Open in app Get started

Published in Towards Data Science

This is your **last** free member-only story this month.
Sign up for Medium and get an extra one

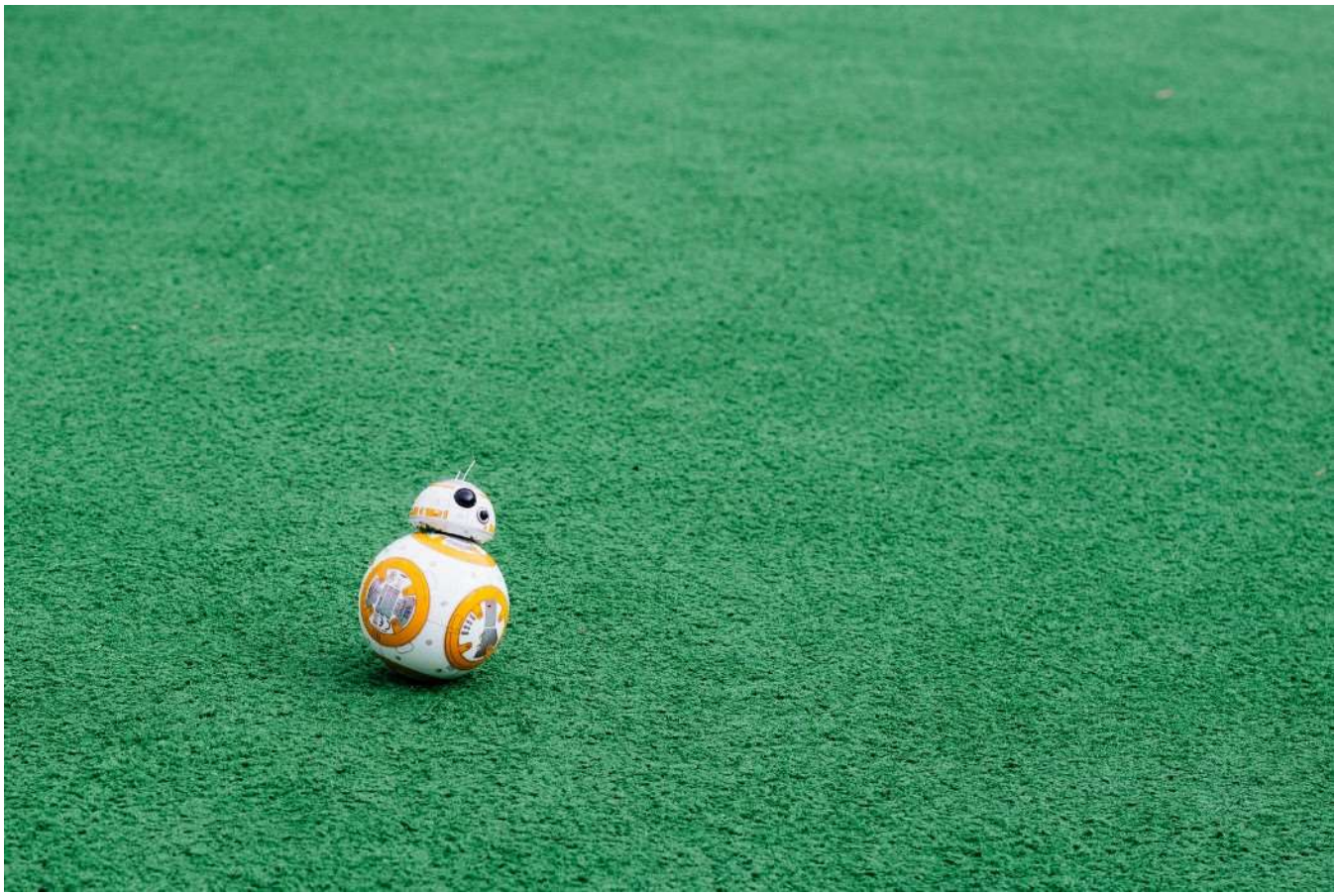
Jere Xu Follow

Jan 12, 2020 · 8 min read ★ · Listen

Save

How To Create A Chatbot with Python & Deep Learning In Less Than An Hour

Obviously don't expect it to be Siri or Alexa...



Hopefully one day BB-8 will become reality...

Open in app Get started

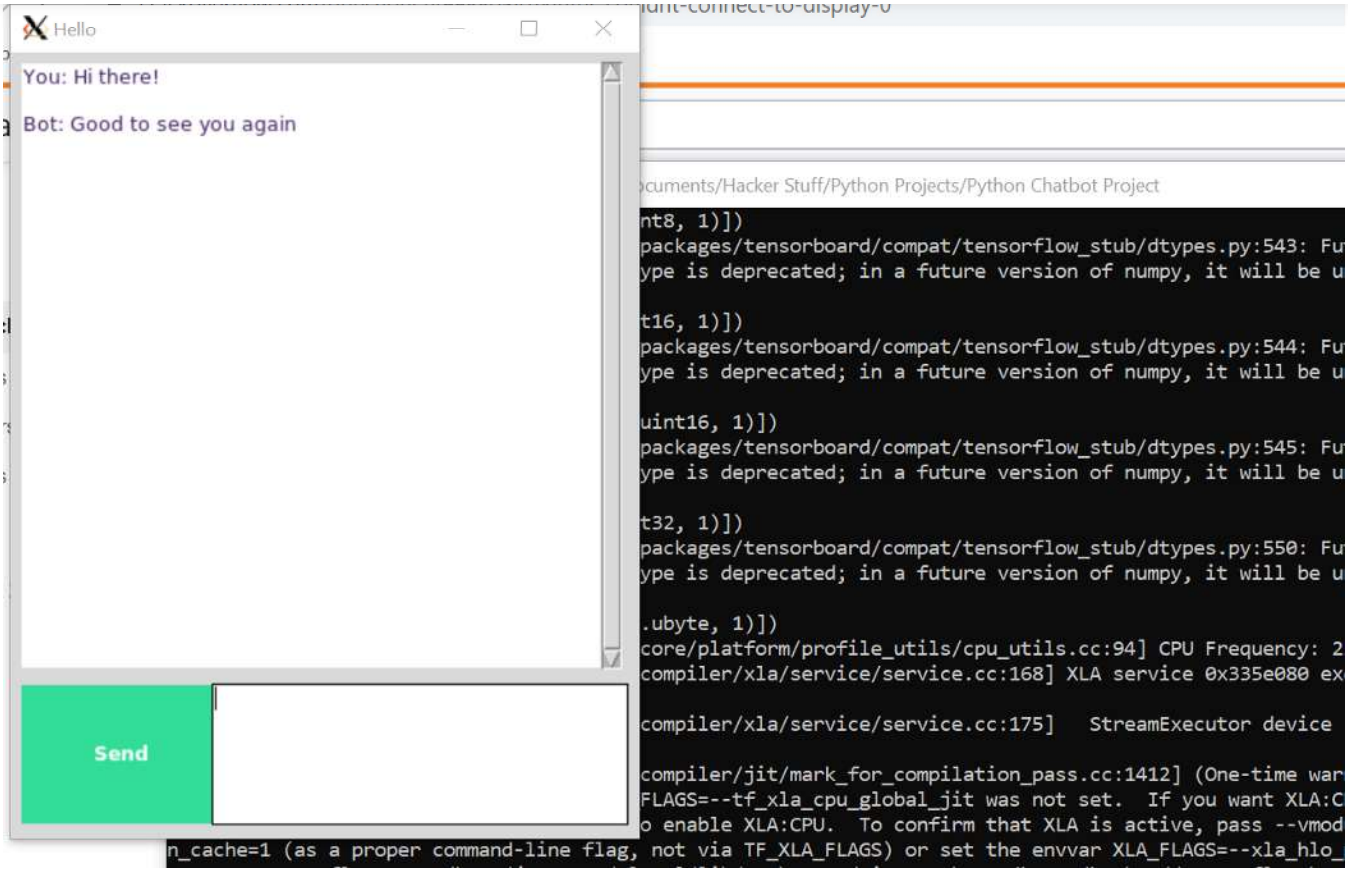
with other people. There are plenty of people on this Earth who are the exact opposite, who get very drained from social interaction.

I'm reminded of a very unique film called *Her (2013)*. The basic premise of the film is that a man who suffers from loneliness, depression, a boring job, and an impending divorce, ends up falling in love with an AI (artificial intelligence) on his computer's operating system. Maybe at the time this was a very science-fictiony concept, given that AI back then wasn't advanced enough to become a surrogate human, but now? 2020? Things have changed a LOT. I fear that people will give up on finding love (or even social interaction) among humans and seek it out in the digital realm. Don't believe me? I won't tell you what it means, but just search up the definition of the term ***waifu*** and just *cringe*.

Now isn't this an overly verbose introduction to a simple machine learning project? Possibly. Now that I've detailed an issue that has grounds for actual concern for many men (and women) in this world, let's switch gears and build something simple and fun!

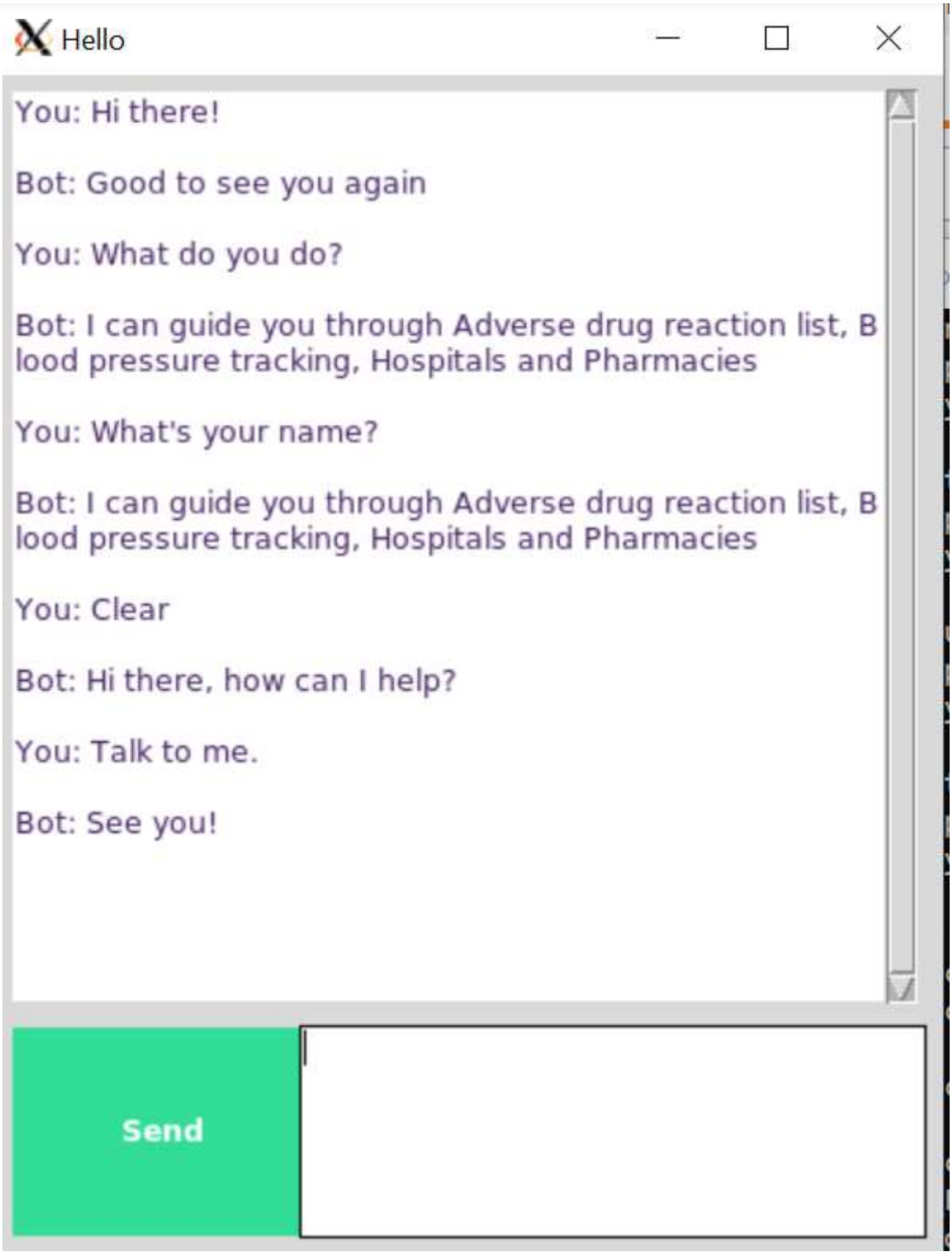
Here's what the finished product will look like

Open in app Get started



Nice simple interface

Open in app Get started



Obviously this chatbot is EXTREMELY limited in its responses

Agenda

1. Libraries & Data

Open in app Get started

4. Building Chatbot GUI
5. Running Chatbot
6. Conclusion
7. Areas of Improvement

If you want a more in-depth view of this project, or if you want to add to the code, check out the GitHub repository.

Libraries & Data

All of the necessary components to run this project are on the GitHub repository. Feel free to fork the repository and clone it to your local machine. Here's a quick breakdown of the components:

- **train_chatbot.py —** the code for reading in the natural language data into a training set and using a Keras sequential neural network to create a model
- **chatgui.py —** the code for cleaning up the responses based on the predictions from the model and creating a graphical interface for interacting with the chatbot
- **classes.pkl —** a list of different types of classes of responses
- **words.pkl —** a list of different words that could be used for pattern recognition
- **intents.json —** abunch of JavaScript objects that lists different tags that correspond to different types of word patterns
- **chatbot_model.h5 —** the actual model created by train_chatbot.py and used by chatgui.py

The full code is on the GitHub repository, but I'm going to walk through the details of the code for the sake of transparency and better understanding.

Now let's begin by importing the necessary libraries. (When you run the python files


Open in app Get started

```
nltk.download('punkt')
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import json
import pickle

import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.optimizers import SGD
import random
```

libraries.py hosted with ❤ by GitHub view raw

We have a whole bunch of libraries like *nltk* (Natural Language Toolkit), which contains a whole bunch of tools for cleaning up text and preparing it for deep learning algorithms, *json*, which loads json files directly into Python, *pickle*, which loads pickle files, *numpy*, which can perform linear algebra operations very efficiently, and *keras*, which is the deep learning framework we'll be using.

Initializing Chatbot Training

```
words=[]
classes = []
documents = []
ignore_words = ['?', '!']
data_file = open('intents.json').read()
intents = json.loads(data_file)
```

init.py hosted with ❤ by GitHub view raw

Now it's time to initialize all of the lists where we'll store our natural language data. We have our json file I mentioned earlier which contains the "intents". Here's a snippet of what the json file actually looks like.

Open in app Get started

```
{"intents": [
        {"tag": "greeting",
         "patterns": ["Hi there", "How are you", "Is anyone there?","Hey","Hola", "Hello", "Good day"],
         "responses": ["Hello, thanks for asking", "Good to see you again", "Hi there, how can I help?"],
         "context": [""]
        },
        {"tag": "goodbye",
         "patterns": ["Bye", "See you later", "Goodbye", "Nice chatting to you, bye", "Till next time"],
         "responses": ["See you!", "Have a nice day", "Bye! Come back again soon."],
         "context": [""]
        },
        {"tag": "thanks",
         "patterns": ["Thanks", "Thank you", "That's helpful", "Awesome, thanks", "Thanks for helping me"],
         "responses": ["Happy to help!", "Any time!", "My pleasure"],
         "context": [""]
        },
        {"tag": "noanswer",
         "patterns": [],
         "responses": ["Sorry, can't understand you", "Please give me more info", "Not sure I understand"],
         "context": [""]
        },
```

Typical json format

We use the json module to load in the file and save it as the variable *intents*.

```
for intent in intents['intents']:
    for pattern in intent['patterns']:

        # take each word and tokenize it
        w = nltk.word_tokenize(pattern)
        words.extend(w)
        # adding documents
        documents.append((w, intent['tag']))

        # adding classes to our class list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])
```

words.py hosted with ❤ by GitHub view raw

If you look carefully at the json file, you can see that there are sub-objects within objects. For example, “patterns” is an attribute within “intents”. So we will use a *nested for loop* to extract all of the words within “patterns” and add them to our **words** list. We then add to our **documents** list each pair of patterns within their corresponding tag. We also add the tags into our **classes** list, and we use a simple conditional statement to

Open in app Get started

```
3
4   classes = sorted(list(set(classes)))
5
6   print (len(documents), "documents")
7
8   print (len(classes), "classes", classes)
9
10  print (len(words), "unique lemmatized words", words)
11
12
13  pickle.dump(words,open('words.pkl','wb'))
14  pickle.dump(classes,open('classes.pkl','wb'))
```

lem.py hosted with ❤ by GitHub view raw

Next, we will take the **words** list and lemmatize and lowercase all the words inside. In case you don't already know, **lemmatize** means to turn a word into its base meaning, or its **lemma.** For example, the words "walking", "walked", "walks" all have the same lemma, which is just "walk". The purpose of lemmatizing our words is to narrow everything down to the simplest level it can be. It will save us a lot of time and unnecessary error when we actually process these words for machine learning. This is very similar to **stemming**, which is to reduce an inflected word down to its base or root form.

Next we sort our lists and print out the results. Alright, looks like we're set to build our deep learning model!

Building the Deep Learning Model

```
1   # initializing training data
2   training = []
3   output_empty = [0] * len(classes)
4   for doc in documents:
5       # initializing bag of words
6       bag = []
7       # list of tokenized words for the pattern
8       pattern_words = doc[0]
```

Open in app Get started

```
14
15      # output is a '0' for each tag and '1' for current tag (for each pattern)
16      output_row = list(output_empty)
17      output_row[classes.index(doc[1])] = 1
18
19      training.append([bag, output_row])
20  # shuffle our features and turn into np.array
21  random.shuffle(training)
22  training = np.array(training)
23  # create train and test lists. X - patterns, Y - intents
24  train_x = list(training[:,0])
25  train_y = list(training[:,1])
26  print("Training data created")
```

Let's initialize our training data with a variable *training*. We're creating a giant nested list which contains bags of words for each of our documents. We have a feature called *output_row* which simply acts as a key for the list. We then shuffle our training set and do a train-test-split, with the patterns being the X variable and the intents being the Y variable.

```
1   # Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd output laye
2   # equal to number of intents to predict output intent with softmax
3   model = Sequential()
4   model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
5   model.add(Dropout(0.5))
6   model.add(Dense(64, activation='relu'))
7   model.add(Dropout(0.5))
8   model.add(Dense(len(train_y[0]), activation='softmax'))
9
10  # Compile model. Stochastic gradient descent with Nesterov accelerated gradient gives good resu
11  sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
12  model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
13
14  #fitting and saving the model
15  hist = model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=5, verbose=1)
16  model.save('chatbot_model.h5', hist)
17
18  print("model created")
```

Open in app Get started

details about how deep learning models work, but if you are curious, check out the resources at the bottom of the article.

The Sequential model in keras is actually one of the simplest neural networks, a multi-layer perceptron. If you don't know what that is, I don't blame you. Here's the documentation in keras.

This particular network has 3 layers, with the first one having 128 neurons, the second one having 64 neurons, and the third one having the number of intents as the number of neurons. Remember, the point of this network is to be able to predict which intent to choose given some data.

The model will be trained with stochastic gradient descent, which is also a very complicated topic. Stochastic gradient descent is more efficient than normal gradient descent, that's all you need to know.

After the model is trained, the whole thing is turned into a numpy array and saved as *chatbot_model.h5*.

We will use this model to form our chatbot interface!

Building Chatbot GUI

```
1  from keras.models import load_model
2  model = load_model('chatbot_model.h5')
3  import json
4  import random
5  intents = json.loads(open('intents.json').read())
6  words = pickle.load(open('words.pkl','rb'))
7  classes = pickle.load(open('classes.pkl','rb'))
```

chat_init.py hosted with ❤ by GitHub view raw

Once again, we need to extract the information from our files.

Open in app Get started

```python
# return bag of words array: 0 or 1 for each word in the bag that exists in the sentence

def bow(sentence, words, show_details=True):
    # tokenize the pattern
    sentence_words = clean_up_sentence(sentence)
    # bag of words - matrix of N words, vocabulary matrix
    bag = [0]*len(words)
    for s in sentence_words:
        for i,w in enumerate(words):
            if w == s:
                # assign 1 if current word is in the vocabulary position
                bag[i] = 1
                if show_details:
                    print ("found in bag: %s" % w)
    return(np.array(bag))

def predict_class(sentence, model):
    # filter out predictions below a threshold
    p = bow(sentence, words,show_details=False)
    res = model.predict(np.array([p]))[0]
    ERROR_THRESHOLD = 0.25
    results = [[i,r] for i,r in enumerate(res) if r>ERROR_THRESHOLD]
    # sort by strength of probability
    results.sort(key=lambda x: x[1], reverse=True)
    return_list = []
    for r in results:
        return_list.append({"intent": classes[r[0]], "probability": str(r[1])})
    return return_list

def getResponse(ints, intents_json):
    tag = ints[0]['intent']
    list_of_intents = intents_json['intents']
    for i in list_of_intents:
        if(i['tag']== tag):
            result = random.choice(i['responses'])
            break
    return result

def chatbot_response(msg):
    ints = predict_class(msg, model)
    res = getResponse(ints, intents)
    return res
```

Open in app Get started

which takes the sentences that are cleaned up and creates a bag of words that are used for predicting classes (which are based off the results we got from training our model earlier).

In our *predict_class()* function, we use an error threshold of 0.25 to avoid too much overfitting. This function will output a list of intents and the probabilities, their likelihood of matching the correct intent. The function *getResponse()* takes the list outputted and checks the json file and outputs the most response with the highest probability.

Finally our *chatbot_response()* takes in a message (which will be inputted through our chatbot GUI), predicts the class with our *predict_class()* function, puts the output list into *getResponse()*, then outputs the response. What we get is the foundation of our chatbot. We can now tell the bot something, and it will then respond back.

```
#Creating GUI with tkinter
import tkinter
from tkinter import *


def send():
    msg = EntryBox.get("1.0",'end-1c').strip()
    EntryBox.delete("0.0",END)

    if msg != '':
        ChatLog.config(state=NORMAL)
        ChatLog.insert(END, "You: " + msg + '\n\n')
        ChatLog.config(foreground="#442265", font=("Verdana", 12 ))

        res = chatbot_response(msg)
        ChatLog.insert(END, "Bot: " + res + '\n\n')

        ChatLog.config(state=DISABLED)
        ChatLog.yview(END)


base = Tk()
base.title("Hello")
base.geometry("400x500")
```


Open in app Get started

```
ChatLog.config(state=DISABLED)

#Bind scrollbar to Chat window
scrollbar = Scrollbar(base, command=ChatLog.yview, cursor="heart")
ChatLog['yscrollcommand'] = scrollbar.set

#Create Button to send message
SendButton = Button(base, font=("Verdana",12,'bold'), text="Send", width="12", height=5,
                    bd=0, bg="#32de97", activebackground="#3c9d9b",fg='#ffffff',
                    command= send )

#Create the box to enter message
EntryBox = Text(base, bd=0, bg="white",width="29", height="5", font="Arial")
#EntryBox.bind("<Return>", send)


#Place all components on the screen
scrollbar.place(x=376,y=6, height=386)
ChatLog.place(x=6,y=6, height=386, width=370)
EntryBox.place(x=128, y=401, height=90, width=265)
SendButton.place(x=6, y=401, height=90)

base.mainloop()
```

Here comes the run part (if the other parts weren't run already). We can create our GUI with tkinter, a Python library that allows us to create custom interfaces.

We create a function called *send()* which sets up the basic functionality of our chatbot. If the message that we input into the chatbot is not an empty string, the bot will output a response based on our *chatbot_response()* function.

After this, we build our chat window, our scrollbar, our button for sending messages, and our textbox to create our message. We place all the components on our screen with simple coordinates and heights.

Running Chatbot

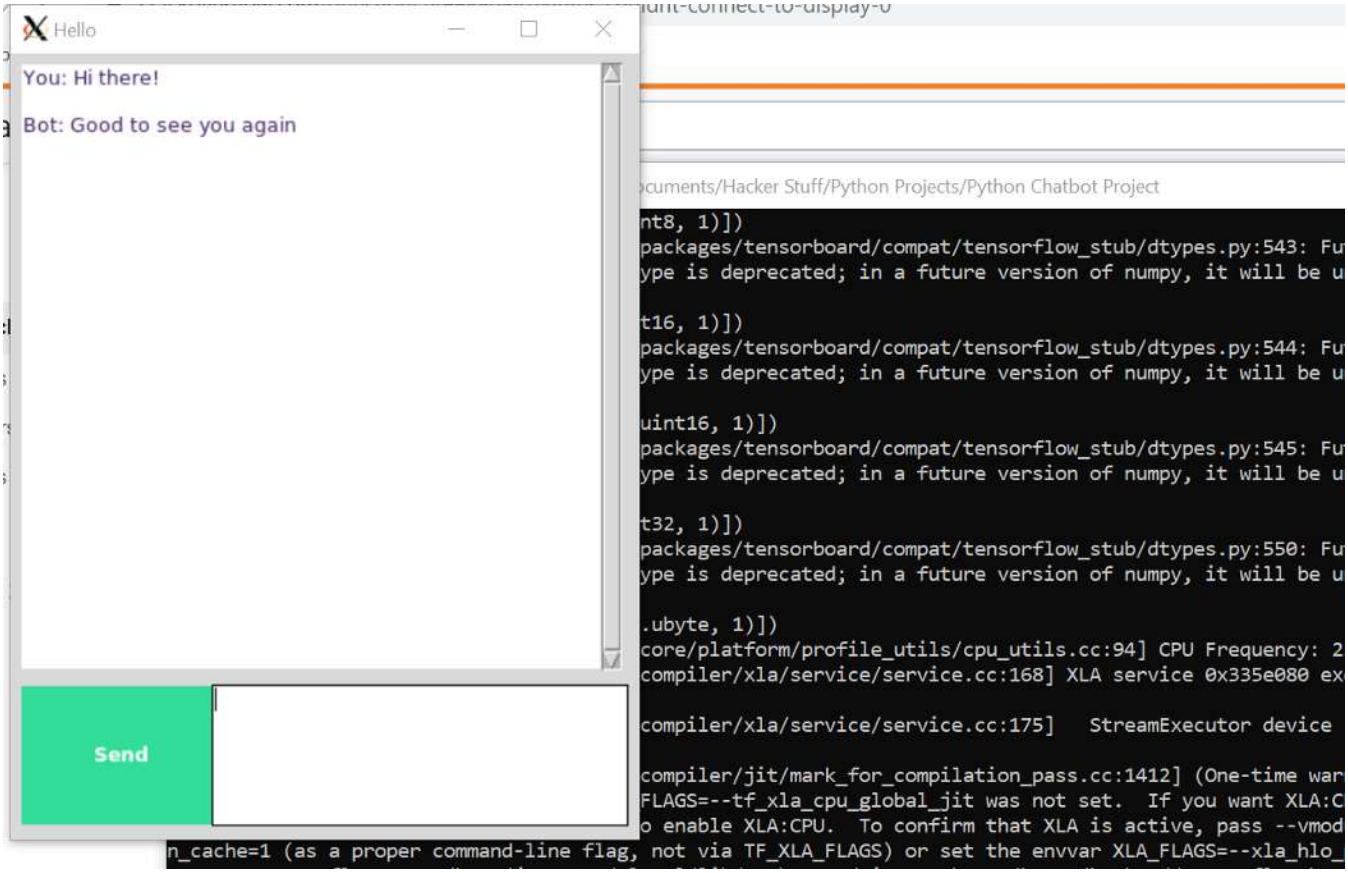
Finally it's time to run our chatbot!

Open in app Get started

program failing, you can download Xming.

Before you run your program, you need to make sure you install python or python3 with pip (or pip3). If you are unfamiliar with command line commands, check out the resources below.

Once you run your program, you should get this.



Conclusion

Congratulations on completing this project! Building a simple chatbot exposes you to a variety of useful skills for data science and general programming. I feel that the best way (for me, at least) to learn anything is to just build and tinker around. If you want to become good at something, you need to get in lots of practice, and the best way to practice is to just get your hands dirty and build!

Open in app Get started

Thank you for taking the time to read through this article! Feel free to check out my portfolio site or my GitHub.

1. Trying out different neural networks

We used the simplest keras neural network, so there is a LOT of room for improvement. Feel free to try out convolutional networks or recurrent networks for your projects.

2. Using more data

Our json file was extremely tiny in terms of the variety of possible intents and responses. Human language is billions of times more complex than this, so creating JARVIS from scratch will require a lot more.

3. Using different frameworks

There are many more deep learning frameworks than just keras. There's tensorflow, Apache Spark, PyTorch, Sonnet, and more. Don't limit yourself to just one tool!

Resources

- Deep Learning
- Natural Language Processing
- Command Line
- Neural Networks
- Tkinter

1.1K | 10

Sign up for The Variable

Open in app Get started

Get this newsletter

About Help Terms Privacy

Get the Medium app

Download on the App Store GET IT ON Google Play