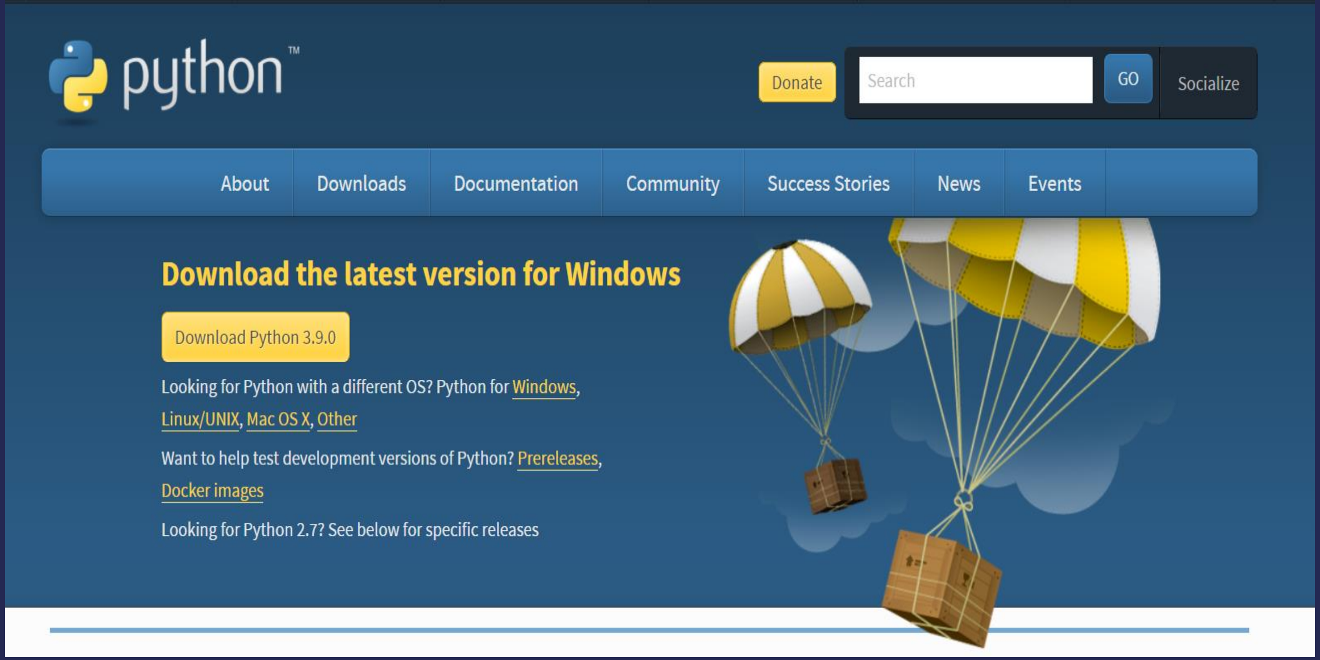
DEEP LEARNING E-BOOK

Chatbot Project

PYCHARM INSTALLATION

- Before installing PyCharm, you should install the latest version of Python

- Download and install Python visit the official website of Python https://www.python.org/downloads/ and choose your version



Download the latest version for Windows

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Prereleases, Docker images

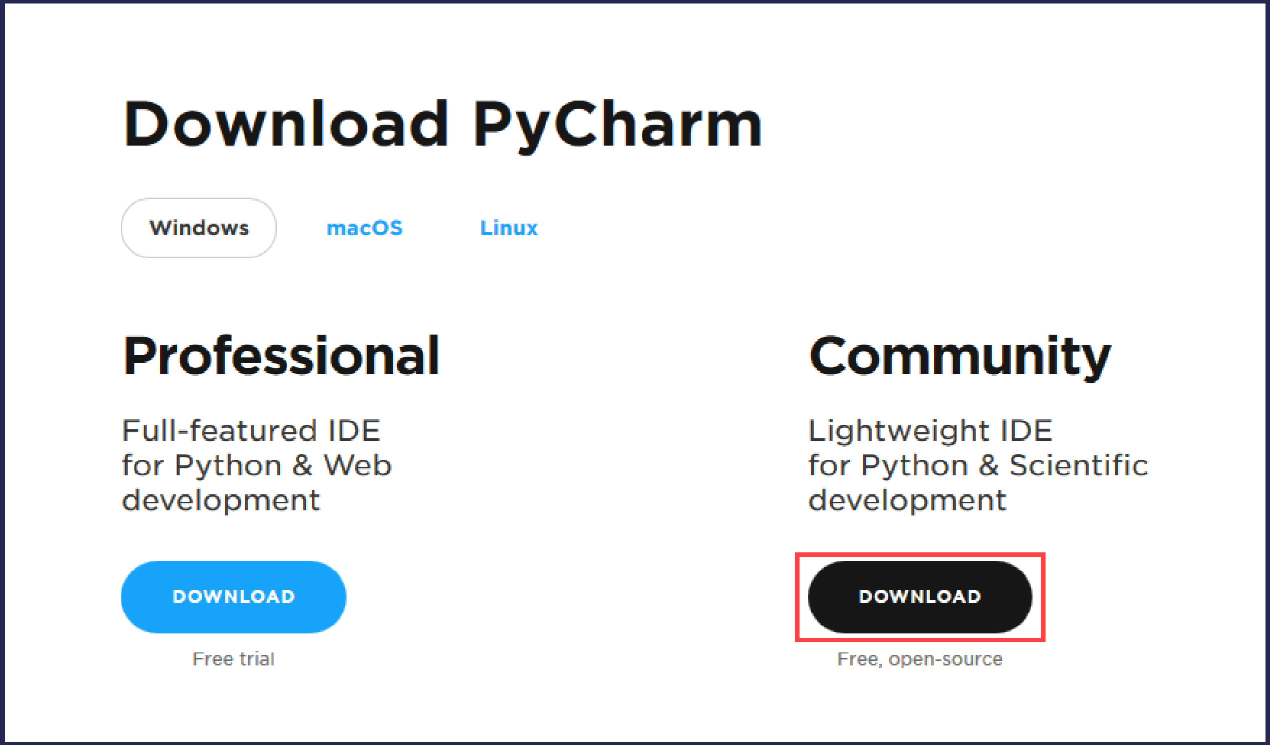
Looking for Python 2.7? See below for specific releases

PYCHARM INSTALLATION

To download PyCharm visit the website

https://www.jetbrains.com/pycharm/download/

and Click the DOWNLOAD link under the Community Section.



Download PyCharm

Professional

Full-featured IDE for Python & Web development

Free trial

Community

Lightweight IDE for Python & Scientific development

Free, open-source

CHATBOT

- A chatbot is an intelligent piece of software that is capable of communicating and performing actions similar to a human.
- Chatbots are used a lot in customer interaction, marketing on social network sites and Instagram.
- There are two basic types of chatbot models based on how they are built; Retrieval based and Generative based models.
- We are using Retrieval based model for our chatbot project.
- A retrieval-based chatbot uses predefined input patterns and responses.

ABOUT THE PROJECT – CHATBOT

- In this Python project, we are going to build a chatbot using deep learning techniques.
- The chatbot will be trained on the dataset which contains categories (intents), pattern and responses.
- We use a special recurrent neural network to classify which category the user's message belongs to and then we will give a random response from the list of responses.
- The dataset we will be using is 'intents.json'. This is a JSON file that contains the patterns we need to find and the responses we want to return to the user.

PREREQUISITES

- Natural language processing (NLTK).
- Python,
- Keras,
- Tensorflow

You can download using the python-pip command.

```
pip install tensorflow-cpu, keras, pickle, nltk
```

TYPES OF FILES NEEDED FOR CHATBOT

- **Intents.json –** The data file which has predefined patterns and responses.
- **train_chatbot.py –** In this Python file, we wrote a script to build the model and train our chatbot.
- **Words.pkl –** This is a pickle file in which we store the words Python object that contains a list of our vocabulary.
- **Classes.pkl –** The classes pickle file contains the list of categories.
- **Chatbot_model.h5 –** This is the trained model that contains information about the model and has weights of the neurons.
- **Chatgui.py –** This is the Python script in which we implemented GUI for our chatbot. Users can easily interact with the bot.

NOTE : Pickle converts a Python object into byte streams that can be saved to disks or can be transferred over the network. JSON stands for JavaScript Object Notation. This is similar to Pickling but instead of storing data in binary format, JSON stores data in human-readable text format.

5 STEPS TO CREATE A CHATBOT

1. Import and load the data file
2. Preprocess data
3. Create training and testing data
4. Build the model
5. Predict the response

1. IMPORT AND LOAD THE DATA FILE

```
import nltk
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

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

- First, create a file name as train_chatbot.py and import and load the necessary packages for our chatbot

2. PREPROCESS DATA

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

    #tokenize each word
    w = nltk.word_tokenize(pattern)
    words.      (w)
    #add documents in the corpus
    documents.      ((w, intent['tag']))

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

These highlighted boxes ▨ are to be filled with the appropriate word.

Here you have to fill it with appropriate method and you can use either append() or extend() method.

- Python append() method adds an element to a list
- The extend() method concatenates the first list with another list (or another iterable).

2. PREPROCESS DATA

```
# lemmaztize and lower each word and remove duplicates
words = [lemmatizer.lemmatize(w.lower()) for w in          if w not in
ignore_words]
words = sorted(list(set(          )))
# sort classes
classes = sorted(list(set(classes)))
# documents = combination between patterns and intents
print (len(          ), "documents")
# classes = intents
print (len(classes), "classes“, classes)
# words = all words, vocabulary
print (len(words), "unique lemmatized words“ ,           )


pickle.dump(words, open('words.pkl', 'wb'))
pickle.dump(classes, open('classes.pkl', 'wb'))
```

- First, we should lemmatize each word and remove duplicate words from the list.
- Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item.

3. CREATE TRAINING AND TESTING DATA

```
# create our training data
training = []
# create an empty array for our output
output_empty = [0] * len(classes)
# training set, bag of words for each sentence
for doc in documents:
    # initialize our bag of words
    bag = []
    # list of tokenized words for the pattern
    pattern_words = doc[0]
    # lemmatize each word - create base word, in attempt to represent related words
    pattern_words = [lemmatizer.lemmatize(word.lower()) for word in          ]
    # create our bag of words array with 1, if word match found in current pattern
    for w in          :
        bag.append(1) if w in pattern_words else bag.append(0)

    # output is a '0' for each tag and '1' for current tag (for each pattern)
    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1

    training.append([bag, output_row])
# shuffle our features and turn into np.array
random.shuffle(training)
training = np.array(training)
# create train and test lists. X - patterns, Y - intents
train_x = list(training[:,0])
train_y = list(training[:,1])
print("Training data created")
```

4. BUILD THE MODEL

```
# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd output layer contains number of
neurons
# equal to number of intents to predict output intent with softmax
model = Sequential()
model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(train_y[0]), activation='softmax'))

# Compile model. Stochastic gradient descent with Nesterov accelerated gradient gives good results for this model
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

#fitting and saving the model
hist = model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=5, verbose=1)
model.save('chatbot_model.h5', hist)

print("model created")
```

5. PREDICT THE RESPONSE (GRAPHICAL USER INTERFACE)

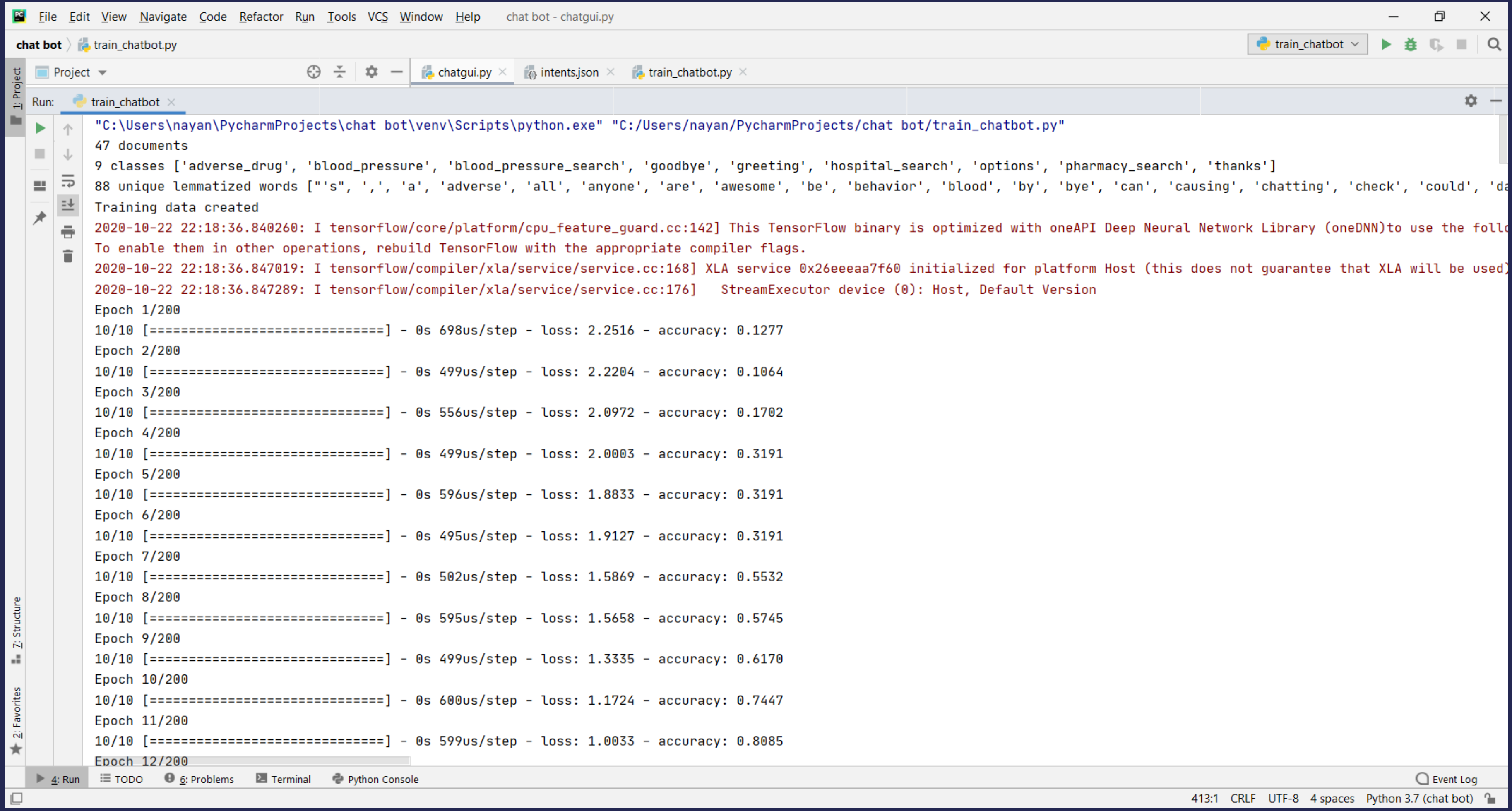
- Next we should predict the sentences and get a response from the user.
- We should create a new file ‘chatgui.py’ and the link to access the file is given below

- We will load the trained model and then use a graphical user interface that will predict the response from the bot.
- The model will only tell us the class it belongs to, so we will implement some functions which will identify the class and then retrieve us a random response from the list of responses.

6. RUN THE CHATBOT

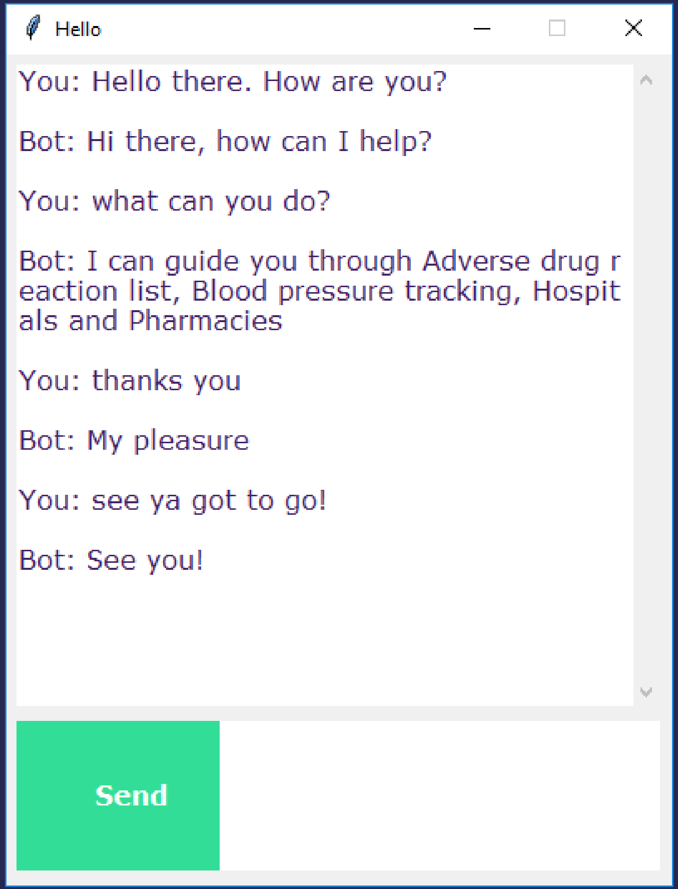
- To run the chatbot, we have two main files; **train_chatbot.py** and **chatgui.py**.
- First run the **train_chatbot.py** file. We train the model by running this file.
- If we don't see any error during training, we have successfully created the model. Then to run the app, we run the second file.
- The program will open up a GUI window within a few seconds. With the GUI you can easily chat with the bot., **chatgui.py**.

O/P OF TRAIN_CHATBOT.PY


File Edit View Navigate Code Refactor Run Tools VCS Window Help chat bot - chatgui.py
chat bot train_chatbot.py
train_chatbot
Project
chatgui.py
intents.json
train_chatbot.py
Run: train_chatbot
4: Run TODO 6: Problems Terminal Python Console
Event Log
413:1 CRLF UTF-8 4 spaces Python 3.7 (chat bot)

```
"C:\Users\nayan\PycharmProjects\chat bot\venv\Scripts\python.exe" "C:/Users/nayan/PycharmProjects/chat bot/train_chatbot.py"
47 documents
9 classes ['adverse_drug', 'blood_pressure', 'blood_pressure_search', 'goodbye', 'greeting', 'hospital_search', 'options', 'pharmacy_search', 'thanks']
88 unique lemmatized words ["'s", ',', 'a', 'adverse', 'all', 'anyone', 'are', 'awesome', 'be', 'behavior', 'blood', 'by', 'bye', 'can', 'causing', 'chatting', 'check', 'could', 'da
Training data created
2020-10-22 22:18:36.840260: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN)to use the follo
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2020-10-22 22:18:36.847019: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x26eeeaa7f60 initialized for platform Host (this does not guarantee that XLA will be used)
2020-10-22 22:18:36.847289: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecutor device (0): Host, Default Version
Epoch 1/200
10/10 [==============================] - 0s 698us/step - loss: 2.2516 - accuracy: 0.1277
Epoch 2/200
10/10 [==============================] - 0s 499us/step - loss: 2.2204 - accuracy: 0.1064
Epoch 3/200
10/10 [==============================] - 0s 556us/step - loss: 2.0972 - accuracy: 0.1702
Epoch 4/200
10/10 [==============================] - 0s 499us/step - loss: 2.0003 - accuracy: 0.3191
Epoch 5/200
10/10 [==============================] - 0s 596us/step - loss: 1.8833 - accuracy: 0.3191
Epoch 6/200
10/10 [==============================] - 0s 495us/step - loss: 1.9127 - accuracy: 0.3191
Epoch 7/200
10/10 [==============================] - 0s 502us/step - loss: 1.5869 - accuracy: 0.5532
Epoch 8/200
10/10 [==============================] - 0s 595us/step - loss: 1.5658 - accuracy: 0.5745
Epoch 9/200
10/10 [==============================] - 0s 499us/step - loss: 1.3335 - accuracy: 0.6170
Epoch 10/200
10/10 [==============================] - 0s 600us/step - loss: 1.1724 - accuracy: 0.7447
Epoch 11/200
10/10 [==============================] - 0s 599us/step - loss: 1.0033 - accuracy: 0.8085
Epoch 12/200
```

O/P OF CHATGUI.PY



```
You: Hello there. How are you?

Bot: Hi there, how can I help?

You: what can you do?

Bot: I can guide you through Adverse drug r
eaction list, Blood pressure tracking, Hospit
als and Pharmacies

You: thanks you

Bot: My pleasure

You: see ya got to go!

Bot: See you!
```

This is the GUI window. With this you can easily chat with the bot.