CREATING A CHATBOT

OBJECTIVE: To create a Chatbot using dialogflow and python code.

ABSTRACT:

Chatbots are poised to revolutionize User Interface design.

- Chatbots, or conversational interfaces as they are also known, present a new way for individuals to interact with computer systems.
- Traditionally, to get a question answered by a software program involved using a search engine, or filling out a form.
- A chatbot allows a user to simply ask questions in the same manner that they would address a human.
- The most well known chatbots currently are voice chatbots: Alexa and Siri. However, chatbots are currently being adopted at a high rate on computer chat platforms.
- Most commercial chatbots are dependent on platforms created by the technology giants for their natural language processing.
- These include Amazon Lex, Microsoft Cognitive Services, Google Cloud Natural Language API, Facebook DeepText, and IBM Watson.
- Platforms where chatbots are deployed include Facebook Messenger, Skype, and Slack, among many others.

INTRODUCTION:

Chatbots, also known as conversational agents, are designed with the help of AI (Artificial intelligence) software. They simulate a conversation (or a chat) with users in a natural language via messaging applications, websites, mobile apps, or phone.

There are two primary ways chatbots are offered to visitors:

- Web-based applications
- Standalone applications

Chatbots represent a potential shift in how people interact with data and services online. While there is currently a surge of interest in chatbot design and development, we lack knowledge about why people use chatbots.

Here are specific steps to keep in mind for chatbot development.

1. Defined Objectives :

Chatbots today mimic human conversations. Thanks to their learning ability and 24/7 presence. They can optimize communication and create real engagement. The client must build a creative and user-friendly interface for communication. Over-burdening your chatbot with traits and crafting it to ace all undertakings will probably set you up for disappointment.

The Approach

Instead of spreading the chatbot too thin over multiple functions, it can be crafted to focus entirely on one essential command. Always keep in mind; individuals need quality, not quantity.

2. Shorter Responses:

In today’s fast-paced world, with attention spans growing shorter every second, no one has the time to read out long conversations. Jutting in complicated languages and lengthy dialogues will make the chatbot seem tedious. Though your bot is capable of handling long messages and sending responses to the user, we need a mechanism to ensure that the bot is interactive and capable of responding to diverse and yet common queries that the user might have

The Approach

When it comes to general usage, a bot should reply to the query in advance during interactions involving single-line or two-line messages. Be imaginative. Keep it basic; the bot must be clear about the next step. To achieve this, we need to train the bot to reply quicker for frequently asked messages.

3. Bot Humanization

There is a fine line between a decent bot and an incredible bot, and the latter is possible only if you give your bot a genuine identity (named as human).

It is not merely enough to pack a sequence of answers and algorithms with a human touch. Never neglect to humanize your bot, as it can leave your potential customer with mixed feelings. Users prefer to have a human conversation, irrespective of the knowledge that they are chatting with a chatbot.

The Approach

You can give a human personality to your bot with a cool title. Discover a particular and personalized name for your bot so that your users can find it easily. Additionally, educate your bot about its representation. Ensure your bot has specific information about its personification, especially when users attempt to get some info about your bot name, age, or its central goal. Ensure to keep your chatbot holistic in approach.

4. Design the conversation

Chatbot conversations are designed to attract customers. But when this is not executed correctly, it can be taxing on the customer experience. Most chatbots redirect to the Live agent quickly, wherever there are in-depth queries and conversations required. This can help retain the customers' interest.

The Approach

Conversational chatbots are now enabling you to comprehend your customer's demands better and collect more significant information to make the interaction between your bot & customer more open and easier. We need to monitor the use-cases in clients' current activities and save communication flows.

METHODOLOGY:

The technology at the core of the rise of the chatbot is natural language processing ("NLP"). Recent advances in machine learning have greatly improved the accuracy and effectiveness of natural language processing, making chatbots a viable option for many organizations. This improvement in NLP is firing a great deal of additional research which should lead to continued improvement in the effectiveness of chatbots in the years to come.

- 10 STEPS TO DEFINE CHATBOT STRATEGY

1. Define Your Goals.

   Before you develop a chatbot, you should outline your goals. Usually, companies create chatbots to drive sales using messengers, improve brand's online presence, provide users with a personal human-like assistant, or automate specific tasks such as customer support or the processing of user queries.

2. Understand Your Users.

   Understanding your users' needs, behavior, and expectations is one of the keys to success.

   If there are different user types within your brand target auditory, it's necessary to identify them all from the early start. When it's done, you can figure out who your bot interacts with and how the bot can enrich relations between these people and your brand. Classifying your audience is one of your major tasks, because such an insight will help you keep your chatbot strategy and product focused and help you deliver effective experience.

3. Learn from Competitors.

   It's important to analyze your competitive landscape when you start any project. Even though it can be tricky to use competitors as a source for inspiration in building a chatbot strategy.Chatbots' popularity is going up. But the amount of real-life examples is not enough. Nevertheless, it's a good idea to try different chatbots, regardless of the industry. Testing allows to try conversational interface and, possibly, come up with an idea how take advantage of it.

4. Pick a Platform.

   You may need to build a chatbot for more than one platform. The good news is that modern frameworks for bot creation help developers scale one chatbot for several platforms at a click. Compare it with custom development for several mobile OS and you will see a viable opportunity for saving on cost.

5. Capture Requirements.

- If it’s easy to identify the user groups for your chatbot, you can apply a standard framework for user stories. Such framework forces you to think from the user’s perspective and define a separate set of requirements for each user group. A user story has a format similar to this:

As a <type of user>, I want <action/some goal>, so that <outcome>.

- If your bot is focused on completing small tasks, but targeted at a larger audience, it is better to use jobs to be done framework. This framework is focused on the event or situation, motivation and goal, and the intended outcome. Job stories have a following format:

When a <situation>, I want to <motivation>, so I can <outcome>.

6. Prioritize Your Desires.

The process of capturing the requirements unlocks your creativity. However, you shouldn’t forget the speed to market principle. The faster you launch your chatbot, the faster you get feedback from your customers.Moreover, it’s always better to start with a small project and improve it over time, rather than to invest much before you can validate your hypothesis.

7. Consider Brand and Build Your Bot's Personality.

Chatbot is an additional way of interaction between your customer and your brand. This is why this experience must be consistent with the other elements of your brand’s style.

Why you should think about the bot’s personality and tone of voice while building a chatbot strategy? Because instead of visual interface, your bot will use conversation. In this context, the tone of voice you apply should resonate with your brand’s communication style and the expectations of your target audience.

8. Design a Conversation Flow.

First of all, you should have a proper onboarding to introduce bot’s functionality at the start of a session. It’s important to minimize user effort and build only clear and unambiguous bot messages.Every sentence your chatbot sends should to be carefully thought through. Avoid gender-specific pronouns and open-ended questions. To build a more natural conversation flow, diversify you bot’s replies as much as possible.

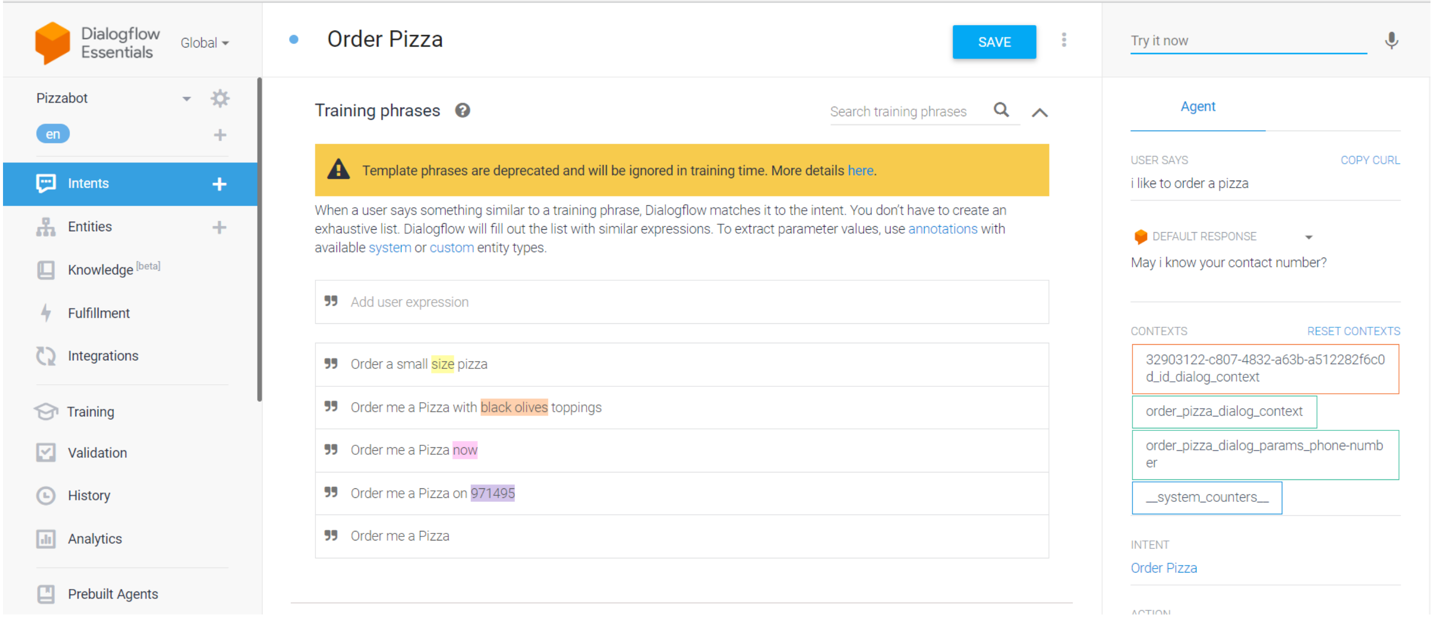
9. Select appropriate technology.

   There are 2 major categories of these tools. The first one is available do-it-yourself platforms. These tools work best for simple chatbot projects. The other one includes different NLP engines that help developers enrich their chatbots with natural language understanding features.
   DIY platforms allow to integrate a source code that will send data from your web server to your bot. This is how it can display this data to your customers

10. Take Analytics into account.

    You will want to know how good your chatbot is. To monitor its performance, you need to choose a proper tool for analytics. The tool that will help you keep an eye on the way your customers interact with the bot.

DIALOG FLOW CODE



Order Pizza

Training phrases

Search training phrases

Template phrases are deprecated and will be ignored in training time. More details here.

When a user says something similar to a training phrase, Dialogflow matches it to the intent. You don't have to create an exhaustive list. Dialogflow will fill out the list with similar expressions. To extract parameter values, use annotations with available system or custom entity types.

Add user expression

Order a small size pizza

Order me a Pizza with black olives toppings

Order me a Pizza now

Order me a Pizza on 971495

Order me a Pizza

USER SAYS

i like to order a pizza

DEFAULT RESPONSE

May i know your contact number?

CONTEXTS

32903122-c807-4832-a63b-a512282f6c0d_id_dialog_context

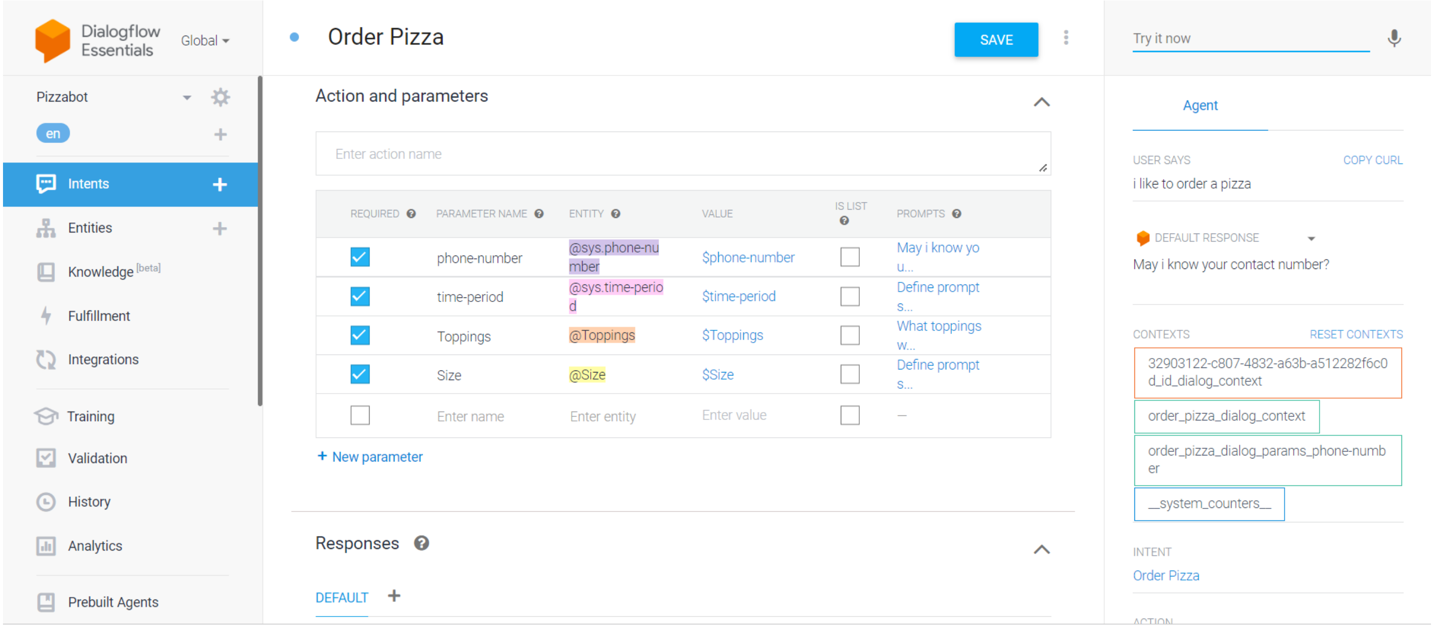
order_pizza_dialog_context

order_pizza_dialog_params_phone-number

__system_counters__

INTENT

Order Pizza



Order Pizza

Action and parameters

Enter action name

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST	PROMPTS
☑	phone-number	@sys.phone-number	$phone-number	□	May i know yo u...
☑	time-period	@sys.time-period	$time-period	□	Define prompt s...
☑	Toppings	@Toppings	$Toppings	□	What toppings w...
☑	Size	@Size	$Size	□	Define prompt s...
□	Enter name	Enter entity	Enter value	□	–

+ New parameter

Responses

DEFAULT +

USER SAYS COPY CURL

i like to order a pizza

DEFAULT RESPONSE

May i know your contact number?

CONTEXTS RESET CONTEXTS

32903122-c807-4832-a63b-a512282f6c0d_id_dialog_context

order_pizza_dialog_context

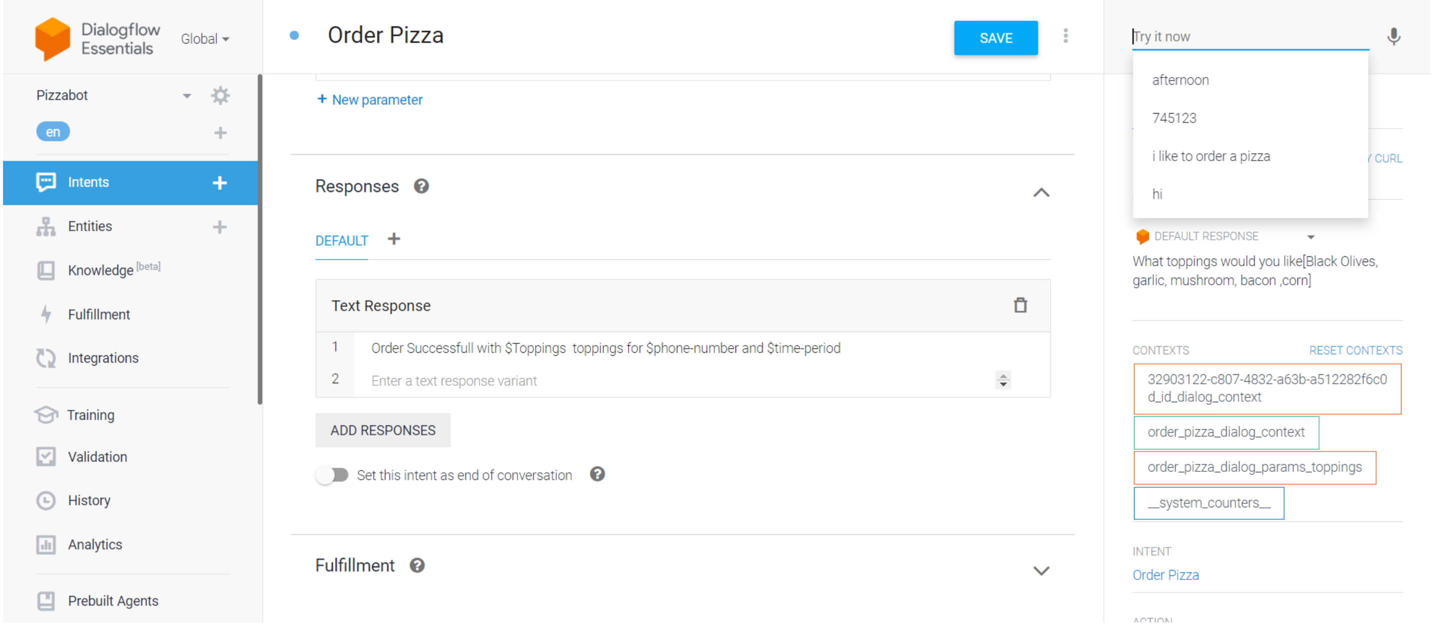
order_pizza_dialog_params_phone-number

__system_counters__

INTENT

Order Pizza

ACTION



Order Pizza

+ New parameter

Responses

DEFAULT +

Text Response

1 Order Successfull with $Toppings toppings for $phone-number and $time-period

2 Enter a text response variant

ADD RESPONSES

Set this intent as end of conversation

Fulfillment

afternoon

745123

i like to order a pizza

hi

DEFAULT RESPONSE

What toppings would you like[Black Olives, garlic, mushroom, bacon ,corn]

CONTEXTS RESET CONTEXTS

32903122-c807-4832-a63b-a512282f6c0d_id_dialog_context

order_pizza_dialog_context

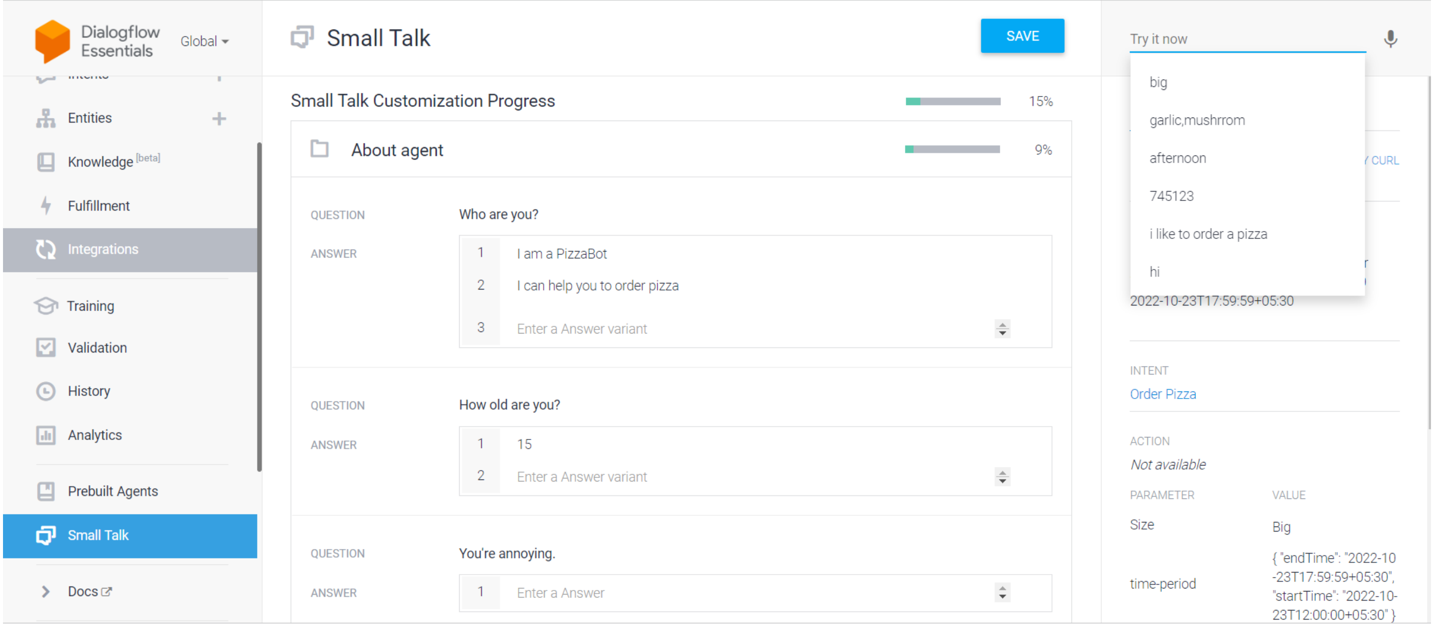
order_pizza_dialog_params_toppings

__system_counters__

INTENT

Order Pizza

ACTION



Small Talk

Small Talk Customization Progress 15%

About agent 9%

QUESTION Who are you?

ANSWER

1 I am a PizzaBot

2 I can help you to order pizza

3 Enter a Answer variant

QUESTION How old are you?

ANSWER

1 15

2 Enter a Answer variant

QUESTION You're annoying.

ANSWER

1 Enter a Answer

big

garlic,mushrrom

afternoon

745123

i like to order a pizza

hi

2022-10-23T17:59:59+05:30

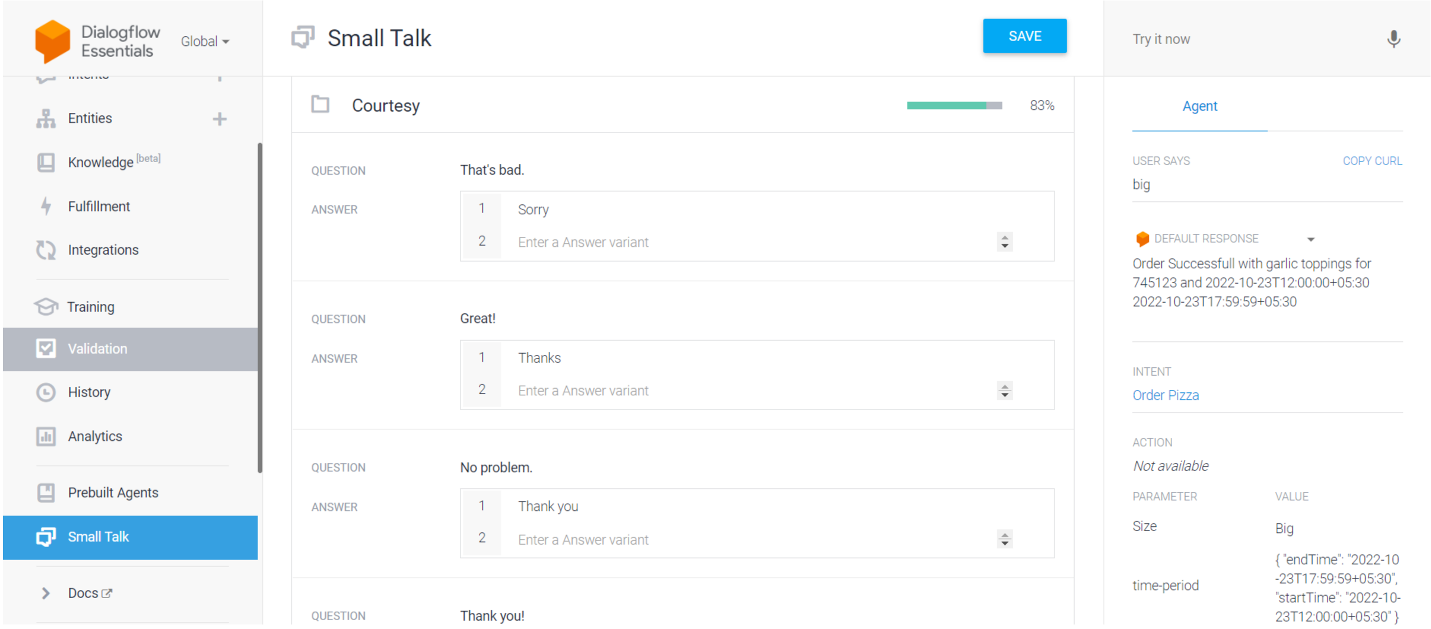
INTENT

Order Pizza

ACTION

Not available

PARAMETER	VALUE
Size	Big
time-period	{ "endTime": "2022-10-23T17:59:59+05:30", "startTime": "2022-10-23T12:00:00+05:30" }



Courtesy 83%

QUESTION That's bad.
ANSWER 1 Sorry
2 Enter a Answer variant

QUESTION Great!
ANSWER 1 Thanks
2 Enter a Answer variant

QUESTION No problem.
ANSWER 1 Thank you
2 Enter a Answer variant

QUESTION Thank you!

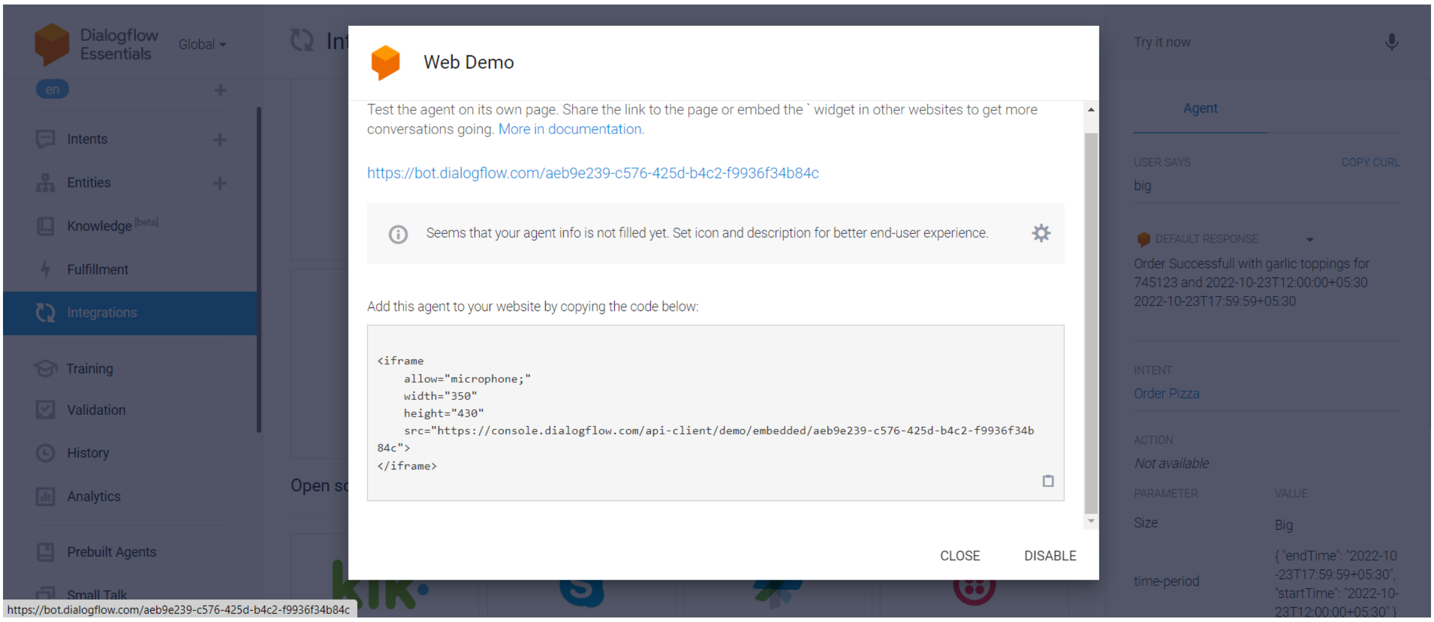
USER SAYS
big

DEFAULT RESPONSE
Order Successfull with garlic toppings for 745123 and 2022-10-23T12:00:00+05:30 2022-10-23T17:59:59+05:30

INTENT
Order Pizza

ACTION
Not available

PARAMETER	VALUE
Size	Big
time-period	{ "endTime": "2022-10-23T17:59:59+05:30", "startTime": "2022-10-23T12:00:00+05:30" }



Web Demo

Test the agent on its own page. Share the link to the page or embed the ` widget in other websites to get more conversations going. More in documentation.

https://bot.dialogflow.com/aeb9e239-c576-425d-b4c2-f9936f34b84c

Seems that your agent info is not filled yet. Set icon and description for better end-user experience.

Add this agent to your website by copying the code below:

```
<iframe
    allow="microphone;"
    width="350"
    height="430"
    src="https://console.dialogflow.com/api-client/demo/embedded/aeb9e239-c576-425d-b4c2-f9936f34b
84c">
</iframe>
```

USER SAYS
big

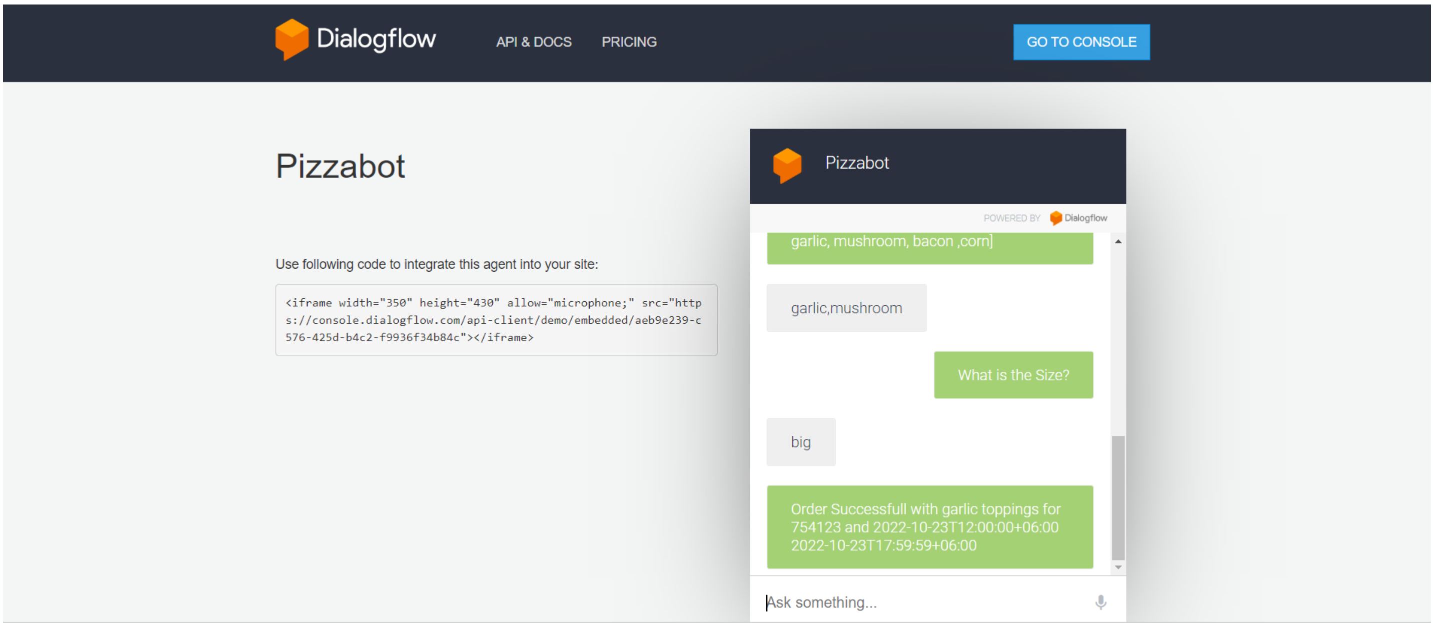
DEFAULT RESPONSE
Order Successfull with garlic toppings for 745123 and 2022-10-23T12:00:00+05:30 2022-10-23T17:59:59+05:30

INTENT
Order Pizza

ACTION
Not available

PARAMETER	VALUE
Size	Big
time-period	{ "endTime": "2022-10-23T17:59:59+05:30", "startTime": "2022-10-23T12:00:00+05:30" }

https://bot.dialogflow.com/aeb9e239-c576-425d-b4c2-f9936f34b84c



Pizzabot

Use following code to integrate this agent into your site:

```
<iframe width="350" height="430" allow="microphone;" src="http
s://console.dialogflow.com/api-client/demo/embedded/aeb9e239-c
576-425d-b4c2-f9936f34b84c"></iframe>
```

Pizzabot

garlic, mushroom, bacon ,corn]

garlic,mushroom

What is the Size?

big

Order Successfull with garlic toppings for 754123 and 2022-10-23T12:00:00+06:00 2022-10-23T17:59:59+06:00

Ask something...

PYTHON CODE

Imports necessary Libraries and Conventions

```
import pickle
import numpy as np
import matplotlib.pyplot as plt
from keras_preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer    # tokenizer
from keras.models import Sequential, Model
from keras.layers import Embedding
from keras.layers import Input, Activation,Dense, Permute, Dropout, add, dot, concatenate,
from IPython.display import Image
# from tensorflow.keras.preprocessing.sequence import pad_sequences
# from keras.utils.data_utils import pad_sequences

# open the train dataset using pickle
with open("train_qa.txt", "rb") as fp:
    train_data = pickle.load(fp)


train_data
```

```
     [(['Mary',
        'moved',
        'to',
        'the',
        'bathroom',
        '.',
        'Sandra',
        'journeyed',
        'to',
        'the',
        'bedroom',
        '.'],
       ['Is', 'Sandra', 'in', 'the', 'hallway', '?'],
       'no'),
      (['Mary',
        'moved',
        'to',
        'the',
        'bathroom',
        '.',
        'Sandra',
        'journeyed',
        'to',
        'the',
        'bedroom',
        '.',
        'Mary',
        'went',
        'back',
        'to',
        'the',
        'bedroom',
        '.',
        'Daniel',
        'went',
        'back',
        'to',
        'the',
        'hallway',
        '.'],
       ['Is', 'Daniel', 'in', 'the', 'bathroom', '?'],
       'no'),
      (['Mary',
        'moved',
        'to',
        'the',
        'bathroom',
        '.',
        'Sandra',
        'journeyed',
        'to',
        'the',
        'bedroom',
        '.',
        'Mary',
        'went',
        'back',
        'to'
```

```
# open the test dataset using pickle
with open("test_qa.txt", "rb") as fp:
    test_data = pickle.load(fp)
```

```
test_data
```

```
     [(['Mary',
        'got',
        'the',
        'milk',
        'there',
        '.',
        'John',
        'moved',
        'to',
        'the',
        'bedroom',
        '.'],
       ['Is', 'John', 'in', 'the', 'kitchen', '?'],
       'no'),
      (['Mary',
        'got',
        'the',
        'milk',
        'there',
        '.',
        'John',
        'moved',
        'to',
        'the',
        'bedroom',
        '.',
        'Mary',
        'discarded',
        'the',
        'milk',
        '.',
        'John',
        'went',
        'to',
        'the',
        'garden',
        '.'],
       ['Is', 'John', 'in', 'the', 'kitchen', '?'],
       'no'),
      (['Mary',
        'got',
        'the',
        'milk',
        'there',
        '.',
        'John',
        'moved',
        'to',
        'the',
        'bedroom',
        '.',
        'Mary',
        'discarded',
        'the',
        'milk',
        '.',
        'John',
        'went',
```

```
type(test_data)
```

```
     list
```

```
len(test_data)   # 1,000 for test
```

```
     1000
```

```
len(train_data) # 10,000 for train
```

```
     10000
```

```
train_data[0]
```

```
     (['Mary',
       'moved',
       'to',
       'the',
       'bathroom',
       '.',
       'Sandra',
       'journeyed',
       'to',
       'the',
       'bedroom',
       '.'],
      ['Is', 'Sandra', 'in', 'the', 'hallway', '?'],
      'no')
```

```
' '.join(train_data[0][0])  # our story line
```

```
     'Mary moved to the bathroom . Sandra journeyed to the bedroom .'
```

```
' '.join(train_data[0][1])  # our question line
```

```
     'Is Sandra in the hallway ?'
```

```
train_data[0][2]    # answer
```

```
     'no'
```

```
# setting vocabulary
vocab = set() # an empty set
```

```
all_data = test_data + train_data
```

```
all_data
```

```
     [(['Mary',
        'got',
        'the',
        'milk',
        'there',
        '.',
        'John',
        'moved',
        'to',
        'the',
        'bedroom',
        '.'],
       ['Is', 'John', 'in', 'the', 'kitchen', '?'],
       'no'),
      (['Mary',
        'got',
        'the',
        'milk',
        'there',
        '.',
        'John',
        'moved',
        'to',
        'the',
        'bedroom',
        '.',
        'Mary',
        'discarded',
        'the',
        'milk',
        '.',
        'John',
        'went',
        'to',
        'the',
        'garden',
        '.'],
       ['Is', 'John', 'in', 'the', 'kitchen', '?'],
       'no'),
      (['Mary',
        'got',
        'the',
        'milk',
        'there',
        '.',
        'John',
        'moved',
        'to',
        'the',
        'bedroom',
        '.',
        'Mary',
        'discarded',
        'the',
        'milk',
        '.',
        'John',
        'went',
```

```
type(all_data)
```

```
     list
```

```
for data in all_data:
    print(data)
    print("\n")
```

```
     Streaming output truncated to the last 5000 lines.
```

```
     (['Sandra', 'moved', 'to', 'the', 'bedroom', '.', 'Sandra', 'went', 'back', 'to',


     (['Sandra', 'journeyed', 'to', 'the', 'bathroom', '.', 'John', 'grabbed', 'the', '


     (['Sandra', 'journeyed', 'to', 'the', 'bathroom', '.', 'John', 'grabbed', 'the', '


     (['Sandra', 'journeyed', 'to', 'the', 'bathroom', '.', 'John', 'grabbed', 'the', '


     (['Sandra', 'journeyed', 'to', 'the', 'bathroom', '.', 'John', 'grabbed', 'the', '


     (['Sandra', 'journeyed', 'to', 'the', 'bathroom', '.', 'John', 'grabbed', 'the', '
     ([ 'John', 'moved', 'to', 'the', 'garden', '.', 'Sandra', 'journeyed', 'to', 'the',

     ([ 'John', 'moved', 'to', 'the', 'garden', '.', 'Sandra', 'journeyed', 'to', 'the',

     ([ 'John', 'moved', 'to', 'the', 'garden', '.', 'Sandra', 'journeyed', 'to', 'the',

     ([ 'John', 'moved', 'to', 'the', 'garden', '.', 'Sandra', 'journeyed', 'to', 'the',

     ([ 'John', 'moved', 'to', 'the', 'garden', '.', 'Sandra', 'journeyed', 'to', 'the',

     ([ 'John', 'moved', 'to', 'the', 'office', '.', 'Mary', 'grabbed', 'the', 'football

     ([ 'John', 'moved', 'to', 'the', 'office', '.', 'Mary', 'grabbed', 'the', 'football

     ([ 'John', 'moved', 'to', 'the', 'office', '.', 'Mary', 'grabbed', 'the', 'football

     ([ 'John', 'moved', 'to', 'the', 'office', '.', 'Mary', 'grabbed', 'the', 'football

     ([ 'John', 'moved', 'to', 'the', 'office', '.', 'Mary', 'grabbed', 'the', 'football

     ([ 'Mary', 'took', 'the', 'milk', 'there', '.', 'Mary', 'went', 'to', 'the', 'kitch

     ([ 'Mary', 'took', 'the', 'milk', 'there', '.', 'Mary', 'went', 'to', 'the', 'kitch
```

```
# consider a vocab for storing the all non repeated word from the datasets
for story, question, answer in all_data:
    vocab = vocab.union(set(story))
    vocab = vocab.union(set(question))


# add the yes and no to the vocab since the vocab contains only the story and query
vocab.add('yes')
vocab.add('no')


vocab # no word will be repeated in a set
```

```
{'.',
 '?',
 'Daniel',
 'Is',
 'John',
 'Mary',
 'Sandra',
 'apple',
 'back',
 'bathroom',
 'bedroom',
 'discarded',
 'down',
 'dropped',
 'football',
 'garden',
 'got',
 'grabbed',
 'hallway',
 'in',
 'journeyed',
 'kitchen',
 'left',
 'milk',
 'moved',
 'no',
 'office',
 'picked',
 'put',
 'the',
 'there',
 'to',
 'took',
 'travelled',
 'up',
 'went',
 'yes'}
```

```
len(vocab)
```

```
     37
```

```
vocab_len = len(vocab) + 1 # extra space to hold 0 for the keras pair sequence
vocab_len
```

```
     38
```

```
all_data
```

```
     [(['Mary',
        'got',
        'the',
        'milk',
        'there',
        '.',
        'John',
        'moved',
        'to',
        'the',
        'bedroom',
        '.'],
       ['Is', 'John', 'in', 'the', 'kitchen', '?'],
       'no'),
      (['Mary',
        'got',
        'the',
        'milk',
        'there',
        '.',
        'John',
        'moved',
        'to',
        'the',
        'bedroom',
        '.',
        'Mary',
        'discarded',
        'the',
        'milk',
        '.',
        'John',
        'went',
        'to',
        'the',
        'garden',
        '.'],
       ['Is', 'John', 'in', 'the', 'kitchen', '?'],
       'no'),
      (['Mary',
        'got',
        'the',
        'milk',
        'there',
        '.',
        'John',
        'moved',
        'to',
        'the',
        'bedroom',
        '.',
        'Mary',
        'discarded',
        'the',
        'milk',
        '.',
        'John',
        'went',
```

```
for data in all_data:
    print(len(data[0]))   # shows the length of the each story
```

```
Streaming output truncated to the last 5000 lines.
13
26
38
51
63
12
24
36
49
61
```

```
    12
    24
    35
    48
    61
    12
    24
    37
    50
    63
    12
    24
    37
    49
    63
    12
    24
    36
    48
    61
    14
    26
    38
    51
    64
    25
    37
    50
    63
    74
    23
    36
    47
    58
    70
    24
    36
    48
    59
    82
    13
    25
    37
    49
    61
    13
```

```
max([len(data[0]) for data in all_data])
```

```
    156
```

```
max_story_len = max([len(data[0]) for data in all_data])
max_story_len
```

```
    156
```

```
max_question_len = max([len(data[1]) for data in all_data])
```

```
max_question_len
```

```
     6
```

```
max_answer_len = max([len(data[2]) for data in all_data])
max_answer_len
```

```
     3
```

```
# Vectorize data means convert the data into numerical form
# convert to numerical form vectorize use keras
```

```
from keras_preprocessing.sequence import pad_sequences
# from keras.utils.data_utils import pad_sequences
from keras.preprocessing.text import Tokenizer    # tokenizer
# from tensorflow.keras.preprocessing.sequence import pad_sequences
# tokenizer divides the given data into tokens and assigns a specific value for each token
```

```
tokenizer = Tokenizer(filters = [])
```

```
tokenizer.fit_on_texts(vocab)
```

```
tokenizer.word_index    # Id's were given to each word
```

```
     {'office': 1,
      '?': 2,
      'got': 3,
      'discarded': 4,
      'apple': 5,
      'picked': 6,
      'yes': 7,
      'left': 8,
      'sandra': 9,
      'john': 10,
      'down': 11,
      'in': 12,
      'football': 13,
      'is': 14,
      'took': 15,
      'no': 16,
      'kitchen': 17,
      'there': 18,
      'moved': 19,
      'bedroom': 20,
      'daniel': 21,
      'put': 22,
      'hallway': 23,
      'garden': 24,
      'up': 25,
      'travelled': 26,
      'the': 27,
      'journeyed': 28,
      'milk': 29,
      'to': 30,
      'grabbed': 31,
      'dropped': 32,
      '.': 33,
      'mary': 34,
      'bathroom': 35,
      'went': 36,
      'back': 37}
```

```
train_story_text = []
train_question_text = []
train_answers = []
# appending the story and question to the empty train lists
for story, question, answer in train_data:
    train_story_text.append(story)
    train_question_text.append(question)
```

```
train_story_seq = tokenizer.texts_to_sequences(train_story_text)
train_story_seq
# output the text into form of sequence of tokens using tokenizer
```

```
     [[34, 19, 30, 27, 35, 33, 9, 28, 30, 27, 20, 33],
      [34,
       19,
       30,
       27,
       35,
       33,
       9,
       28,
       30,
       27,
       20,
       33,
       34,
       36,
       37,
       30,
       27,
       20,
       33,
       21,
       36,
       37,
       30,
       27,
       23,
       33],
      [34,
       19,
       30,
       27,
       35,
       33,
       9,
       28,
       30,
       27,
   20,
   33,
   34,
   36,
   37,
   30,
   27,
   20,
   33,
   21,
   36,
   37,
   30,
   27,
   23,
   33,
   9,
   36,
   30,
   27,
   17
```

```
len(train_story_text)
```

```
10000
```

```
len(train_story_seq)
```

```
10000
```

```
train_story_seq
```

```
[[34, 19, 30, 27, 35, 33, 9, 28, 30, 27, 20, 33],
 [34,
  19,
  30,
  27,
  35,
  33,
  9,
  28,
  30,
  27,
  20,
  33,
  34,
  36,
  37,
  30,
  27,
  20,
  33,
  21,
  36,
  37,
  30,
  27,
  23,
       33],
      [34,
       19,
       30,
       27,
       35,
       33,
       9,
       28,
       30,
       27,
       20,
       33,
       34,
       36,
       37,
       30,
       27,
       20,
       33,
       21,
       36,
       37,
       30,
       27,
       23,
       33,
       9,
       36,
       30,
       27,
       17,
```

```
train_story_text
```

```
     [['Mary',
       'moved',
       'to',
       'the',
       'bathroom',
       '.',
       'Sandra',
       'journeyed',
       'to',
       'the',
       'bedroom',
       '.'],
      ['Mary',
       'moved',
       'to',
       'the',
       'bathroom',
       '.',
       'Sandra',
       'journeyed',
       'to',
       'the',
       'bedroom',
       '.',
       'Mary',
       'went',
       'back',
       'to',
       'the',
       'bedroom',
       '.',
       'Daniel',
       'went',
       'back',
       'to',
       'the',
       'hallway',
       '.'],
      ['Mary',
       'moved',
       'to',
       'the',
       'bathroom',
       '.',
       'Sandra',
       'journeyed',
       'to',
       'the',
       'bedroom',
       '.',
       'Mary',
       'went',
       'back',
       'to',
       'the',
       'bedroom',
       '.',
       'Daniel',
```

```
# Separating story, query and answer

def vectorize_stories(data, word_index = tokenizer.word_index, max_story_len = max_story_l

    X = [] # story
    Xq = [] # query/question
    Y = [] # correct answer

    for story, query, answer in data:
        x = [word_index[word.lower()] for word in story]     # storing the each word from
        xq = [word_index[word.lower()] for word in query]    # storing the each word from
        y = np.zeros(len(word_index) + 1)

        y[word_index[answer]] = 1

        X.append(x)
        Xq.append(xq)
        Y.append(y)
    return(pad_sequences(X, maxlen = max_story_len), pad_sequences(Xq, maxlen = max_questi
# pad_sequence ensures that all sequence in the list have same length


inputs_train, queries_train, answers_train = vectorize_stories(train_data)
```

```
inputs_test, queries_test, answers_test = vectorize_stories(test_data)
```

```
inputs_train
```

```
array([[ 0,  0,  0, ..., 27, 20, 33],
       [ 0,  0,  0, ..., 27, 23, 33],
       [ 0,  0,  0, ..., 27, 35, 33],
       ...,
       [ 0,  0,  0, ..., 27, 20, 33],
       [ 0,  0,  0, ..., 29, 18, 33],
       [ 0,  0,  0, ...,  5, 18, 33]], dtype=int32)
```

```
queries_test
```

```
array([[14, 10, 12, 27, 17,  2],
       [14, 10, 12, 27, 17,  2],
       [14, 10, 12, 27, 24,  2],
       ...,
       [14, 34, 12, 27, 20,  2],
       [14,  9, 12, 27, 24,  2],
       [14, 34, 12, 27, 24,  2]], dtype=int32)
```

```
answers_test
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

```
tokenizer.word_index['yes']
```

```
7
```

```
tokenizer.word_index['no']
```

```
16
```

Create Model

```
from keras.models import Sequential, Model
from keras.layers import Embedding
from keras.layers import Input, Activation, Dense, Permute, Dropout, add, dot, concatenate
```

```
# initiate keras tensor
input_sequence = Input((max_story_len,))
question = Input((max_question_len,))
```

Building an m to m network

```
# input encoder m
input_encoder_m = Sequential()
input_encoder_m.add(Embedding(input_dim = vocab_len, output_dim=64))
input_encoder_m.add(Dropout(0.3))
#  Dropout randomly sets input units to 0 with a frequency of rate at each step during tra


# we need to input the sequence with size of maximum query length
input_encoder_c = Sequential()
input_encoder_c.add(Embedding(input_dim = vocab_len, output_dim = max_question_len))
input_encoder_c.add(Dropout(0.3))


# question_encoder
question_encoder = Sequential()
question_encoder.add(Embedding(input_dim = vocab_len, output_dim = 64, input_length = max_
question_encoder.add(Dropout(0.3))


# we have to input the input_sequence into the input questions
# encode the sequences
input_encoded_m = input_encoder_m(input_sequence)
input_encoded_c = input_encoder_c(input_sequence)
question_encoded = question_encoder(question)
```

Activation Function :

Activation Function activates certain artificial neurons i.e mathematical unitfor certain outputs.

Layers :

Relu activation layer : used for applying the rectified linear unit activation

Sigmoid activation layer : we use the sigmoid function

Softmax activation layer : used to implement softmax activation in the neurl networkTanh

activation function : used to implement Tanh function for neural networks.

Here we used Softmax activation function for output layer for multiclass classification.

```
sal = Image(url="softmax.png")
```

```
sal
# sal_formula = Image(url="softmax_function_formula.png")
# sal_formula
```

LOGITS :

Logits are the raw score values produce by the last layer of the neural network beforeapplying any activation function on it.

SOFTMAX Function :

SoftMax function turn logits value into probabilities by taking the exponents of eachoutput and then normalize each number by the sum of those exponents so that the entire output vector adds up to one.

softmax_function_formula.jpg

```
# now let us use the dot product to match between the first input vector and the query
# dot product
match = dot([input_encoded_m, question_encoded], axes = (2,2))
match = Activation('softmax')(match)
```

```
# add this match matrix to second input vector sequence
response = add([match, input_encoded_c])
response = Permute((2,1))(response)
# permutes the dimensions i.e 2 and 1
```

```
# concatenate
answer = concatenate([response, question_encoded])
answer
```

```
<KerasTensor: shape=(None, 6, 220) dtype=float32 (created by layer 'concatenate_1')>
```

LSTM - Long Term Short Memory

It is special type of RNN having higher capability to store the previous information

```
# apply RNN
answer = LSTM(32)(answer)
```

```
# regularize with dropouts
answer = Dropout(0.5)(answer)
answer = Dense(vocab_len)(answer)   # dense deeply connected neural network layer


answer = Activation('softmax')(answer)


# build final modelm
model = Model([input_sequence, question], answer)
model.compile(optimizer = 'rmsprop', loss = 'categorical_crossentropy', metrics = ['accura


model.summary()
```

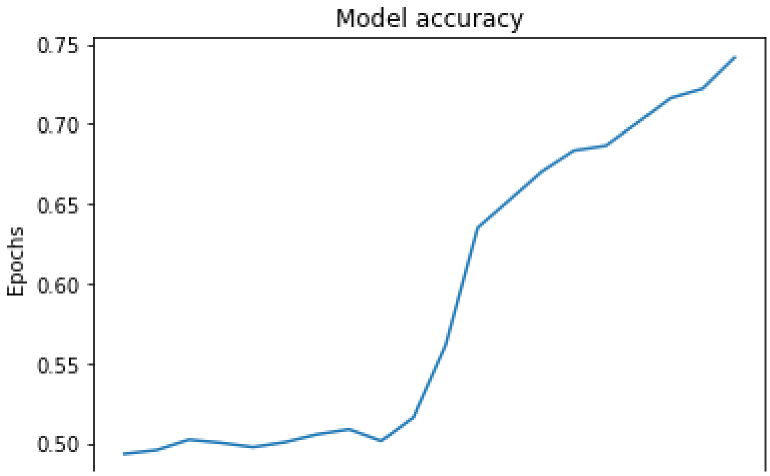
```
Model: "model_1"
__________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================
 input_3 (InputLayer)           [(None, 156)]        0           []

 input_4 (InputLayer)           [(None, 6)]          0           []

 sequential_3 (Sequential)      (None, None, 64)     2432        ['input_3[0][0]']

 sequential_5 (Sequential)      (None, 6, 64)        2432        ['input_4[0][0]']

 dot_1 (Dot)                    (None, 156, 6)       0           ['sequential_3[0][0
                                                                  'sequential_5[0][0

 activation_2 (Activation)      (None, 156, 6)       0           ['dot_1[0][0]']

 sequential_4 (Sequential)      (None, None, 6)      228         ['input_3[0][0]']

 add_1 (Add)                    (None, 156, 6)       0           ['activation_2[0][0
                                                                  'sequential_4[0][0

 permute_1 (Permute)            (None, 6, 156)       0           ['add_1[0][0]']

 concatenate_1 (Concatenate)    (None, 6, 220)       0           ['permute_1[0][0]',
                                                                  'sequential_5[0][0

 lstm_1 (LSTM)                  (None, 32)           32384       ['concatenate_1[0][

 dropout_7 (Dropout)            (None, 32)           0           ['lstm_1[0][0]']

 dense_1 (Dense)                (None, 38)           1254        ['dropout_7[0][0]']

 activation_3 (Activation)      (None, 38)           0           ['dense_1[0][0]']

==================================================================================
Total params: 38,730
Trainable params: 38,730
Non-trainable params: 0
__________________________________________________________________________________
```

```
# train model
# Epochs are the total number of iterations for training the machine learning model with a
history = model.fit([inputs_train, queries_train], answers_train, batch_size = 32, epochs
# you can take even more than 100 epochs as per your system specifications
```

```
     Epoch 1/20
     313/313 [==============================] - 13s 22ms/step - loss: 0.9643 - accuracy:
     Epoch 2/20
     313/313 [==============================] - 5s 16ms/step - loss: 0.7077 - accuracy: 0
     Epoch 3/20
     313/313 [==============================] - 5s 16ms/step - loss: 0.6969 - accuracy: 0
     Epoch 4/20
     313/313 [==============================] - 5s 16ms/step - loss: 0.6950 - accuracy: 0
     Epoch 5/20
     313/313 [==============================] - 5s 16ms/step - loss: 0.6949 - accuracy: 0
     Epoch 6/20
     313/313 [==============================] - 5s 16ms/step - loss: 0.6948 - accuracy: 0
     Epoch 7/20
     313/313 [==============================] - 5s 16ms/step - loss: 0.6943 - accuracy: 0
     Epoch 8/20
     313/313 [==============================] - 5s 16ms/step - loss: 0.6941 - accuracy: 0
     Epoch 9/20
     313/313 [==============================] - 5s 16ms/step - loss: 0.6942 - accuracy: 0
     Epoch 10/20
     313/313 [==============================] - 5s 16ms/step - loss: 0.6924 - accuracy: 0
     Epoch 11/20
     313/313 [==============================] - 5s 16ms/step - loss: 0.6808 - accuracy: 0
     Epoch 12/20
     313/313 [==============================] - 5s 16ms/step - loss: 0.6425 - accuracy: 0
     Epoch 13/20
     313/313 [==============================] - 5s 16ms/step - loss: 0.6303 - accuracy: 0
     Epoch 14/20
     313/313 [==============================] - 5s 16ms/step - loss: 0.6145 - accuracy: 0
     Epoch 15/20
     313/313 [==============================] - 5s 16ms/step - loss: 0.6040 - accuracy: 0
     Epoch 16/20
     313/313 [==============================] - 5s 16ms/step - loss: 0.5974 - accuracy: 0
     Epoch 17/20
     313/313 [==============================] - 8s 24ms/step - loss: 0.5840 - accuracy: 0
     Epoch 18/20
     313/313 [==============================] - 5s 16ms/step - loss: 0.5696 - accuracy: 0
     Epoch 19/20
     313/313 [==============================] - 5s 16ms/step - loss: 0.5576 - accuracy: 0
     Epoch 20/20
     313/313 [==============================] - 5s 16ms/step - loss: 0.5313 - accuracy: 0
```

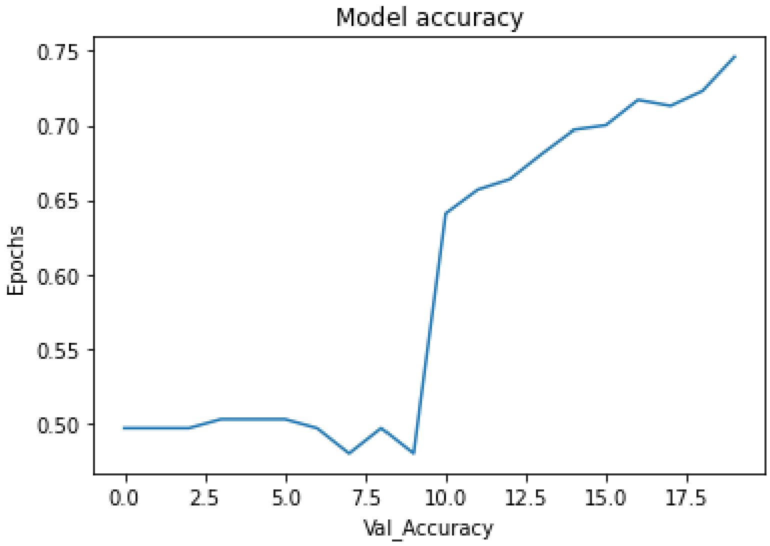
```
# valuating the model
import matplotlib.pyplot as plt
print(history.history.keys())
plt.plot(history.history['accuracy'])
plt.title("Model accuracy")
plt.xlabel("Accuracy")
plt.ylabel("Epochs")
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
Text(0, 0.5, 'Epochs')
```



```
plt.plot(history.history['val_accuracy'])
plt.title("Model accuracy")
plt.xlabel("Val_Accuracy")
plt.ylabel("Epochs")
```

```
Text(0, 0.5, 'Epochs')
```

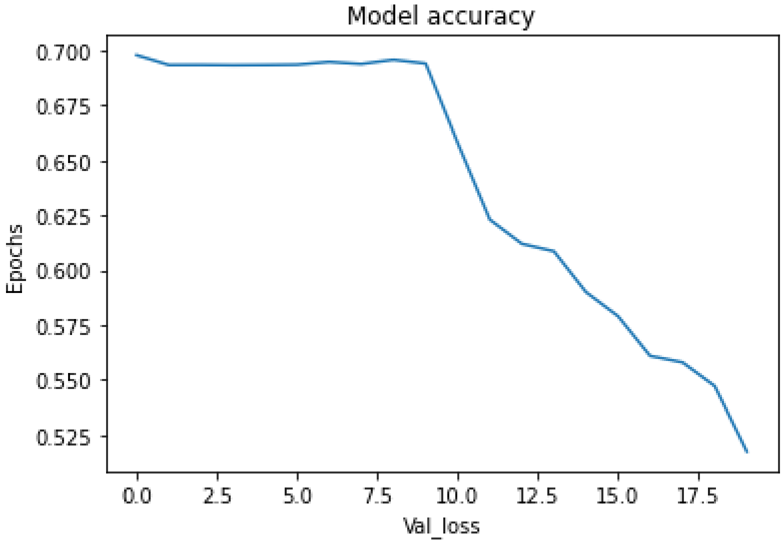


```
plt.plot(history.history['loss'])
plt.title("Model accuracy")
plt.xlabel("loss")
plt.ylabel("Epochs")
```

```
Text(0, 0.5, 'Epochs')
```

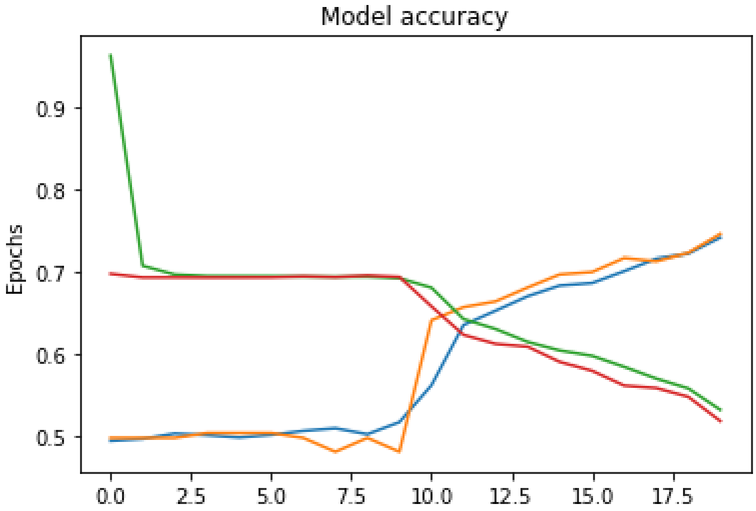
```
plt.plot(history.history['val_loss'])
plt.title("Model accuracy")
plt.xlabel("Val_loss")
plt.ylabel("Epochs")
```

```
Text(0, 0.5, 'Epochs')
```



```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("Model accuracy")
plt.ylabel("Epochs")
```

```
Text(0, 0.5, 'Epochs')
```



```
# Save Model
model.save('chatbot_model')
```

```
WARNING:absl:Found untraced functions such as lstm_cell_1_layer_call_fn, lstm_cell_1
```

```
# Evaluation on test set
model.load_weights('chatbot_model')
```

```
    <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7ff7fed43dd0>
```

```
pred_results = model.predict((inputs_test, queries_test))
```

```
    32/32 [==============================] - 1s 5ms/step
```

```
test_data[0][0]
```

```
    ['Mary',
     'got',
     'the',
     'milk',
     'there',
     '.',
     'John',
     'moved',
     'to',
     'the',
     'bedroom',
     '.']
```

1

```
story_1 = ' '.join(word for word in test_data[0][0])
story_1
```

```
    'Mary got the milk there . John moved to the bedroom .'
```

```
query_1 = ' '.join(word for word in test_data[0][1])
query_1
```

```
    'Is John in the kitchen ?'
```

```
test_data[0][2]
```

```
    'no'
```

10

```
story_10 = ' '.join(word for word in test_data[10][0])
story_10
```

```
'John moved to the hallway . Sandra went to the bedroom .'
```

```
query_10 = ' '.join(word for word in test_data[10][1])
query_10
```

```
'Is John in the hallway ?'
```

```
test_data[10][2]
```

```
'yes'
```

13

```
story_13 = ' '.join(word for word in test_data[13][0])
story_13
```

```
'John moved to the hallway . Sandra went to the bedroom . Sandra travelled to the ga
rden . John got the football there . Daniel went back to the bedroom . Mary moved to
the bathroom   Mary went to the kitchen   Sandra went to the hallway   '
```

```
query_13 = ' '.join(word for word in test_data[13][1])
query_13
```

```
'Is Mary in the kitchen ?'
```

```
test_data[13][2]
```

```
'yes'
```

```
# generate predictions from model
val_max = np.argmax(pred_results[1])
for key, val in tokenizer.word_index.items():
    if val == val_max:
        k = key
print("Predicted answer : ", k)
print("Probability of certainity :", pred_results[1][val_max])
```

```
Predicted answer :  no
Probability of certainity : 0.7065056
```

```
# create story by ourselves
vocab
```

```
{'.',
 '?',
 'Daniel',
 'Is',
 'John',
 'Mary',
 'Sandra',
 'apple',
 'back',
 'bathroom',
 'bedroom',
 'discarded',
 'down',
 'dropped',
 'football',
 'garden',
 'got',
 'grabbed',
 'hallway',
 'in',
 'journeyed',
 'kitchen',
 'left',
 'milk',
 'moved',
 'no',
 'office',
 'picked',
 'put',
 'the',
 'there',
 'to',
 'took',
 'travelled',
 'up',
 'went',
 'yes'}
```

Prediction_1

```
story = "Mary dropped football . Sandra discarded apple in kitchen . Daniel went to office
story.split()
```

```
     ['Mary',
      'dropped',
      'football',
      '.',
      'Sandra',
      'discarded',
      'apple',
      'in',
      'kitchen',
      '.',
      'Daniel',
      'went',
      'to',
      'office']
```

```
my_question = "Is Sandra in the kitchen ? "
my_question.split()
```

```
     ['Is', 'Sandra', 'in', 'the', 'kitchen', '?']
```

```
mydata = [(story.split(), my_question.split(), 'yes')]
```

```
my_story, my_question, my_answer = vectorize_stories(mydata)
```

```
pred_results = model.predict(([my_story, my_question]))
```

```
1/1 [==============================] - 0s 31ms/step
```

```
# generate predictions from model
val_max = np.argmax(pred_results[0])
for key, val in tokenizer.word_index.items():
    if val == val_max:
        k = key

print("Predicted answer : ", k)
print("Probability of certainity : ", pred_results[0][val_max])
```

```
Predicted answer :  yes
Probability of certainity :  0.5964135
```

Prediction_2

```
story = "John dropped football . Mary went to office "
story.split()
```

```
['John', 'dropped', 'football', '.', 'Mary', 'went', 'to', 'office']
```

```
my_question = "Is Mary in the office ? "
my_question.split()
```

```
['Is', 'Mary', 'in', 'the', 'office', '?']
```

```
mydata = [(story.split(), my_question.split(), 'yes')]
```

```
my_story, my_question, my_answer = vectorize_stories(mydata)
```

```
pred_results = model.predict(([my_story, my_question]))
```

```
1/1 [==============================] - 0s 25ms/step
```

```
# generate predictions from model
val_max = np.argmax(pred_results[0])
for key, val in tokenizer.word_index.items():
    if val == val_max:
        k = key
print("Predicted answer : ", k)
print("Probability of certainity :", pred_results[0][val_max])
```

```
Predicted answer :  no
Probability of certainity : 0.5826604
```

CHATBOT CODE LINK:

Python code link: https://colab.research.google.com/drive/1Bwb7jTmpV-hB_WmRFnUO1b0gq675NvcO?usp=sharing

Dialog flow code: https://bot.dialogflow.com/aeb9e239-c576-425d-b4c2-f9936f34b84c

CONCLUSION:

However, there is one solution primed to satisfy the modern customer, and that is a chatbot. With a chatbot, your organization can easily offer high-quality support and conflict resolution any time of day, and for a large quantity of customers simultaneously.

According to Microsoft, **90%** of consumers expect an online portal for customer service. As a significant aspect of business evolution, the need for AI-powered chatbots will only continue to rise. Now is the time to deploy a chatbot solution so that your company doesn’t get left behind.