

Development of Chatbot Using Deep NLP and Python

Project report submitted in partial fulfillment of the requirement for the
degree of Bachelor of Technology

In

Computer Science and Engineering

By

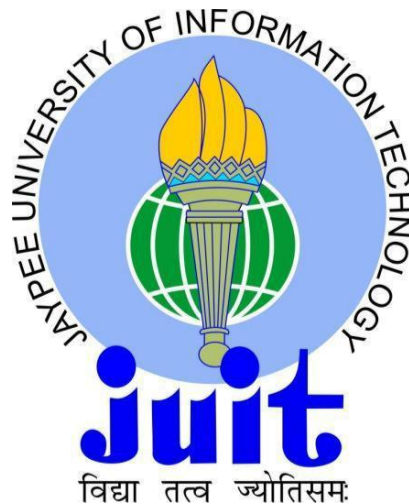
Gokaran (151270)

Ayush (151263)

Under the supervision of

Dr. Jagpreet Sidhu

To



Department of Computer Science & Engineering and Information Technology

**Jaypee University of Information Technology Waknaghat, Solan-173234,
Himachal Pradesh**

CERTIFICATE

Candidate's Declaration

This is to certify that the work which is being presented in the report entitled **“Development of Chatbot using Deep NLP and Python”** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology Waknaghat is an authentic record of our own work carried out over a period from August 2018 to May 2019 under the supervision of **Dr. Jagpreet Sidhu** (Assistant Professor, Senior Grade, Computer Science & Engineering Department).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

This is to certify that the above statement made by the candidates is true to the best of my knowledge.

Dr. Jagpreet Sidhu
Assistant Professor (Senior Grade)
Computer Science & Engineering Department
Dated:

ACKNOWLEDGEMENT

We owe our profound gratitude to our project supervisor **Dr. Jagpreet Sidhu**, who took keen interest and guided us all along in my project work titled — **Development of Chatbot using Deep NLP and Python**, till the completion of our project by providing all the necessary information for developing the project. The project development helped us in research and we got to know a lot of new things in our domain. We are really thankful to him.

TABLE OF CONTENTS

CERTIFICATE.....	i
ACKNOWLEDGEMENT	ii
ABSTRACT	5
1)INTRODUCTION.....	6
1.1)THE CHATBOT.....	7
1.2)PROBLEM STATEMENT.....	9
1.3)METHODOLOGY.....	9
1.4) DESIGN FUNDAMENTALS OF... BOT.....	10
2)LITEATURE SURVEY	
2.1) RESEARCH PAPER-1.....	15
2.2) RESEARCH PAPER-2.....	16
3) SYSTEM DEVLOPMENT	
3.1)SOFTWARE REQUIREMENTS.....	17
3.2)HARDWARE REQUIREMENTS.....	17
4)ALGORITHMS.....	18-44
5)RESULT.....	45
6)CONCLUSION.....	46
7) REFERENCES.....	47

ABSTRACT

User's interface for software apps come in a variety of format, ranging from command-line, web application, and even voice. The popular user interface include graphical and web-based application, the need arise for an alternative interface. Whether due to multi-threaded complexity, or surrounding execution of the service, a chatbot is the need.

Chatbot provide a text-based User Interface, thus allowing the user to type command and receive text as well as text to speech response. Chatbots are usually a stateful service, remembering previous commands in order to provide functionality. When chatbot is integrated with web services it can be used by an even larger audience.

1. INTRODUCTION

A Chat-Bot is a man made person, animal holding talks with humans. Chat-Bot can be a text based, spoken or can be a non-flask communication. Chat bot can work both on PCs and smart phones, but mostly works on the internet. Chat-bot can be understood a machine which can communicate with humans. They can converse in almost every language. Their (Natural Language Processing, NLP) skill can varies from exceedingly idle to clever, needy and plumb. Sometime it can feel like a cartoon made by a kid and other side it can be realistic. They all are “chat-bots”.

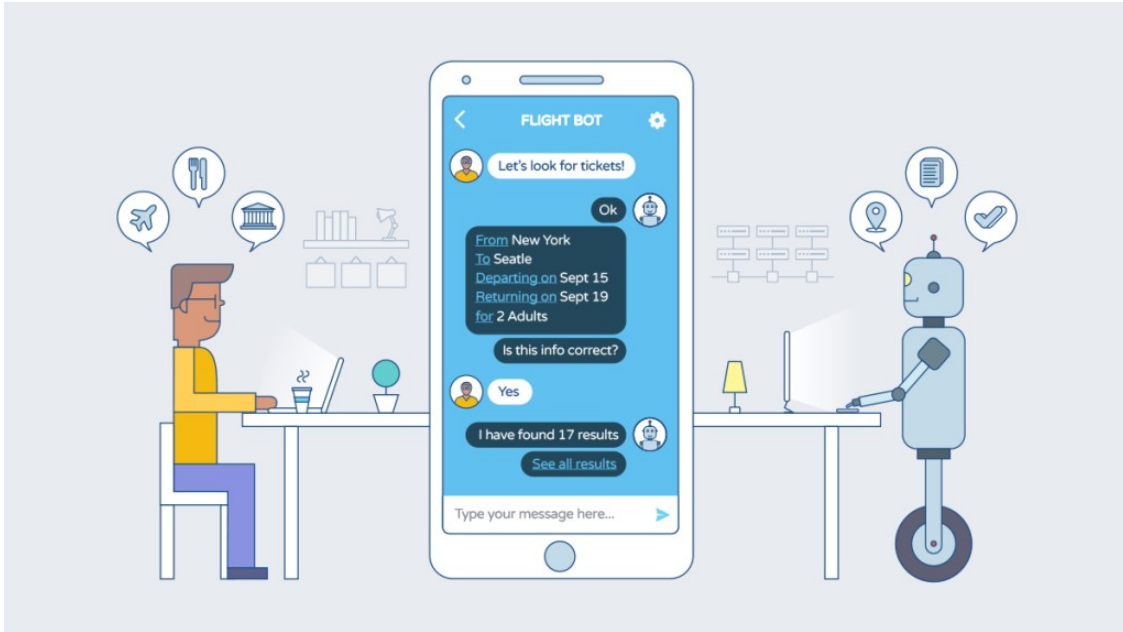


Fig.1 Application of ChatBot

1.1.1 Growth of the ChatBot

Over few year, message apps become popular than Social networking. People are using applications these days such as FB Messenger, Kik, etc.This is making other business available on message platform leads to proactive communication with user about their things. To interact on these apps with many users, the business can write a computer program that can communicate like men which is called a chatbot.

1.1.2 Natural Language Understanding Engine

The chatbot system (engine) is one of the most critical part of a chatbot, alias “**Natural Language Understanding (NLU) engine**” (Kar, R and Haldar, R. 2016). The NLU holds responsibility for the translation of dialogs to actions which are understood by the machine. NLU engines use of artificial intelligence methods to understand the natural language used in interfaces such as chatbots. These concepts are used to develop the behavior of the chatbot and how coherently it interacts with the user. Intents are used to establish a connection between the user and the action to be executed by the chatbot in order for the user to achieve their goal.

1.1.3 Challenges

The CBot system is unknown to those who are not familiar to tech. Chatbots are not much accurate in providing the answers or solutions, though there are many chatbots. Student are required to put in physical work by visiting the colleges to get their requesting information replied by institute help desk. Overall process takes a lot of money as well as time because creators can be very far from institute..

1.2 Problem Statement

Implementation of Chatbot using Tensor flow and Python.

1.3 Methodology

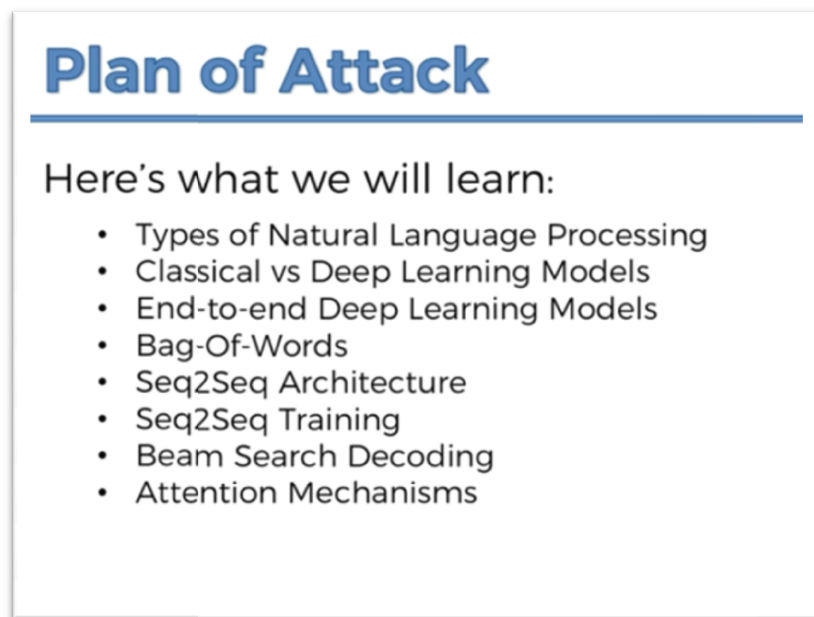


Fig.1.2 Plan of Attack

We've got quite a diverse and involved intuition session coming up. Lots of concept to grasp so it will be good for us too. At the start have a bird's eye view of the things that we're going to discuss. Here's what we will learn in this section on intuition. First of all we'll discuss about the types of natural language processing this will lay the foundation and we will discuss a Venn diagram which demonstrates this deep learning which is deep in our and shows us where the sequence to sequence what lies in that diagram will keep coming up because it will be very helpful for us to keep track of how we're progressing into the world of world of natural language processing. Then we'll talk about classical versus deep learning models. Look at some actual examples of applications of a.p. And there's also been useful just to show us that natural language processing is not limited simply to chat bots. It's actually covers a huge range of different applications and we'll see a couple of them here. There will discuss about entry into deep learning models and how they're different to enter in deep learning models and what they're

advantages and then we'll talk about the bag of words and our model. We'll see two of its variations and this will help us gradually progress to the most advanced model. **The sequence which will start discussing ahead in the secrecy was architecture. This is the model that we're actually after. This is what we're implementing in our chat.** Then we'll talk about the sequence of sequence training. So it's intentionally placed in this order. That first bit of architecture and we'll actually see it in action as if it was trained because that will help us better understand what is desired from the training and then it will be easier to cover all the training afterwards. And that's why they're in that order. Then we'll talk about search decoding which is the way that the sequence secrecy secret architecture actually comes up. Wherefore the model comes up with the outputs that we want to see you know put the different. Very subtle very interesting conceptual approach here for research decoding. And finally we'll talk about at mechanisms. So this is an additional augmentation of the seq2seq model which helps with a longer term memory for the algorithm.

1.4 Design Fundamentals of Bot

1.4.1 Types of Natural Language Processing

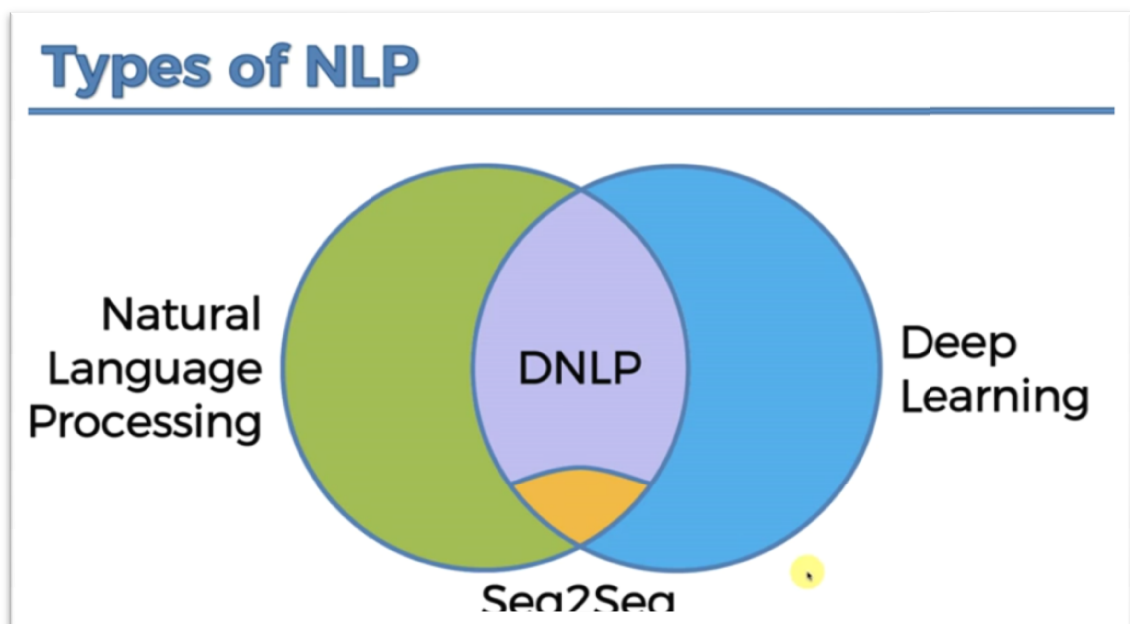


Fig.1.3 Types of NLP

So we've got two Venn diagrams here or we got a Venn diagram of two circles in it and we're going to look at the different areas of natural language processing that are going to come up in this course. So on the left of natural language processing overall and this refers to the whole circle on the left. So the reason why we've called in just this great part is because that's not overlapping part. So we know that anything here is just natural language processing. We evolved with disregard to the second circle. But natural language processing is indeed everything that is in this circle. Then we've got on the right deep learning. So these are all algorithms that have something to do with neural networks deep learning. Basically anything that's called a deeper or an algorithm falls in here. They don't have to be natural language processing. They can be classification they can be anything so they can be that's deeper here and natural language processing is any algorithm any mortal that has something to do with processing of natural language into machine terms. And then finally in the overlap we have deep and all. So these are models which have to do with natural language processing but also which are deep learning more which are neural networks. And so that's the part that we're going to be aiming for but it's also very good to have visibility of all three. We will be talking about some models that fall just in here and then we'll be talking about those here and it'll be good to compare and see how the world has changed over time and why these models are often better than these models. And the other thing to note here is that the size of these diagrams is not reflective of the importance or the volumes of these different fields so I just said circles on the same size simply because we want a visual representation of all the overlap and that these fields exist but don't take size into account. It's not to scale at all. And finally there is a another part another part of this event diagram which is very important to us and it is this part over here a sub section of the deep and Piccolo sequence to sequence so sequences sequence models of the most cutting edge the most powerful models that exist right now for natural language processing. We think we can apply them in a natural neuro machine translation we can apply them in image captioning we can apply them in speech recognition questions and answers summarization lots and lots of models so we will be looking at different ones and they will be of different types. So this map will come in handy as we go through the course and it will be popping up here and there. So I think it was very important for us to set the foundation right so that now we're ready to proceed.

1.4.2 Bags of Words

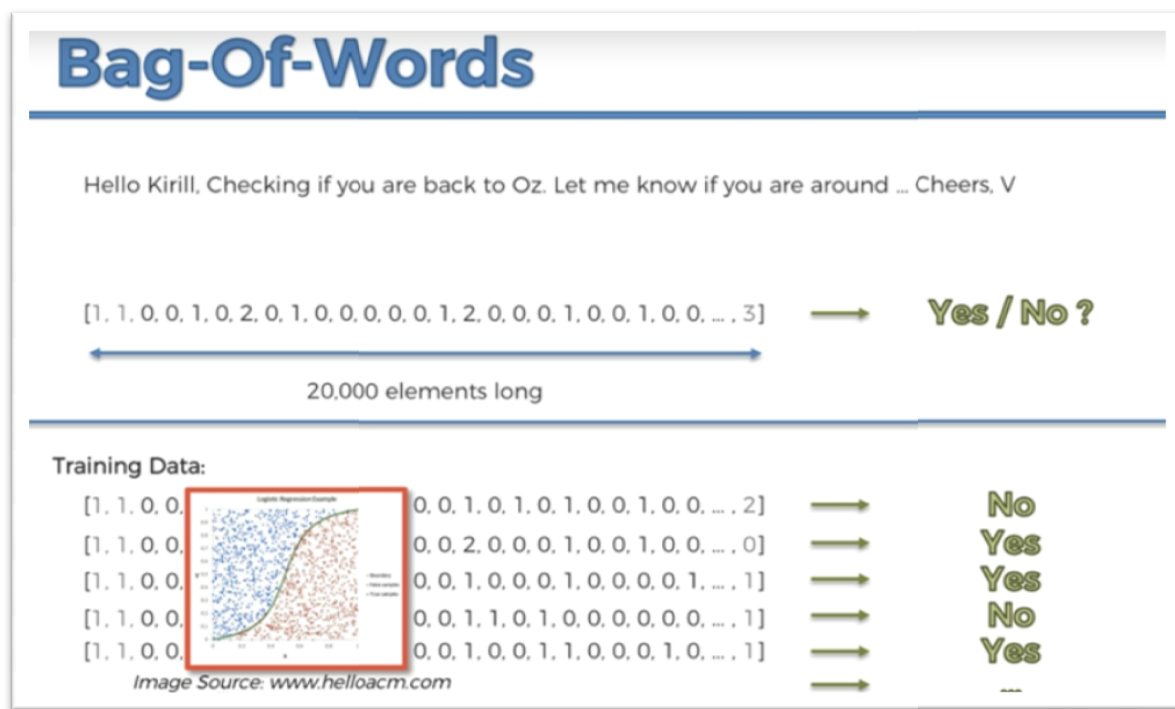


Fig.1.4 Bags of Words Model

Today we're looking at a bag of words model. First thing I'd like us to look at is an email an email I received just a few days ago. So here we go. The e-mail is about to catch up and my friend is asking. Hello Carol. Checking if you're back in Oz hands for Australia let me know if you're around and keen to see on how things are going. I definitely could use some of your creative thinking to help of mine. Cheers B. And so what I'd like us to pay attention to. First of all of course you can see that I sent this email to myself but that is because I wanted to keep my friend. Actually it is because I read your reply to the email and then I wanted to send it and also wanted to keep my friend keep his privacy but this is a real email. This is the exact text that I got literally a couple of days ago and the title was different but I just called a change to catch up and so on. What is interesting about this we're going to be looking at how we can apply natural language processing to this email in the next couple tutorials and it will help us work with a real life example and then the other thing is that it is here you can see in Google the Gmail app for phone you can see that is give me some suggestions. Very interesting it's saying I was requiring

some quick replies that I can use it can be yes I'm around. I'm back so I am not very interesting so let's keep that in mind and we will come back to this later. In the meantime text of the email is here. What can we do with it. So first things were going to start off simple. We're going to create a model. We're going to look at how we can create a model that will give us a yes no response because that's one of those questions. The question is are you back in Australia. Let me know if you're around and keen to say so. Yes. No of course it's better to have a long response. And that's the social norm and it's the etiquette to converse with people and they just say you know what even Let's try to get a yes no response and see how would you go about that because that's a first step into an LP and then further on we will see how we can expand that and more. so we're going to start off with we're vector a vector or a just like an hour a full of zeros let's call it vectors. These are like that. So just 0 0 0 0. How many zeros. Well a lot of zeros. Twenty thousand elements in total twenty thousand. Why is that? Well it's because of the way that we're building a small 20000 is the number of words that are commonly used by the average native English language speakers. So here's a quick search on Google. How many words in the English. That's the search I took I came up with how many words are there in English language. Hundred seventy one thousand pardons seventy six words. That's how many entries in our tradition plus some obsolete words plus derivative words and so on. But also people also you can see Google's giving a suggestion that more subtle adult native test takers range from 20 to 30 trade thirty five thousand words average native test takers of age eight or ten thousand words average age of test takers or you know 5000 or that it's an adult native test takers low almost white whatever this is going to is what's. But the interesting thing here is that Harmacy like what I wanted to point out. First of all 20000 and we will see why exactly resisted a lot more. What I wanted to point out is how many words are there in the English language. Even this in its own is actually Google's applying natural language processing. It's looking at what we wrote and then it's also checking other similar answers. How many words in the English language does that other person the average person know? So that's a question I ask but it came up with that. Then you came up with many other questions. So you can see that the irony is that even in this search on its own We're really falling victim of natural language processing even though that wasn't our intention and that's not what we're going to be talking about. But it's just funny that it came up anyway. So 20000 words and fun fact is that we actually use about 3000 words out of those hundred seventy one thousand four is only six words we only use 3000 words not just in conversational language

but you can see here vocabulary of just 3000 words provides coverage for around 95 percent of common texts 95 percent of common text that I like I'm assuming that's including books and stuff like that. So if you do the math it's I only use one point seventy five percent of the total number of words in English language. So as you can see even that 3000 like our 20000 is more than even the 3000 that covers any facts of the situation so we're pretty good.

2. LITERATURE SURVEY

A literature review means to evaluate and interpret available research related to a particular research question, or area. Its main aim is to present a fair research by conducting a rigorous and auditable methodology .

2.1) Title:“ Emassnuela Haaller and Traiiian Raebedea, “Designing a Chat-bot that Simulates an Historical Figure”, IEEE Conference Publications, July 2013.

There may be applications that are incorporating man-like appearance and intending to imitate human, but in most of the scenarios the information of the conversational bot is stored in a db created by someone who has prolonged knowledge in that field. However, few experts may have investigated the idea of creating a Chat Bot using an artificial character and personality beginning from web-pages or plain-text of a certain person. The paper elaborates the idea of pointing out the important facts in texts explaining the life of a ancient figure for creating an agent that is used in school scenarios.

2.2 Title: Maja Pantic, Reinier Zwitterloot, and Robbert Jan Grootjans, “Teaching Introductory Artificial Intelligence Using A simple Agent Framework”, IEEE Transactions On Education, Vol. 48, No. 3, August 2005.

The paper explain a way of teaching artificial intelligence (AI) using a genuine, naive agent frameworks only for this course. Though many agent frameworks has been proposed in the literature, none of the available structures was easy or simple to be used by to be graduates of CSE. The main objective of using such a study was to keep busy the students into which they found very interesting. A constructive approach and a traditional approach was used so that students learn very effectively.

3.SYSTEM DEVELOPMENT

3.1) SOFTWARE REQUIREMENTS:

- Anaconda
- Spyder
- Python

3.2) HARDWARE REQUIREMENTS:

- **System Requirements:**
 - ✓ CPU: 2.3 GHz Processor and above
 - ✓ RAM: 4 GB or above
 - ✓ OS: Windows 7 or above

The purpose is to model chatbot which is extremely intelligent to answer your questions and can use in any field.

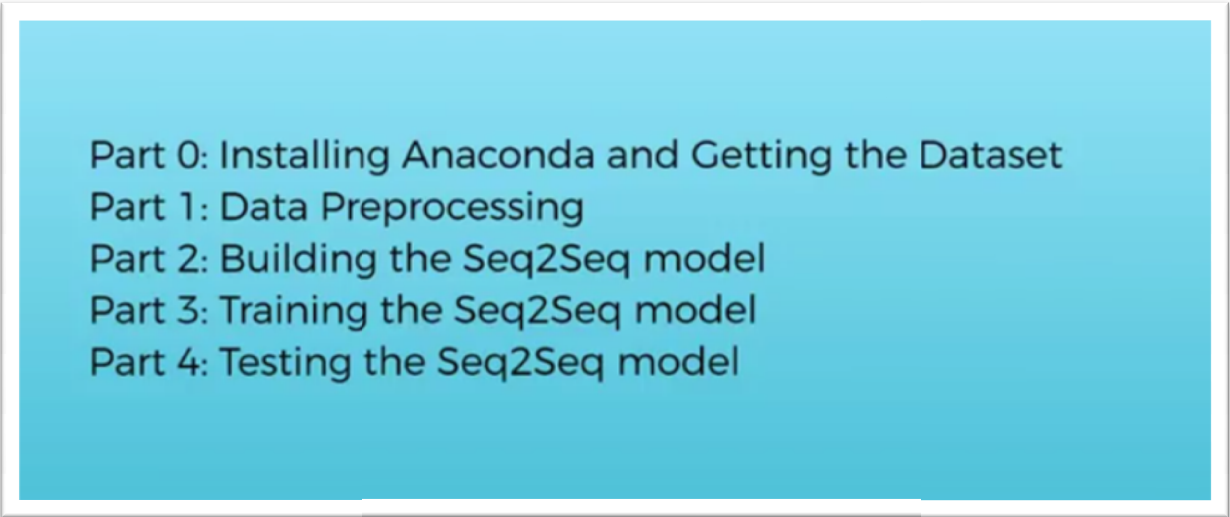
- ▣ The assumption is that in a Chatbot
 - The chatbot will be used by many users
 - It should be extremely simple
 - Can be used in any field i.e. it can be a Multi tasking Chatbot

4. ALGORITHMS

Chat Bot Algorithm which is been used in this project has been created by Machael Maudllin in 1994 and was first published in Julia. He had used this algorithm for the development of verbot which was first AI based ChatBot.

- We store our question that is asked by user in a var "queue"
- After this we fetch the main keywords from query table in database.
- Check if "queue" had any of main words.
- If there is no word, we will say no answer found.
- If there is match found all the keywords with matching keyword is fetched.
- Then we pass "queue" through 4 word process of keyword checking.
- We take its response and then submit it to the user if there is match found.
- If not match then it passes "queue" via 3-kedyword match algo
- If it is so for 2 and 1 word match.

4.1) ARCHITECTURAL FLOW:



Part 0: Installing Anaconda and Getting the Dataset
Part 1: Data Preprocessing
Part 2: Building the Seq2Seq model
Part 3: Training the Seq2Seq model
Part 4: Testing the Seq2Seq model

Fig.4.1 Architectural Flow

We'll build the chat but from scratch. And of course following a step by step approach. We will build a super powerful Chatbot by implementing a state of the art deep and LP model which will be the seq2seq and we will implement that with one of the best API is to build deep ideations or art of her own talents which will be tensorflow. We will get into the details and the high level techniques of tense of flow to build our childhood. So this implementation will be done in five parts which are the common five parts when implementing a deeper application or an AI. So here we go. First part: part zero. Installing Anaconda and getting the data set. Anaconda is the idea we will use to build our chat but we will actually use more precisely spider inside and We can it because it's like a studio. But and then we will get the data set which is the Cornell movie corpus. They said basically it's a data set of more than 600 movies containing thousands of conversations between lots of characters. And we wanted to train our Chad. But on this day he said because we wanted to build a general chat but that can have general conversation with us like a friend instead of you know specified chat but there is use for some specific purpose. That being said the model we will use can be trained on other datasets for some other purposes you know for example you will be able to train the same chat but on a more specific dataset like a calendar assistant or a navigation assistant. These are some more specific applications but this is not what we do in this course. We will try in a general chat but to talk about everyday conversations and that is why movies are because in movies you have a lot of random

conversations general conversations between friends. And so this will actually be pretty fun. Then part one. So that was part zero and part one will be data processing data processing is inevitable whenever you build an AI or whenever you build a machinery model you have to make the data set compatible with the model you're going to build. We're going to build a neural network based model and therefore the data will have to have a special format especially for the inputs. Besides we'll have to clean the text because the less we clean it and simplify it the more difficult it will be for a model to train itself to talk like a human. So we will have to do a lot of data processing this will not be the funniest part but we will try to do it the most efficiently so that we can get to the exciting parts which are actually part 2.3 and part four when we get into the heart of the model. And speaking of these parts part two will be building the seq to seq tomorrow the sequence to sequence model which is a state of the art deep an LP model. So we will build it. We will actually build a brain composed of an encoder and then a decoder and we will assemble all of them to build the final brain which be careful will not be trained yet. And speaking of training that leads us to part three part three will be about training this set to segmental that is training this brain that we would have just made in part two. So we will train it. We will set up a loss function get the optimizer and then apply some to get a gradient in the center to update the weight of the neurons of the brain so that it improves its ability to talk with us. And finally part for the last part of this implementation we will test our Chad. But that is we will test the SEC to Sec. All that is will make some kind of a code to you know once we executed have an interface where we can ask some questions and then the Chatbot will answer and we will just test the Chatbot. But by observing its answers and see how it's capable of conversing with us. So this will be pretty exciting. And here we go. That's the plan of attack of this implementation. And without waiting we are going to start with part zero directly in the next tutorial. We will install efficiently and. And then in the next tutorial after that we'll get the data set and then it will be good to go. We will start part 1 day of pre-processing and the next parts after that.

4.3 Part 1: Data Preprocessing.

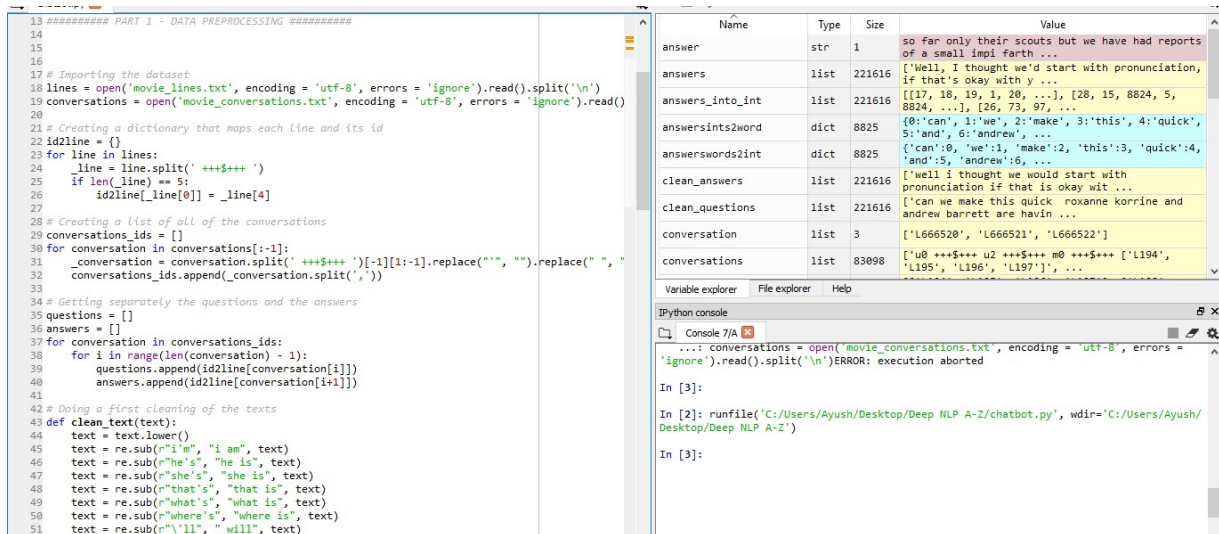


Fig.4.3 Data Pre-Processing

Libraries Imported-numpy,tensorflow,re,time

These libraries will be used to clean the text and especially to replace some characters by some more simple characters to simplify the conversation as simple as possible .Time library is imported to calculate the training time.

What we want to think in the end is a data set that contains basically two columns the input and the output because inputs will be the inputs that will be fed into the neural network and the outputs will be the target. Following steps will be performed to do so-

- 1.Importing the dataset-We start the data pre-processing journey by importing the data set and we are going to use open function to import our data set. We will load both the conversations and the lines using the open function.
2. Creating a dictionary that maps each line and its id-So where we want to go is the two words dataset composed of inputs and the outputs and the easiest way to do this is with dictionaries because we will have to keep track of the conversations to make a correct mapping between the inputs and the output.
- 3.Creating the list of all the conversations-We already have a list of conversations but we have a lot of meta data and we want to only keep what is interesting for us and what we will use for the

training and what we will only use for the training. In this we will keep ID's part of the conversation.

4. Getting separately the question and the answers-As we explained earlier we will create two columns the input and the output and so the questions will serve as the inputs and the answers will be the target. So we are going to get them separately before cleaning all the questions and the answers.

5. Cleaning of texts-This is the part where we will clean all the test of the questions and answers. Everything will be put in lowercase and we are going to remove all the apostrophes and then we will do some more cleaning by removing the non important words. We are going to make function that we are going to call clean text and that will be applied on a text like a specific question or specific answer and that will do multiple cleaning at the same time. The function will take only one argument and that will be the text.

6. Cleaning the questions and answers-We are going to apply clean text function on all the questions and the answers to effectively clean their attacks. We are going to make a new list that will contain all the clean questions, clean answers and therefore clean text function will be called on those core questions, core answers and the result will be stored in the new list.

7. Creating the dictionary that maps each words to its number of occurrence-We are going to start the whole process of removing the non so much frequent word of our corpus and we want to do that because we want to optimize the training and to optimize the training we need the essential words of the vocabulary. So we are going to remove the words that appear from time to time less than 5 percent actually of the whole corpus. The best way to do this is to create a dictionary that is going to map each word to its number of occupancies. We first need to get all the words separately and for each of these words we get the number and so that is number of times they appear in the corpus.

8. Creating two dictionaries that map the questions words and the answers words to a unique integer-In this step we are going to perform two essential tasks which are tokenization and filtering the non frequent words. We are going to do that at the same time by creating two dictionaries. One that is going to map each word of all the questions to a unique integer and one other dictionary that is going to map all the words of all the answers to a unique integer and

while creating this dictionary we will add an if condition that will check if the number of arguments is of the word. We will be dealing that if the number of arguments of the words will be higher than a certain threshold that we will choose and if that is the case we will include that word in this dictionary and if the number of arguments of the words is below the threshold we will not include this word in the dictionary so that at the same time we proceed to the process of tokenization by getting separately all the different words of all the questions and all the answers and attributing a unique integer to each of this word and at the same time we filter out the words that don't appear frequently in the dictionary that is there or below the threshold that we're going to choose.

9. Adding the last tokens to these two dictionaries-In this we are going to add the last tokens to our two previous dictionaries that we just created. And these last tokens are the ones that are useful for the encoder and the decoder in the sector like model which are you know the start of string that we're going to encode by as so as the end of string that we're going to encode by the OS the iPad which is very important for our model because the process turning data and the sequences in the batches should all have the same length and therefore we have to input this token in an empty position. And eventually we will create a last token which will call out and that corresponds to all the words that were filtered out by our two previous dictionaries. We will use these dictionaries to replace all the words all these five percent less frequent words.

10. Creating the inverse dictionary-In this one we are going to create the inverse dictionary of the answerswords2int in dictionary. That's because we will need inverse mapping from the integers to the answers words in the implementation of the sectors like model. And we actually just need for the answers words in dictionary and not the questions word.

11. Adding the end of String token to end of every answer-In this we are going to add the token end of string to the end of every answer . We are going to loop over all the answers in our clean answers. We are going to get back to the original clean answers list which is the list of all the answers that we have cleaned. One by one and to each of these answers we're going to add this as string. So we're going to simply loop over all the indexes of the clean answers list.

12. Translating all the questions and the answers into integers and replacing all the words that were filtered out by <out>-Basically we just need to translate all this into this new language of integers according to how our previous dictionary was made. And therefore we're going to have

to make some for loops to loop not only over all the questions here and then all the answers and then for each of these questions here we will loop over again all the words of the question and applying the previous dictionary to translate each of these words into their unique associated integers.

13. Sorting questions and answers by the length of questions -Basically we want to sort the questions and the answers by the length of the questions because this will speed up the training and help to reduce the loss. And the reason for this is that because it will reduce the amount of padding during the training. So it's pretty technical but it's important to know why we're doing this. And so right now we want to go what we want to do is to sort the questions and the answers by the likes of the questions to speed up the training and optimize it.

The image shows a Python IDE with two main components: a code editor on the left and a variable explorer on the right.

Code Editor: The code defines a function to clean text by removing special characters and normalizing spaces. It then iterates over a list of questions and answers, applying the cleaning function. A word count dictionary is created for both questions and answers, and a threshold is set to filter out words that appear less than 20 times. Finally, dictionaries mapping words to unique integers are created for both questions and answers.

```

52 text = re.sub(r"\\ve", " have", text)
53 text = re.sub(r"\\re", " are", text)
54 text = re.sub(r"\\d", " would", text)
55 text = re.sub(r"won't", "will not", text)
56 text = re.sub(r"can't", "cannot", text)
57 text = re.sub(r"[-()\"#/:;<{}+~|.?!]", "", text)
58 return text
59
60 # Cleaning the questions
61 clean_questions = []
62 for question in questions:
63     clean_questions.append(clean_text(question))
64
65 # Cleaning the answers
66 clean_answers = []
67 for answer in answers:
68     clean_answers.append(clean_text(answer))
69
70 # Creating a dictionary that maps each word to its number of occurrences
71 word2count = {}
72 for question in clean_questions:
73     for word in question.split():
74         if word not in word2count:
75             word2count[word] = 1
76         else:
77             word2count[word] += 1
78 for answer in clean_answers:
79     for word in answer.split():
80         if word not in word2count:
81             word2count[word] = 1
82         else:
83             word2count[word] += 1
84
85 # Creating two dictionaries that map the questions words and the answers words to a unique int
86 threshold_questions = 20
87 questionswords2int = {}
88 word_number = 0
89 for word, count in word2count.items():
90     if count >= threshold_questions:
91         questionswords2int[word] = word_number

```

Variable Explorer: This window displays the state of variables in memory. It shows a list of variables including 'answer', 'answers', 'answers_into_int', 'answersints2word', 'answerswords2int', 'clean_answers', 'clean_questions', 'conversation', and 'conversations'. Each entry shows the variable name, its type, size, and a preview of its value.

Name	Type	Size	Value
answer	str	1	so far only their scouts but we have had reports of a small impi farth ...
answers	list	221616	['Well, I thought we'd start with pronunciation, if that's okay with y ...
answers_into_int	list	221616	[[17, 18, 19, 1, 20, ...], [28, 15, 8824, 5, 8824, ...], [26, 73, 97, ...
answersints2word	dict	8825	{0:'can', 1:'we', 2:'make', 3:'this', 4:'quick', 5:'and', 6:'andrew', ...
answerswords2int	dict	8825	{'can':0, 'we':1, 'make':2, 'this':3, 'quick':4, 'and':5, 'andrew':6, ...
clean_answers	list	221616	['well i thought we would start with pronunciation if that is okay wit ...
clean_questions	list	221616	['can we make this quick roxanne korrine and andrew barrett are havin ...
conversation	list	3	['L666520', 'L666521', 'L666522']
conversations	list	83898	['\u0 +++\$+++ u2 +++\$+++ m0 +++\$+++ ['L194', 'L195', 'L196', 'L197']', ...

Python Console: The console shows the execution of the code, including an error message: 'ERROR: execution aborted'.

```

...: conversations = open('movie_conversations.txt', encoding = 'utf-8', errors =
'ignore').read().split('\n')ERROR: execution aborted

In [3]:

In [2]: runfile('C:/Users/Ayush/Desktop/Deep NLP A-2/chatbot.py', wdir='C:/Users/Ayush/Desktop/Deep NLP A-2')

In [3]:

```

Fig.4.4 Data Pre-Processing

4.3 Part 2: Building the Seq2Seq Model.

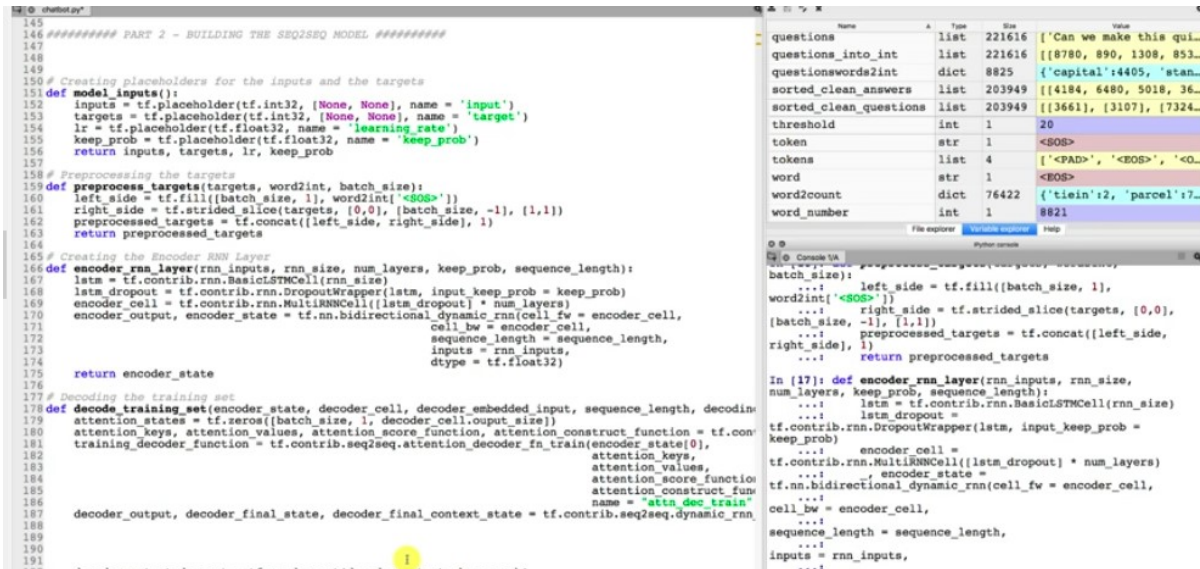


Fig.4.5 Building the Seq2Seq Model

Eventually we're getting there and that is in this part that we're going to start using tensor flow to build the architecture of the. But it is the architecture of the sectors like model which is at the heart at the core of the chatbot. So in the previous part we made a lot of for loops. We're going to make a lot of functions in this part because these are going to be the functions that will define the architecture of the sector PSEC model. So actually we're going to make one function purgatorial and the function we're going to start with right now is going to be a function that will create placeholders for the inputs and the target. So what is a placeholder and why do we need to create it. That's the first question we will answer right now. So intense a flow all variables are used in tensors. Tensors are like an advanced array more advanced than an empire array which is of a single type and that allows fastest computations in the deep neural networks. So we need to go from the number of arrays the class Signum by race to tensors. But that's not all because then in terms of flow all the variables used in tensors must be defined as what we call tensor flow placeholders. So basically that's an even more advanced data structure that can contain tensors and also other features. So that's always the first thing we must do when starting to build a deep neural network with tons of flow we need to create some placeholders for the inputs. And that's

exactly what we're going to do right now but we're going to do it through a function that we'll call model inputs. And so inside this function we will create a placeholder for the inputs a placeholder for the targets and then we will add a learning rate and even more hybrid parameters. So in short we are creating these placeholders to be able to use these variables in the future training. We need to start with them. Then we need to give a name to that function and we're going to call it model underscore inputs and this function is not going to take any arguments because it will just directly take the input and output. And for each of them convert them into placeholders into tensor flow placeholders so that's all good. so we're going to start by creating some new variable that we're going to call inputs and that will be this tensor flow placeholder containing the inputs. And so to create a sense of placeholder we'll first we need to call our sense of library which we can get things to the shortcut we made which is T.F. So we called sense of flow and from sense of flow we're going to get the place holder function that is a function that can create placeholders for the input. so now in this placeholder function we need to enter three parameters. The first one is going to be the type of the data there are the inputs and remember that we converted our input into unique integers and part 1 data preprocessing. So the type that will choose here is going to be T.F. we need to call it from tensor of flow that int 32. Then next argument is going to be the dimensions of the matrix of the input data. And since the inputs are the lists of the questions encoded into unique integers we can have a look at it again sorting questions which you must understand are the inputs. So before we start creating the encoding layer and the decoding layer we have to prepare a set of targets. Why is that? That's because the decoder will only accept a certain format of the targets. You know the decoder gets to target and in order for the targets to be accepted into the neural network of the Dakota which is an LACMA record a new network. Well the targets need to have a special format. So first of all what is exactly this format this format is twofold. First the targets must be in two batches. The record renewal network of the decoder will not accept single targets that are single answers. Remember the targets are the answers. So that's the first important thing of this special structure that we will create in this. So the targets must be in two batches. So for example if we take our sort of clean answers which are the targets Well you know the neural network will not accept them one by one like that they will only accept them into batches so instead of feeding them it will that work with single answers we will feed the neural network with batches of for example 10 answers at a time and that's the first important thing we need to understand. We need to choose a better size and

then create batches of size the bedside that we choose and these batches for example of 10 answers will go into the record renewal network of the doctor. So we'll have this first one here and then the next one after that containing the 10 next answers. And that's the first important element of the special format we want to create and then the second important element is that each of the answers in the batch of Target must start with the S O S tokens. And as you can see here the answer is start with the first word of the answer that is the first unique integer encoding the first word of the answer and not the S.O.S. So what we need to do besides creating batches of answers we need to put the S.O.S token or at least a unique integer encoding so as to go in at the beginning of each of these answers inside each batch. And that's two things we will do in this. Create the batches and then adding the S.O.S token. Hello and welcome to this new very exciting and this one we're going to start creating the architecture eventually of our set to take model. This architecture is twofold. We'll first create the encoder RNN layer because this is what comes first in the architecture of the seq2seq model and then we'll create the decoder RNN layer. So you got this right. We have the neural network for both the anchor and the decoder will be an RNN. It will not be a simple iron and rest assured it will be stacked episteme which reply drop out to improve as much as we can the accuracy and we'll create all this in the next. And today we're going to start with the encoder RNN there. And so we're going to make a function that will define in some way the architecture of this anchor RNA layer. But in this one with tensorflow we are going to include an LSTM instead of a jar. And for this we're going to use the basic LSTM Cell class by tens of flow. Hello and welcome to this new and the previous to toile we took care of the encoder and layer and then this one we're going to take care of the decoder RNN layer and we're going to have to do it in three steps. First step will be to decode the training set and that's what we'll do in the Statoil. Second step will be to decode the validation set. And that's what we'll do in the next. And then eventually we'll be ready to create the decoder RNN there. So let's take the first step in today's decoding the training set. So again we're going to make another function. This architecture is all about making functions and this function we're going to make is going to be called the decode. And those cores training on this core set that way we clearly understand what's going on. The coach training set and it is going to take a lot of arguments. So let's add them one by one. The first argument is going to be the encoder underscore state because our decoder is getting the encoded state as input as part of the inputs to proceed to the decoding. And therefore we will need the anchor to state that was returned by the previous function and

current RNA layer to proceed to the decoding. So go to state then the next argument is going to be the decoder cell. And that's basically the cell in the recurrent neural network of the decoder. Then the next argument is going to be the decoder embedded on this core input and that's basically the inputs on which were applied embedding and I'm going to explain to you right now what this means. We made to decode training said function that decoded the observations of the training set. And at the same time we turned the output of the decoder but that was for the observations of the training said that is some observations there are going back into the neural network to update the weights and improve the ability of the chatbot to talk like a human. But now we're going to make kind of the same function. But for a new kind of observations these observations are actually the observations of the test set and the validation set. And basically these are new observations that won't be used for the training and that's why as you can see here I called this code section decoding the test validation set because the function we're about to make will be used not only to predict the observations of the test in the end that is the answers of the questions we'll be asking to the chat but in the test phase part 4 remember. But then also the validation set. And where does the validation set that the set we will make during the training also. But we will do some cross-validation technique to keep separately 10 percent of the train set for cross-validation which is a technique that keeps a small part of the training data to test the predictive power on new observations. That's a very useful technique to not only reduce overfitting but also to improve the accuracy on your observations. And so the function we were about to make right now will be very similar to the one we just made only there is going to be one fundamental change.

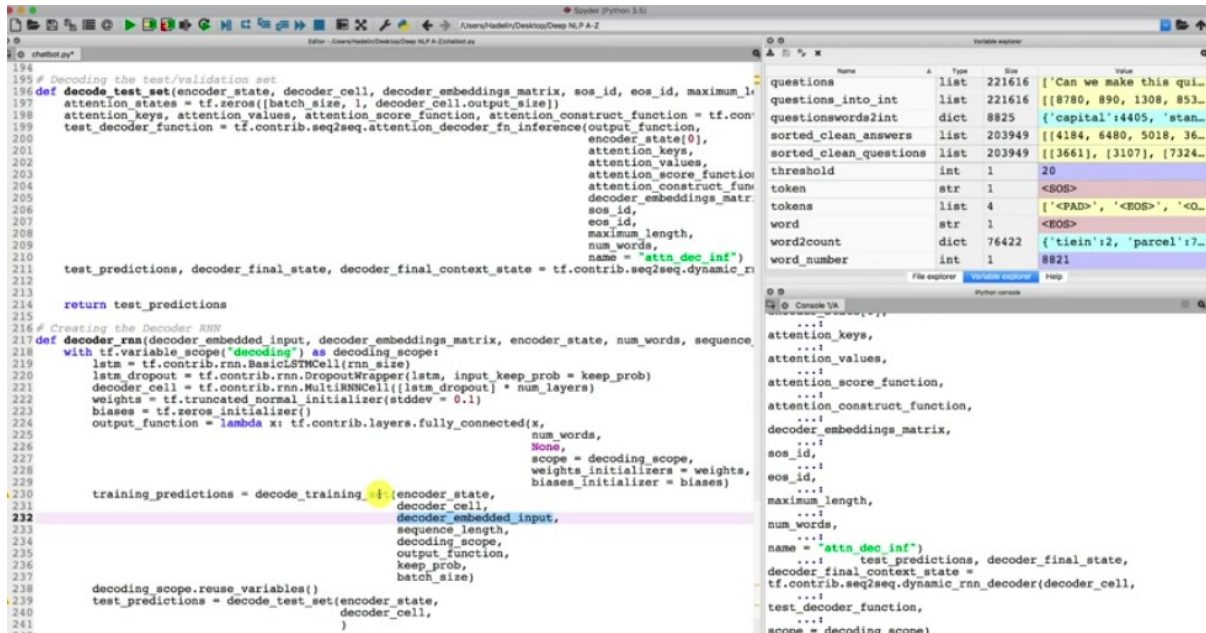


Fig.4.6 Building the Seq2Seq Model

We will create the decoder RNN as opposed to what we did before when we created the encoder RNN. Remember in this code section here we made the encoder RNN function by the way I just replaced the previous name encoder RNN layer by anchor RNN at the end of the previous tole. And so in today's Statoio we're going to make another record a new work which will be this time to record a new network of the decoder and therefore let's scroll down back. We're going to make a new function here that we're going to call decoder underscore our. And then record a neural network of the decoder. So if you're ready let's begin. So this day go to RNN function is going to take several arguments. Actually lots of them. So we're going to put them one by one quite quickly. Now that we are starting to get comfortable with all these different objects. . So the first argument is going to be the decoder underscore and underscore inputs. We already covered that. Let's move on to the next one then the next one is going to be the decoder embeddings matrix. Then the next one is going to be the and coater underscore state which I remind that a very important thing to keep in mind is that it's the output of the encoder but that becomes the input of the decoder. So in current state we will definitely need it for our decoder RNN then we're going to take again the total number of words in our corpus of answers then sequence lengths. Again we've already seen that sequence length. Then the RNN underscore size which is another one of our previous arguments then the number of layers that we called Number layers so that this time

the number of layers we want to have inside the record of a neural network of our decoder layers than what else we're going to need our word to in a dictionary. But that can actually be any dictionary it's just an argument. But that will of course be our answers words to any dictionary that we made in the previous part but one data pre-processing then we're going to apply some drop out again because there are going to be some training in the implementation of these RNN and therefore we are going to use to keep underscore prob parameter to control the dropout rate. And finally one last one. Try to guess what it is. It's very important. You know it's at the heart of any neural network it is the batch size because the inputs of neural networks whether it is in terms of flow or by torch are in two batches and we need to specify a batch size to choose the size of these batches. . So that's it for our arguments. So we're going to start with actually with the syntax of Python supersized that we're making this decode RNN in a decoding scope. Remember the decoding scope I explain that. And we think s ago it's a more advanced terms of the variable that will contain several data entities and therefore what we need to start here is introduce this code. And we're going to do it this way with a sense of flow and then from Tensorflow we're going to get that variable underscore scope. And now inside the scope we're going to make everything happen. So the first thing we need to start with is create our LSD layer. And so as for my encoder I'm going to call it L S T M equals and then exactly the same as our encoder we're going to get our sense of the library then from this library we're going to get the module then from the country model we're going to get the RNN sub module and from this RNN sub module we're going to get the basic LSD and sell the basic LACMA cell class that takes as inputs on you one argument which is the RNN size. Remember that's exactly what we did here for the encoder we created as a layer which was same an object of the basic LACMA cell class that takes us and put the r in incise. So we now have our Alice GM and so now that we have Arliss GM What do we do. Well it's time to apply some drop out regularization to reduce overfilling and improve the accuracy. And so we're going to introduce the same variable name and underscore dropout meaning an LACMA layer on which dropout is applied. And that again will be an object of another class that we used for our encoder. There it is. And so we can basically copy this because it's going to be exactly the same it is going to be an object of the drop out wrapper that needs to take the previous LACMA that we created as an object of the basic himself last and that uses the key probe parameter controlling the dropout rate. So let's do this let's that here T.F. RNN dropout rapper LACMA and put rirap equals keep. So now we have

our LACMA there with our applied but that gives us one LS gem there. RNN we're building a stacked Ellis GM and therefore now we're going to use to multiply RNN sell to stack several LACMA layers with drop out applied. And again that's exactly what we did for our encoder. So we're going to simply well copy this line and just below for our decoder RNN. Well we're going to introduce this new variable here that will be our decoder cell. I remind that the decoder cell is what contains the stacked LACMA layers with about applied. So that's basically a cell LACMA layers and therefore this occurs there is going to be the taste of what I just copied that is an object of the multi RNN cell taken from the RNN module of the country module from the tents of the library which takes as input the previews LACMA layer in which drug was applied times the number of layers we want to have in this stacked LSD layer inside the Dakota cell. Then next step the next step is actually something new something we haven't done before. We need to initialize some weights that will be associated to the neurons of the fully connected layers inside our decoder. That is the last layer and to initialize them well first we need to introduce a new variable that I'm calling waits and we're going to initialize them by using an initializer. That is the T.F. that truncated normal initializer. And that will generate a truncated normal distribution of the weights. This function takes only one argument. We're going to choose a standard deviation which will be 0.1. The standard deviation and that is equal to point 1. So now we have our fully connected the weights initialized then next step naturally after the weights the next step is to get the biases. We now have all the tools that we just need to assemble to build this final set to set moral. And that's exactly what we're going to do. And this is oil we will just use the previous tools we made like first of all of course the encoder RNN which is the first main part of the SEC to sec then the decoder RNN which is the second main part of the SEC to Sec. And as we made things to these two functions here deco turning set and Dakota set which we want to have to use again because these were just used to make the decoder RNN the decoder record new in that work. So basically we will just assemble these two and could RNN and decoder RNN and that will be our final round for this part to building the same model. So let's do it let's make this final function that we're going to call sec 2 sec underscore morrow. . It's the final ultimatum model that we will chat with during our testing phase in part 4. Basically you got it. That's the brain of our chatbot.

```

251
252 # Building the seq2seq model
253 def seq2seq_model(inputs, targets, keep_prob, batch_size, sequence_length, answers_num_words, questions_num_words):
254     encoder_embedded_input = tf.contrib.layers.embed_sequence(inputs,
255                                                             answers_num_words + 1,
256                                                             encoder_embedding_size,
257                                                             initializer = tf.random_uniform_initializer(
258                                                                 -1, 1, seed = 1234))
259     preprocessed_targets = preprocess_targets(targets, questionswords2int, batch_size)
260     decoder_embeddings_matrix = tf.Variable(tf.random_uniform([questions_num_words + 1, decoder_embedding_size],
261                                                            -1, 1, seed = 1234))
262     decoder_embedded_input = tf.nn.embedding_lookup(decoder_embeddings_matrix, preprocessed_targets)
263     training_predictions, test_predictions = decoder_rnn(decoder_embedded_input,
264                                                         decoder_embeddings_matrix,
265                                                         encoder_state,
266                                                         questions_num_words,
267                                                         sequence_length,
268                                                         rnn_size)
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299

```

Name	Type	Size	Value
questions	list	221616	['Can we make this...
questions_into_int	list	221616	[[8780, 890, 1308, ...
questionswords2int	dict	8825	{'capital':4405, 'u...
sorted_clean_answers	list	203949	[[4184, 6480, 5018, ...
sorted_clean_questions	list	203949	[[3661], [3107], [7...
threshold	int	1	20
token	str	1	<SOS>
tokens	list	4	['<PAD>', '<EOS>', ...
word	str	1	<EOS>
word2count	dict	76422	{'tiein':2, 'parcel...
word_number	int	1	8821

Fig.4.7 Building the Seq2Seq Model

4.4 Part 3: Training the Seq2Seq Model.

```

276 ##### PART 3 - TRAINING THE SEQ2SEQ MODEL #####
277
278
279 # Setting the Hyperparameters
280 epochs = 100
281 batch_size = 64
282 rnn_size = 512
283 num_layers = 3
284 encoding_embedding_size = 512
285 decoding_embedding_size = 512
286 learning_rate = 0.01
287 learning_rate_decay = 0.9
288 min_learning_rate = 0.0001
289 keep_probability = 0.5
290
291 # Defining a session
292 tf.reset_default_graph()
293 session = tf.InteractiveSession()
294
295 # Loading the model inputs
296 inputs, targets, lr, keep_prob = model_inputs()
297
298 # Setting the sequence length
299 sequence_length = tf.placeholder_with_default(25, None, name = 'sequence_length')
300
301 # Getting the shape of the inputs tensor
302 input_shape = tf.shape(inputs)
303
304 # Getting the training and test predictions
305 training_predictions, test_predictions = seq2seq_model(tf.reverse(inputs, [-1]),
306                                                       targets,
307                                                       keep_prob,
308                                                       batch_size,
309                                                       sequence_length,
310                                                       len(answerswords2int),
311                                                       len(questionswords2int),
312                                                       encoding_embedding_size,
313                                                       decoding_embedding_size,
314                                                       rnn_size,
315                                                       num_layers,
316                                                       keep_probability)
317
318
319
320
321
322

```

Name	Type	Size	Value
answer	str	1	so far only their sco...
answers	list	221616	['Well, I thought we'...
answers_into_int	list	221616	[[4324, 5558, 6397, 7...
answersinto2word	dict	8825	{0:'pickup', 1:'stand...
answerswords2int	dict	8825	{'pickup':0, 'standin...
batch_size	int	1	64
clean_answers	list	221616	['well i thought we w...
clean_questions	list	221616	['can we make this qu...
conversation	list	3	['L666520', 'L666521'...
conversations	list	83098	['u0 +++\$+++ u2 +++\$+...
conversations_ids	list	83097	[['L194', 'L195', 'L1...

Fig.4.8 Training the Seq2Seq Model

Welcome to part three training the sequence to sequence model we just built a brain in the previous part to building the sequence to sequence model Model and now it's time to train this brain and this part three to make it smart or at least able to talk able to chat with us. So for this everything's going to be fine. We're going to set some values for the hyper parameters that will be used during the training obviously and that you will be able to change if you want to experiment more and try to improve the results. So to open a session intensive flow we're going to create an object of the interactive session class and that object will be our session. But before we create this object we need to reset the dense flow graphs to ensure that the graph is ready for the training. So in general when you open intense a closed session to do some training you have to reset the graph first. So we're going to start by doing this resetting the graph and to do this we need to take the tensors of the library and then applied the reset default graph function which is a function by tensorflow that resets the tensors of TensorFlow graph. That's basically the questions that will feed into our set to SEC Network. Then we also have the targets which are the target answers the answers to the questions target. Then the third element returned by this function is the learning rate L R and finally the fourth and last elements returned by this function is the keep prob parameter keep probability and all of these four elements are the four elements returned by our model inputs function. We're going to be setting the sequence length to a maximum length which will be 25 if you remember we already did that previously and actually Part 1 of pre-processing at the end. In this sorting questions and answers by the length of the questions here we chose to go up to a certain sequence length which was 25 and this plus one wishes to include the twenty five or be bound but already by doing that we set a maximum sequence length and we have to do it again here to specify this on our sequence slings variable which is one of the arguments used by the functions of part 2. If we go to part two you see the encoder RNN takes as the argument the sequence length as well as the decode RNN right here sequence lengths. Right now our inputs are in the tensorflow tensor but we need to get the shape of these input because again this will be one of the arguments of one specific function we'll use for the training. This basic function is actually the ones that tensorflow that basically creates a tensor of ones and the dimensions of this tensor is going to be exactly the shape that's why we need to get it now is because it's going to become the argument of this tensorflow ones function will use just like what we did for this sequence length which is going to be the argument of the SEC the SEC moral function. Targets which were already loaded here by calling our modeling

function target. Then the next argument we have to input is to keep probability which is the probability of keeping the neuron activated during each iteration of the training. And we got this key parameter at line. Two hundred and ninety seven that we will later connect to the keep probability variable that we created at line two hundred and ninety and that is a specific entity of the tensor flow API. So we will connect that later. But what we have to input here in this sector signal function is make sure to understand the key variable returns by the model input function and that's it. The key probability tens of loc we will actually connect them later when running the session with the run method of our session object and that will be in the big for loop over the epochs of the training. So keep it up. Then the next one is the batch size always working with batches when doing deep learning. And we get our batch size here 64 meaning that there are going to be 64 inputs and targets in each batch size then the next one is the sequence length which we just got here. And then the next argument is going to be the length of our answers. Words 2 and dictionary. Then we are going to copy this because the next argument is going to be the land of the questions words in the dictionary. So then the next argument is going to be the encoding embedding size which we remind is the number of columns in the embeddings matrix. Each of these columns corresponding to an embedding value. And then again we are going to copy this because the next argument after that is going to be the embedding size. This time the decoding matrix the decoding embedding size. Then the next argument is going to be the RNN size which we also got here and the hyper parameters the size of 512.. And then two more arguments to go the next one is the number of layers. And that again we got when setting the hyper parameters number of layers equal 3. Meaning that there are going to be three layers in the cells of our record neural networks both the one of the encoder and the one of the decoder to numb layers. And finally the last one which is our dictionary questions words to it. So basically now we are ready to execute this code section execute to get our training protections in our test predictions. . So now we're getting closer and closer to the big loop of the training. But we're not there yet. We have a few remaining steps to do. We're going to get very close to the training because we're going to set up the lust for the optimizer and we're going to apply some great unclipping great tipping for those of you who don't know what that is. This is a technique actually some operations that will cap the gradient in the graph between a minimum value and a maximum value and that's to avoid some exploding or vanishing gradient issues. And so we're going to make sure to avoid these issues by applying gradient tipping to our optimizer. And so

we are going to define a new scope here which will contain two elements to final ultimate elements that we'll use for the training which are going to be the last error and the optimizer with grading tipping. The last error is going to be based on a weighted cross entropy loss error which is the most relevant loss to use when dealing with sequences as we are doing right now and in general when dealing with deep and LP and the optimizer that we're going to use will be first an atom optimizer which is one of the best optimizers for us to cast a great descent. And then we will apply gradiently to that optimizer to avoid exploding or vanishing great issues. And so let's do this let's define the scope. You already know how to define a scope. We did that for the decoding scope. So we need to take our sense of library and then the name scope function to which we need to input the name of this scope in quotes.

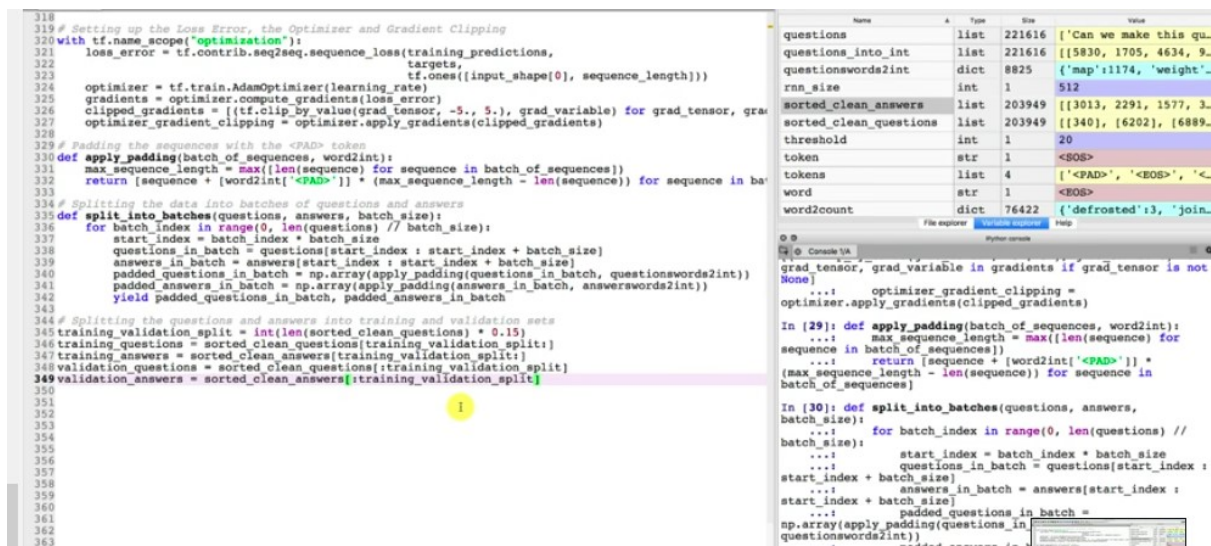


Fig.4.9 Training the Seq2Seq Model

Now we're going to finally apply the padding to the sequences with the pad token. So first we'll answer two questions first why do we have to do that? And second what are we going to do exactly with our questions and our answers. So the answer to why we have to do this is that all the sentences in a batch whether they are questions or answers must have the same length. And now the second question what we are going to do exactly. Well as you can see we have prepared here two sentences one question and one answer. And as you can see the question who are you. And the answer WE are about where WE don't have the same length. And after we apply the padding this question and answer will become this. Who are you? Bad bad bad bad. And WE are about as know. So basically the bad tokens are added so that the length of the question sequence

is equal to the length of the answer sequence. That's the purpose of padding and that's a must do in deep in LP. The two remaining steps that we have to do is first split the data into batches of questions and answers. And then the second step is to split again the questions and answers into training and validation data because we will do some cross validation during the training to keep track of the predictive power on new predictions which will be 10 or 15 percent of the data. So let's take care of this first step in this battle which is to split the data in two batches of questions and answers. That's a fundamental thing to do in deep learning to work with batches. And basically we're going to make the batches. so we're going to do that into a new function that will use the previous function because of course inside each of these batches we will pad the questions with the pad tokens so we will make a new function and then of course in the training we will apply that function to create the batches and feed the neural network with these batches of inputs and targets. That is before the big training for loop. So this final step is to split the questions and answers into the training and validation sets. So we are saying sets because we're going to make basically a set for the training questions a set for the training answers then a set for the validation questions and the set for the validation answers. And for those of you who see Krust edition for the first time well we remind that it is a technique that consists of during the training keeping 10 or 15 percent of the training data aside which won't be used to train the neural network that is which would be basically back propagate it. And just to keep track of the predictive power of the model on these observations are like new observations. So basically we're just testing the model on the side. But during the training just to keep track of what it's capable to do on new observations. And so this validation set that we have to make for both the questions and answers is exactly this 10 or 15 percent of the data on the site. So let's apply cross-validation and that consist exactly of splitting the questions and answers into four sets the training and validation sets. So we're going to start by introducing the Batch index check training class that we're going to say equal to 100. And that's because we will check the training class every 100 badges so you'll see how we'll use it later. Then We are going to keep it as viable because the second variable we will introduce is going to read a batch index check validation lots and that the validation nurse who will check it halfway and at the end of an epoch. And therefore since the middle of an epoch corresponds to half of the numbers of batches .We are going to get this half of the number of batches by taking all our questions. That is the total number of our questions. So $\text{Len training questions that We are going to divide by the batch size to get the total}$

number of batches and then We are going to divide by 2 to get half of this number of batches and then we need to use a minus one just around the down then We are going to introduce the training last error and We are going to initialize it to zero and that will be used to compute the sum of the training losses on 100 batches because we chose to check the last every 100 batches. And then we are going to copy this because the next variable that we are going to introduce is first not a total less error but a list of fewer errors and not at the training but the validation that's going to be the least of the validation plus errors. So we made some good progress here and the next step is to apply some decay to the learning rates and that will be thanks to our learning okay and therefore to do that. We are taking my learning rate and since the decay consists of multiplying the learning rate by doing red decay to reduce it by the decay rate. Well We are going to multiply this learning rate by our learning rate DK and so now remember that we want to have a minimum learning rate and therefore to make sure that we're going to add here that if condition to check if our learning rate that was reduced by the K is lower than this minimum learning rates and if that's the case if it was decayed reduced too much. Well we want to set this learning rate equal to this minimum learning rate. And that was given in the hyper parameters. Good so that's a good thing checked. Now we have to take care of early stopping. So to take care of this we're going to get first our list of validation fewer errors and to this list we're going to spend the last average validation last error that we just got. And then we're going to check if this average validation last error is lower than the minimum of all old validation plus errors that we append in the list and basically that means that if we manage to get an average validation error that is lower than all the ones in the list that means that there is improvement.

```

360 for epoch in range(1, epochs + 1):
361     for batch_index, (padded_questions_in_batch, padded_answers_in_batch) in enumerate(split_into_batches):
362         starting_time = time.time()
363         batch_training_loss_error = session.run([optimizer_gradient_clipping, loss_error], {inputs: padded_questions_in_batch,
364         targets: padded_answers_in_batch, lr: learning_rate, sequence_length: padded_answers_in_batch.get('sequence_length', 0)})
365         total_training_loss_error += batch_training_loss_error
366         ending_time = time.time()
367         batch_time = ending_time - starting_time
368         if batch_index % batch_index_check_training_loss == 0:
369             print('Epoch: {:>3}/{}'.format(epoch, epochs), Batch: '{:}>4}/{}'.format(batch_index, len(split_into_batches)), Training Loss Error: '{:}>6.3f}', Training Time on Batch: '{:}>6.3f}'.format(batch_training_loss_error, batch_time))
370
371     total_training_loss_error = 0
372     if batch_index % batch_index_check_validation_loss == 0 and batch_index > 0:
373         total_validation_loss_error = 0
374         starting_time = time.time()
375         for batch_index_validation, (padded_questions_in_batch_validation, padded_answers_in_batch_validation) in enumerate(split_into_batches_validation):
376             batch_validation_loss_error = session.run([loss_error], {inputs: padded_questions_in_batch_validation, targets: padded_answers_in_batch_validation, lr: learning_rate, sequence_length: padded_answers_in_batch_validation.get('sequence_length', 0)})
377             total_validation_loss_error += batch_validation_loss_error
378         ending_time = time.time()
379         batch_validation_time = ending_time - starting_time
380         average_validation_loss_error = total_validation_loss_error / (len(validation_questions))
381         print('Validation Loss Error: '{:}>6.3f}', Batch Validation Time: '{:}>6.3f}' seconds'.format(average_validation_loss_error, batch_validation_time))
382         if learning_rate < min_learning_rate:
383             learning_rate = min_learning_rate
384         list_validation_loss_error.append(average_validation_loss_error)
385         if early_stopping_check == 0 and min(list_validation_loss_error) < min_validation_loss_error:
386             print('I speak better now!!')
387             early_stopping_check = 1
388             saver = tf.train.Saver()
389             saver.save(session, checkpoint)
390         else:
391             print('Sorry I do not speak better, I need to practice more.')
392             early_stopping_check += 1
393             if early_stopping_check == early_stopping_stop:
394                 break
395         if early_stopping_check == early_stopping_stop:
396             print('My apologies, I cannot speak better anymore. This is the best I can do.')
397
398 In [30]: def split_into_batches(questions, answers, batch_size):
399     for batch_index in range(0, len(questions) // batch_size):
400         start_index = batch_index * batch_size
401         end_index = start_index + batch_size
402         questions_in_batch = questions[start_index : end_index]
403         answers_in_batch = answers[start_index : end_index]
404         padded_questions_in_batch = np.array([padding(question, questionswords2int) for question in questions_in_batch])
405         padded_answers_in_batch = np.array([padding(answer, answerswords2int) for answer in answers_in_batch])
406         yield padded_questions_in_batch, padded_answers_in_batch
407
408 In [31]: training_validation_split = int(len(sorted_clean_questions) * 0.15)
409 sorted_clean_questions_training_validation_split = sorted_clean_questions[:training_validation_split]
410 sorted_clean_questions_validation_validation_split = sorted_clean_questions[training_validation_split:]
411 sorted_clean_answers_training_validation_split = sorted_clean_answers[:training_validation_split]
412 sorted_clean_answers_validation_validation_split = sorted_clean_answers[training_validation_split:]

```

Fig.4.10 Training the Seq2Seq Model

4.5 Part 4: Testing the Seq2Seq Model.

```

413 ##### PART 4 - TESTING THE SEQ2SEQ MODEL #####
414
415 # Loading the weights and Running the session
416 checkpoint = './chatbot_weights.ckpt'
417 session = tf.InteractiveSession()
418 saver = tf.train.Saver()
419 saver.restore(session, checkpoint)
420
421 # Converting the questions from strings to lists of encoding integers
422 def convert_string2int(question, word2int):
423     question = clean_text(question)
424     return [word2int.get(word, word2int['<OUT>']) for word in question.split()]
425
426 # Setting up the chat
427 while(True):
428     question = input("You: ")
429     if question == 'Goodbye':
430         break
431     question = convert_string2int(question, questionswords2int)
432     question = question + [questionswords2int['<PAD>']] * (20 - len(question))
433     fake_batch = np.zeros((batch_size, 20))
434     fake_batch[0] = question
435     predicted_answer = session.run(test_predictions, {inputs: fake_batch, keep_prob: 0.5})[0]
436     answer = ''
437     for i in np.argmax(predicted_answer, 1):
438         if answersints2word[i] == 'i':
439             token = ' I'
440         elif answersints2word[i] == '<EOS>':
441             token = '.'
442         elif answersints2word[i] == '<OUT>':
443             token = 'out'
444         else:
445             token = ' ' + answersints2word[i]
446         answer += token
447         if token == '.':
448             break
449     print('ChatBot: ' + answer)
450
451
452
453
454
455
456
457
458
459
460
461

```

Fig.4.11 Testing the Seq2Seq Model

Indeed we built a brain we trained the brain and now you're about to chat with a chat .we did the most difficult part. And as you can see we've reached already the third book and we already got some weight that we can see here in File Explorer. So these are the first weights and therefore these are not the weights of some very smart brain. Or should we say of some brain with high abilities to speak with us. So we're going to have to wait some more for some better weight.. But no we will get them. We're going to make a function that will convert the questions that are right now in strings to list of encoding integers and why are we doing this. That's because we will need that function for this final step right after this to toile to set up the chat between a chat but and us. So let's make this function. We start with def. We're going to call that function converts Well let's say string to int and that function is going to take two arguments the first one is the question and the second one is word to int which is a dictionary. So it can either be the questions word in Dictionary or the answers words to any dictionary. So these two arguments. And now let's define what it has to do first before making that conversion we're going to clean everything in the question and we have a nice function for that that we made which is the clean text function. So that's why we are going to take my question which is one of my argument and We are going to clean it. Thanks to the context function. And of course this context function takes the question as input and now we can make the conversion. And to make that conversion we'll first we will return directly what we want to return. That is the question as a list of encoding integers. And therefore since we want to return a list we are going to introduce here some square brackets and we are going to use this for loop inside the list trick that you know we've done several times in this course. So there we go for. And we're going to loop over of course the word in the question. So for word in question and then remember we need to add that split with some parenthesis to get the words separately. And so what are we going to do for each of the words in the question. Well we are going to convert them and for that we're going to use our word to ant dictionary but we're going to use a trick which is to get function because the problem is that we filtered out some words that were not frequent. You know among the least frequent words we filter them out and therefore they do not exist in the dictionary but they might exist in the question and we're using just get here to make sure that if this nonfiction word is not found in the dictionary whether out token ID will be returned remember we made an out token for each of these non-African words and that's why in this get function we are going to input first the word that we're dealing with right now. And if this word doesn't exist in the dictionary. Well we'll

return the word to int dictionary of the key which is the out token. So that it returns the unique integer encoding outspoken otherwise it will get the unique integer idea including this word based on the mapping of this dictionary. And that's it. That's just a function we had to make here. We are ready to set up the chat so we'll do that in the next tutorial. And until then I'd been on the. Let's set up the chat between us and the Chatbot. So we're going to start with a while. True which will set the interaction between the Chatbot and us until we apply the break? So we're going to start by getting the question we'll be asking to our chat and this question will get it by using the input function which will take as inputs what we want to enter in the console when we ask a question to the Chatbot and therefore We are adding quote here you which will be you yourself with a colon and a space to separate you and your question. So that's the first step. We get the question then we're directly going to make that break to stop the interaction when we want to. And we are going to start the interaction when we say goodbye to Richard but that is when our question is goodbye and therefore We are going to say now that if our question is equally equal to Goodbye like that. Well in that case we are going to break and this will stop this while loop. Then now that we take care of this we are going to take care of the real conversation between the jackboot and us. So the first thing we need to do is now that we got our question that we're entering the console we're going to do all the process of getting the question in the right format so that it can go into the neural network and so that we can get the predicted answer and therefore the first thing we need to do is well first update our question this way by using the convert string to and function that takes as we define here the question and the questions word to the dictionary. That will convert the string of that question into the list of encoding integers. Great. So that's the first thing we must do then we need to apply the padding. We need to make sure that this question has the same length as the questions that were used for the training and the questions that were used for the training remember have a length of 20 and therefore now we must complete the length of that question with enough tokens and more precisely Petro China IDs so that this question has an length of 20. And to do this we'll first thing the question and updated this way by taking it again and then adding since. Now our question is we're going to do a list addition. We're going to add to our question list the following list. That is the list of the pet tokens that need to be added so that our question has a length of. And to give this you know the trick. We first take our questions words to the dictionary to get the unique integer encoding the token. And therefore here are mentoring the key which is the bad token and then we are going to

multiply this list by the difference of 20 and the length of our question. And that will complete it exactly by the number of better ideas that will make the question have a length of 20. Then next step now that we have the single question in the right format. Well remember the essential thing we need to do when working with neural networks it's the fact that this question must be into a batch the neural networks only accept batches of inputs they only accept input batches and since now this question is by itself we need to put it into a batch and we're actually going to create a fake batch which will contain this question and then some empty questions that will only get zeros. And that's why now We are introducing this fake batch variable that We are going to initialize as a number array of zeros and then zeros function we need to specify the dimensions of this vague batch and that's the only argument that has to be a couple of two elements. These two elements being the number of lines of this array and the number of columns of this array. So the number of lines is going to be of course the batch size because each line corresponds to a question and the number of columns is going to be. Well of course 20 because our new one was trained with questions of length 20 you can actually change that if you want. So that initializes this big batch but then now we need to include our question in this fake batch and to do this we will simply include it in the first row of this fake batch and therefore we are setting this first row this big batch of index 0 to be called to that question. And now that we have our batch of input in the right format Well we're ready to feed that into the new network to get the predicted answer the chat but that is the output of the neural network and this predictive answer to get it we're going to use our session. Then the run method which will take two arguments the first one is our test predictions variable that we made in part to building the second PSEC model and then remember the second argument is a dictionary of two sets of keys and values. The first one is the inputs which are of course our batch of inputs which we called fake that and then keep probability that remember we set it equal to 0.5 50 percent. So that's exactly we'll return a list of several elements. And the one that we're interested in that is the predicted answer is actually the first element of that list and therefore we need to add here some square brackets and then inside the index 0 to get that first element. And now that we get the predicted answer Well we are ready to prepare the output will get in the council so we could directly get this predicted answer but we need to post process it if we may say you know we have to predict the answer but we must turn it into a clean format meaning that will replace for example all the lower case Webuy capital eyes. You know when the Chatbot is saying for example WE are WE are not WE are not. Wherever

we will also replace all the US tokens by Dud's because that corresponds to the end of a sentence and therefore it should be a dead. Will replace all that out tokens that correspond to the filtered out words by simply ours and for the rest of the elements in that particular answer which We remind are the unique integers in the dictionary mapping the tokens to the integer IDs. Well we'll simply return the token associated to that unique integer Plus a space so that in the answer we can separate the different words by spaces. So let's do this let's introduce the final answer with an empty string like that. And then we're going to do a for loop to loop over. So here we are going to use trick non-pay tricks and get a number here and then we are going to use the ARG max function that is going to take as arguments are predicted answer and the axes which is 1. And basically what this will do is get the token IDs in the predicted answer. So I'll provide the documentation for the MAX function by name by. But what you need to understand here is that we are going to take the values of the different token IDs in the predicted answer. And so now we're going to make the replacements we mentioned starting with the lower case that we're going to replace by a capital I. So if the answer is its two word dictionary of key that is key the unique integer idea in the predicted answer is equal to lower. We won that case. We're going to replace that by capital and therefore we are going to introduce a new variable token that will be equal to capital. We then next thing we can catch and replaced by something nicer. Well first we have to use I live same we're going to take our answers in two word dictionary. And so if the token we're dealing with right now is the end of string. So can well in that case we're going to replace that by a dot incorrupt then we are just going to copy this because this is going to be the same. . So this time what do we want to take care of. Well we said that out to Ken outspoken and in that case well we said that the term is going to be out and that's it. That's the main things that we're going to replace. And we're going to finalize this with an else meaning that if we get any other token well in that case we will simply replace that token by a space plus that token we just got which is answers in two words of key I which will return the token. So we got all the conditions and now we need to increment our answer which so far is an empty string by the token and therefore We are getting here my answer incrementally by the token I sew the different tokens in the predicted answer. Clean the by everything we did here will be added one by one to the final answer we'll get in the council. Great. And finally we need to add the very important last condition which will end this follow up and we will end it by a break it's that if the token is an adult Well that means that we've reached the end of the sequence because of this condition here. You know the end is

marked by the end of string and replace that by that. Therefore if we get that then that means that it's the end of the chat. But answer. And in that case are shut but is done talking. And finally let's not forget to print the checkbook answer and to do this will specify for us that it's to chat but speaking so we are adding Chatbot here then a column and then a space to separate this Jack but string by the answer of the Chatbot. And speaking of the answer of the Chatbot. That's exactly what we're at here with plus answer which is the answer we formed here by adding each of these different tokens we can get by going through the different tokens of the predicted answer. So we're done with this code that will hopefully allow us to interact with a chaplain. And now guys congratulations not only you reached the end of the whole implementation but also you were ready to interact with a job. So now we are just going to get a little less excited because remember Archibald's was poorly trained. We just trained it while sleeping for a few hours. So Chatbot what needs to be much more trained than that and therefore what we're about to get here will be a very poor conversation. We will actually just say one thing you'll be convinced of the answer and you'll be convinced that of course we need to train the chat. But more and not only that we will also try to improve them all. We will tweak the parameters.

4.6 ChatBot Resources

ChatBot prototyping:

BotSociety and BotUI Kit are the most common apps for creating your prototypes for your bots.

Platforms: Facebook Messenger

ChatBot Analytics

Botanalytics and Bot Metrics are a conversational analytics for bots that provide fine analytics of your bot with training.

Platforms: Kik, Facebook, Slack.

ChatBot Developer Platforms

Api.AI, Wit.ai and Microsoft Bot Framework are user experience platform to build brand-unique bots, and for devices and applications it process NLP i.e Natural Language Processing.

Platforms: Facebook, Slack, Kik, LINE, Spark, Alexa, Cortana, Twitter .etc.

ChatBot Marketing

bCRM is a crm for marketing the chatbots and grow your business.

Platforms: Facebook Messenger, Slack

ChatBot Translators

Cyrano is a kind of translator that translates every language to bot to understand,thus making your bot multilingual and can be use in different languages.

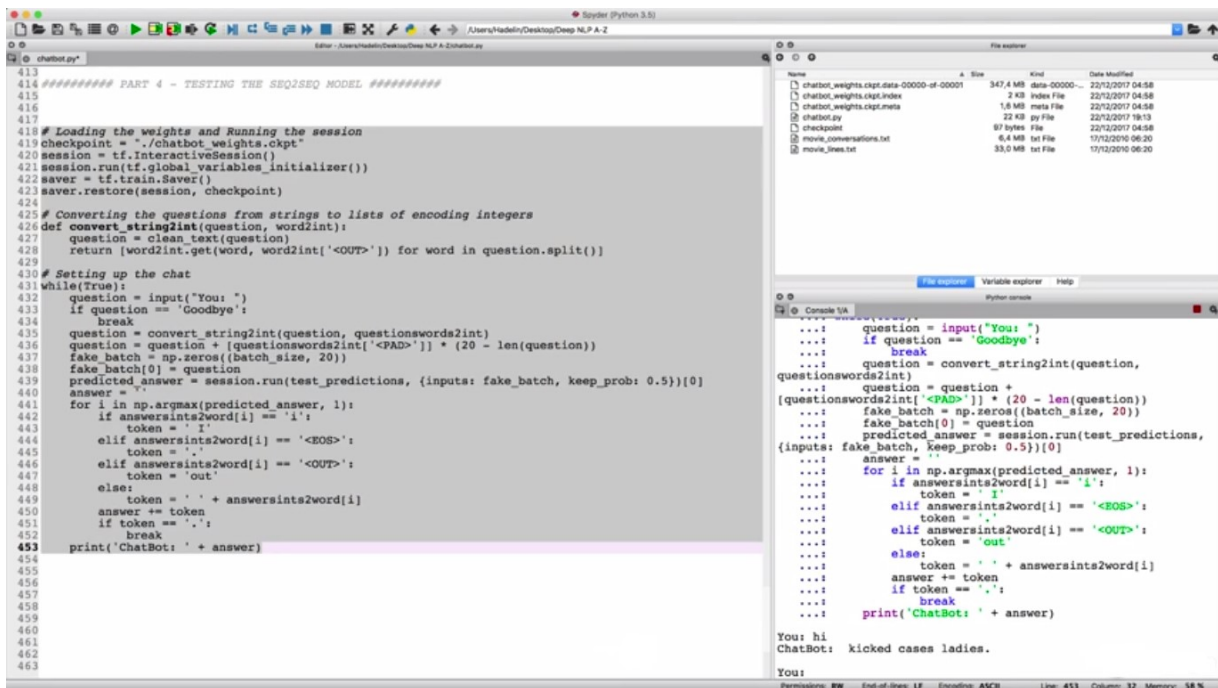
ChatBot Customer service engines

Reply.ai and Agent.ai makes your chatbot really scalable and replies just in seconds to maintain the scalablity.

Platforms: Facebook Messenger, Kik, Telegram,Slack,WE Chat, Skype, Android, IOS.

5. RESULT

The result of the chatbot that we have implemented is shown below. The Chatbot has been trained using seq2seq architecture and has been tested.



The screenshot displays a Python IDE with a code editor on the left and a console on the right. The code in the editor is for a chatbot that tests a seq2seq model. It includes a function to convert strings to integer lists and a main loop that interacts with the user. The console shows the chatbot's response to the user's input 'hi'.

```
413
414 ##### PART 4 - TESTING THE SEQ2SEQ MODEL #####
415
416
417
418 # Loading the weights and Running the session
419 checkpoint = './chatbot_weights.ckpt'
420 session = tf.InteractiveSession()
421 session.run(tf.global_variables_initializer())
422 saver = tf.train.Saver()
423 saver.restore(session, checkpoint)
424
425 # Converting the questions from strings to lists of encoding integers
426 def convert_string2int(question, word2int):
427     question = clean_text(question)
428     return [word2int.get(word, word2int['<OUT>']) for word in question.split()]
429
430 # Setting up the chat
431 while(True):
432     question = input("You: ")
433     if question == 'Goodbye':
434         break
435     question = convert_string2int(question, questionswords2int)
436     question = question + [questionswords2int['<PAD>']] * (20 - len(question))
437     fake_batch = np.zeros((batch_size, 20))
438     fake_batch[0] = question
439     predicted_answer = session.run(test_predictions, (inputs: fake_batch, keep_prob: 0.5))[0]
440     answer = ''
441     for i in np.argmax(predicted_answer, 1):
442         if answersints2word[i] == 'i':
443             token = 'i'
444         elif answersints2word[i] == '<EOS>':
445             token = '.'
446         elif answersints2word[i] == '<OUT>':
447             token = 'out'
448         else:
449             token = ' ' + answersints2word[i]
450         answer += token
451         if token == '.':
452             break
453     print('ChatBot: ' + answer)
454
455
456
457
458
459
460
461
462
463
```

```
Python console
.....
...: question = input("You: ")
...: if question == "Goodbye":
...:     break
...: question = convert_string2int(question,
questionswords2int)
...: question = question +
[questionswords2int["<PAD>"]] * (20 - len(question))
...: fake_batch = np.zeros((batch_size, 20))
...: fake_batch[0] = question
...: predicted_answer = session.run(test_predictions,
{inputs: fake_batch, keep_prob: 0.5})[0]
...: answer = ''
...: for i in np.argmax(predicted_answer, 1):
...:     if answersints2word[i] == 'i':
...:         token = 'i'
...:     elif answersints2word[i] == '<EOS>':
...:         token = '.'
...:     elif answersints2word[i] == '<OUT>':
...:         token = 'out'
...:     else:
...:         token = ' ' + answersints2word[i]
...:     answer += token
...:     if token == '.':
...:         break
...:     print('ChatBot: ' + answer)

You: hi
ChatBot: kicked cases ladies.
You:
```

CONCLUSION

The main objectives of the project were to develop an algorithm that will be used to identify answers related to user submitted questions. To develop a database where all the related data will be stored and to develop a web interface. The web interface developed had two parts, one for simple users and one for the administrator. A background research took place, which included an overview of the conversation procedure and any relevant chat bots available. A database was developed, which stores information about questions, answers, keywords, logs and feedback messages. A usable system was designed, developed and deployed to the web server on two occasions. An evaluation took place from data collected by potential students of the University. Also after received feedback from the first deployment, extra requirements were introduced and implemented

REFERENCES

- [1] Chi-Tsun Cheng, Member, IEEE, Chi K. Tse, Fellow, IEEE, and Francis C. M. Lau, Senior Member, IEEE, “A Delay-Aware Data Collection Network Structure for Wireless Sensor Networks”(2011)- IEEE SENSORS JOURNAL, VOL. 11, NO. 3, MARCH 2011
- [2] Chi-Tsun Cheng, Member, IEEE, Nuwan Ganganath, Student Member, IEEE, and Kai-Yin Fok, Student Member, IEEE, “Concurrent Data Collection Trees for IoT Applications”
- [3] Sushruta Mishra, Dept. of CSE, GEC, BBSR, India, Lambodar Jena, Dept. of CSE, GEC, BBSR, India, AArti Pradhan, Dept. of CSE, GEC, BBSR, India, “Fault Tolerance in Wireless Sensor Networks”(2012) – International Journal of Advanced Research in Computer Science and Software Engineering. Available at: <http://www.ijarcsse.com>
- [4] Michael K. Reiter, Asad Samar, Chenxi Wang, “Distributed Construction of a Fault-Tolerant Network from a Tree” - Proceedings of the 2005 24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)