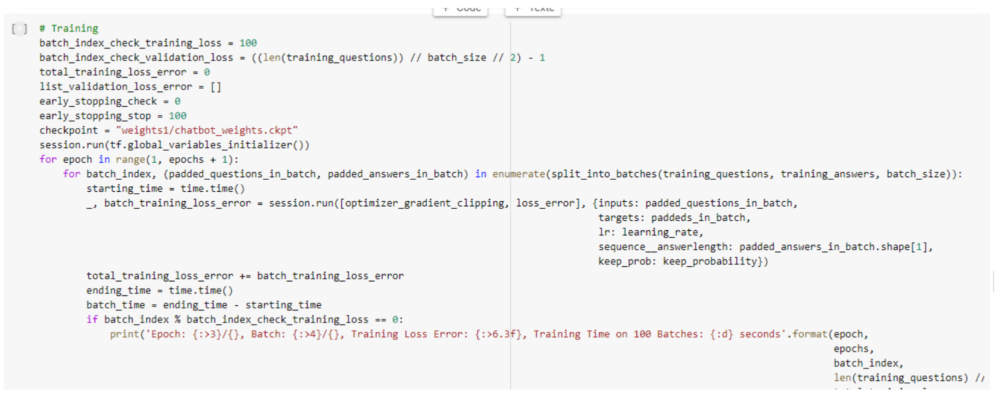
Figure 3.10: Python code for Setting the Hyperparameters

3.7.2 Divide the data into sets of questions and answers:

We need startIndex to set the first index of the question we are adding to the batch as we are dealing with a specific batch. So this is the first question / answer that we add to the batch. To get it we use the batch index * the batch size due to starting at index 0. The first question added will have the starting index of 0, overall in the split loop in batches, it is used for this special indexing.

3.7.3 Training:

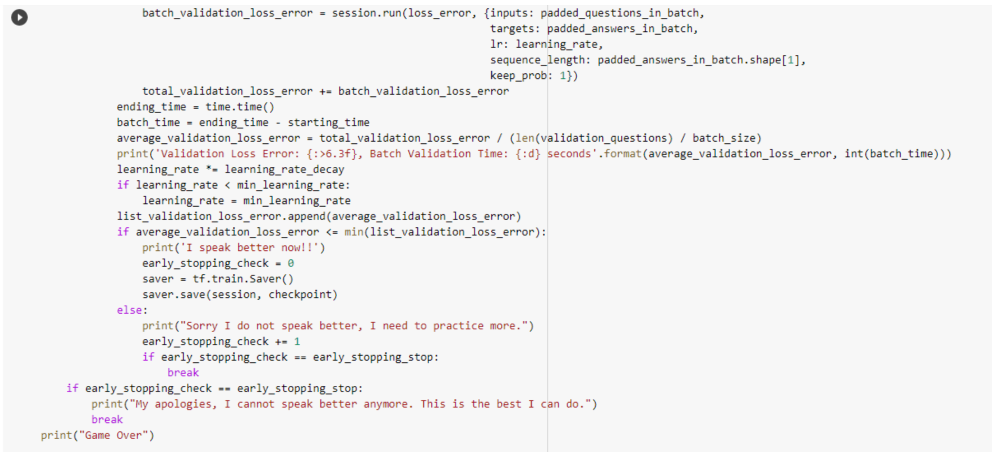
TotalTrainingLossError: will be used to calculate training losses on 100 lots ListValidationLossError: we have to make a list, we have to use the early stepping technique which is to check if we have reached a loss that is below the minimum of all the losses we have suffered and all the losses we have, we put it in this list. EarlyStoppingCheck: is the number of checks whenever there is no improvement in commit loss, so anytime we don't reduce commit loss, the early progress check is going to be incremented by one and one, it hits a number that's going to be our next variable. EarlyStoppingStop: we will stop the training. We choose 1000 because we want to make the formation last until the last epoch which is 100, but if you want to apply an early step, use a value of 100. Checkpoint: this variable will be used to record the weights that we can load each time we want to chat with our trained chatbot. Session.run (tf.globalVariableInitializer ()): used to run a session BatchTime = endHour - startTime: used to calculate the formation time of each batch.



```
# Training
batch_index_check_training_loss = 100
batch_index_check_validation_loss = ((len(training_questions)) // batch_size // 2) - 1
total_training_loss_error = 0
list_validation_loss_error = []
early_stopping_check = 0
early_stopping_stop = 100
checkpoint = "weights1/chatbot_weights.ckpt"
session.run(tf.global_variables_initializer())
for epoch in range(1, epochs + 1):
    for batch_index, (padded_questions_in_batch, padded_answers_in_batch) in enumerate(split_into_batches(training_questions, training_answers, batch_size)):
        starting_time = time.time()
        _, batch_training_loss_error = session.run([optimizer_gradient_clipping, loss_error], {inputs: padded_questions_in_batch,
                                                                                               targets: paddeds_in_batch,
                                                                                               lr: learning_rate,
                                                                                               sequence__answerlength: padded_answers_in_batch.shape[1],
                                                                                               keep_prob: keep_probability})


        total_training_loss_error += batch_training_loss_error
        ending_time = time.time()
        batch_time = ending_time - starting_time
        if batch_index % batch_index_check_training_loss == 0:
            print('Epoch: {:>3}/{}, Batch: {:>4}/{}, Training Loss Error: {:>6.3f}, Training Time on 100 Batches: {:d} seconds'.format(epoch,
                                                                                                                                        epochs,
                                                                                                                                        batch_index,
                                                                                                                                        len(training_questions) //
```

Figure 3.11: Python code for the training1

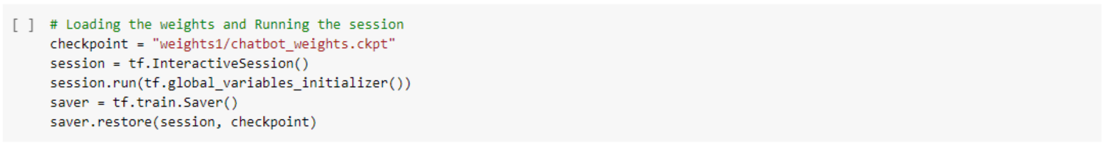


```
                batch_validation_loss_error = session.run(loss_error, {inputs: padded_questions_in_batch,
                                                                       targets: padded_answers_in_batch,
                                                                       lr: learning_rate,
                                                                       sequence_length: padded_answers_in_batch.shape[1],
                                                                       keep_prob: 1})
                total_validation_loss_error += batch_validation_loss_error
            ending_time = time.time()
            batch_time = ending_time - starting_time
            average_validation_loss_error = total_validation_loss_error / (len(validation_questions) / batch_size)
            print('Validation Loss Error: {:>6.3f}, Batch Validation Time: {:d} seconds'.format(average_validation_loss_error, int(batch_time)))
            learning_rate *= learning_rate_decay
            if learning_rate < min_learning_rate:
                learning_rate = min_learning_rate
            list_validation_loss_error.append(average_validation_loss_error)
            if average_validation_loss_error <= min(list_validation_loss_error):
                print('I speak better now!!')
                early_stopping_check = 0
                saver = tf.train.Saver()
                saver.save(session, checkpoint)
            else:
                print("Sorry I do not speak better, I need to practice more.")
                early_stopping_check += 1
                if early_stopping_check == early_stopping_stop:
                    break
    if early_stopping_check == early_stopping_stop:
        print("My apologies, I cannot speak better anymore. This is the best I can do.")
        break
print("Game Over")
```

Figure 3.12: Python code for the training2

3.8 Testing Ses2Seq Model

Loading weights and running the session, conversion of string questions into integer encoding lists and chat configuration



```
# Loading the weights and Running the session
checkpoint = "weights1/chatbot_weights.ckpt"
session = tf.InteractiveSession()
session.run(tf.global_variables_initializer())
saver = tf.train.Saver()
saver.restore(session, checkpoint)
```

Figure 3.13: Python code for the testing phase

Conclusion

Chatbots are an essential application of artificial intelligence and the chatbot industry is booming right now with top companies using chatbots in their latest devices such as Alexa by Amazon or Bixby by Samsung. The chatbot domain remains difficult, such as how to improve responses and choose the best model that produces the most appropriate response based on the request and Tensorflow package gives an efficient tool to make chatbot experience.