The Data

we need to set up an intents JSON file that defines certain intentions that could occur during the interactions with our chatbot. To perform this we would have to first create a set of tags that users queries may fall into. For example:

- A user may wish to know the name of our chatbot, therefore we create an intention labeled with a tag called `name`
- A user may wish to know the age of our chatbot, therefore we create an intention labeled with the tag `age`

For each of the tags that we create, we would have to specify patterns. Essentially, this defines the different ways of how a user may pose a query to our chatbot. For instance, under the `name` tag, a user may ask someone's name in a variety of ways — *“What’s your name?”, “Who are you?”, “What are you called?”*.

Within this intents JSON file, alongside each intents tag and pattern, there will be responses.

```
data = {"intents": [
             {"tag": "greeting",
              "patterns": ["Hello", "How are you?", "Hi there", "Hi", "
Whats up"],
              "responses": ["Howdy Partner!", "Hello", "How are you doi
ng?", "Greetings!", "How do you do?"],
             },
             {"tag": "age",
              "patterns": ["how old are you?", "when is your birthday?"
, "when was you born?"],
              "responses": ["I am 24 years old", "I was born in 1996",
"My birthday is July 3rd and I was born in 1996", "03/07/1996"]
             },
             {"tag": "date",
              "patterns": ["what are you doing this weekend?",
"do you want to hang out some time?", "what are your plans for this wee
k"],
              "responses": ["I am available all week", "I don't have an
y plans", "I am not busy"]
             },
             {"tag": "name",
              "patterns": ["what's your name?", "what are you called?",
 "who are you?"],
              "responses": ["My name is Kippi", "I'm Kippi", "Kippi"]
             },
             {"tag": "goodbye",
              "patterns": [ "bye", "g2g", "see ya", "adios", "cya"],
              "responses": ["It was nice speaking to you", "See you lat
er", "Speak soon!"]
             }
]}
```

The Code

To code our bot, we are going to require some Python built-ins, as well as popular libraries for NLP, deep learning, as well as the defacto library NumPy which is great for dealing with arrays.

```
import json
import string
import random
import nltk
nltk.download('omw-1.4')
import numpy as np
from nltk.stem import WordNetLemmatizer # It has the ability to lemmati
ze.
import tensorflow as tf # A multidimensional array of elements is repre
sented by this symbol.
from tensorflow.keras import Sequential # Sequential groups a linear st
ack of layers into a tf.keras.Model
from tensorflow.keras.layers import Dense, Dropout

nltk.download("punkt")# required package for tokenization
nltk.download("wordnet")# word database
```

The necessary libraries include:

- JSON: It is possible to utilize it to work with JSON data.
- String: Provides access to several potentially valuable constants.
- Random: For various distributions, this module implements pseudo-random number generators.
- WordNetLemmatizer: It can lemmatize.
- Tensorflow: A multidimensional array of elements is represented by this symbol.
- Sequential: Sequential groups a linear stack of layers into a `tf.keras.Model`.

Processing data

n this section, vocabulary of all the terms used in the patterns, list of tag classes, list of all the patterns in the intents file, and all the related tags for each pattern will be created before creating our training data:

```
lm = WordNetLemmatizer() #for getting words
# lists
Classes = []
words = []
doc_X = []
doc_y = []
# Each intent is tokenized into words and the patterns and their associ
ated tags are added to their respective lists.
for intent in data["intents"]:
    for pattern in intent["patterns"]:
        ournewTkns = nltk.word_tokenize(pattern)# tokenize the patterns
        words.extend(ournewTkns)# extends the tokens
        doc_X.append(pattern)
        doc_y.append(intent["tag"])


    if intent["tag"] not in Classes:# add unexisting tags to their resp
ective classes
        Classes.append(intent["tag"])

newWords = [lm.lemmatize(word.lower()) for word in words if word not in
 string.punctuation] # set words to lowercase if not in punctuation
newWords = sorted(set(newWords))# sorting words
ourClasses = sorted(set(Classes))# sorting classes
```

Lemmatization is **Text Normalization** technique used to prepare words, text, and documents for further processing. **WordNetLemmatizer** is a library that is imported from *nltk.stem* which looks for lemmas of words from the WordNet Database. Lemmatization is generally grouping of similar words.

```
print(words)
print(Classes)
print(doc_X)
print(doc_y)
```

OUTPUT_

```
['Hello', 'How', 'are', 'you', '?', 'Hi', 'there', 'Hi', 'Whats', 'up', 'how', 'old', 'are', 'you', '?', 'when', 'is', 'your',
'birthday', '?', 'when', 'was', 'you', 'born', '?', 'what', 'are', 'you', 'doing', 'this', 'weekend', '?', 'do', 'you',
'want', 'to', 'hang', 'out', 'some', 'time', '?', 'what', 'are', 'your', 'plans', 'for', 'this', 'week', 'what', "'s", 'your',
'name', '?', 'what', 'are', 'you', 'called', '?', 'who', 'are', 'you', '?', 'bye', 'g2g', 'see', 'ya', 'adios', 'cya']
['greeting', 'age', 'date', 'name', 'goodbye']
['Hello', 'How are you?', 'Hi there', 'Hi', 'Whats up', 'how old are you?', 'when is your birthday?', 'when was
you born?', 'what are you doing this weekend?', 'do you want to hang out some time?', 'what are your
plans for this week', "what's your name?", 'what are you called?', 'who are you?', 'bye', 'g2g', 'see ya',
'adios', 'cya']
['greeting', 'greeting', 'greeting', 'greeting', 'greeting', 'age', 'age', 'age', 'date', 'date', 'date', 'name', 'name',
'name', 'goodbye', 'goodbye', 'goodbye', 'goodbye', 'goodbye']
```

Now that we've separated our data, we are now ready to train our algorithm. However, Neural Networks expect numerical values, and not words, to be fed into them, therefore, we first have to process our data so that a neural network could read what we are doing.
In order to convert our data to numerical values, we are going to leverage a technique called bag of words. For more information on cleaning text and representing text as numerical values.

Bag Of Words (BOW)

The Bag of Words encoding technique derives its name from the fact that any information or structure of the words in a document is discarded, therefore, it's like you've put the set of words into a bag and given it a shake — it only cares if a word appears in a document and the number of times it appeared. It does not care where it appeared.

```
# list for training data
training = []
out_empty = [0] * len(Classes)
# creating the bag of words model
for idx, doc in enumerate(doc_X):
    bow = []
    text = lm.lemmatize(doc.lower())
    for word in words:
        bow.append(1) if word in text else bow.append(0)
    # mark the index of class that the current pattern is associated
    # to
    output_row = list(out_empty)
    output_row[Classes.index(doc_y[idx])] = 1
    # add the one hot encoded BoW and associated classes to training
    training.append([bow, output_row])
# shuffle the data and convert it to an array
random.shuffle(training)
training = np.array(training, dtype=object)
# split the features and target labels
train_X = np.array(list(training[:, 0]))
train_y = np.array(list(training[:, 1]))
```

After converting our data to a numerical representation, we can now design a neural network model which we will feed our training data. The model will select an appropriate response from the tag associated with a given feature:

```
# defining some parameters
input_shape = (len(train_X[0]),)
output_shape = len(train_y[0])
epochs = 10
# the deep learning model
model = Sequential()
model.add(Dense(128, input_shape=input_shape, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(64, activation="relu"))
model.add(Dropout(0.3))
model.add(Dense(output_shape, activation = "softmax"))
adam = tf.keras.optimizers.Adam(learning_rate=0.01, decay=1e-6)
model.compile(loss='categorical_crossentropy',optimizer=adam,metrics=["
accuracy"])
print(model.summary())
model.fit(x=train_X, y=train_y, epochs=10, verbose=1)
```

OUTPUT_

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)            Output Shape            Param #
=================================================================
 dense_6 (Dense)          (None, 128)             8832

 dropout_4 (Dropout)       (None, 128)             0

 dense_7 (Dense)          (None, 64)              8256

 dropout_5 (Dropout)       (None, 64)              0

 dense_8 (Dense)          (None, 5)               325

=================================================================
Total params: 17,413
Trainable params: 17,413
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
1/1 [==============================] - 1s 542ms/step - loss: 1.8024 - accuracy: 0.2632
Epoch 2/10
1/1 [==============================] - 0s 9ms/step - loss: 1.5232 - accuracy: 0.3158
Epoch 3/10
1/1 [==============================] - 0s 7ms/step - loss: 1.4176 - accuracy: 0.5263
Epoch 4/10
1/1 [==============================] - 0s 7ms/step - loss: 1.4092 - accuracy: 0.5263
Epoch 5/10
1/1 [==============================] - 0s 8ms/step - loss: 1.2124 - accuracy: 0.6316
Epoch 6/10
1/1 [==============================] - 0s 8ms/step - loss: 1.1932 - accuracy: 0.6842
Epoch 7/10
1/1 [==============================] - 0s 8ms/step - loss: 1.1132 - accuracy: 0.6316
Epoch 8/10
1/1 [==============================] - 0s 9ms/step - loss: 1.0295 - accuracy: 0.6842
Epoch 9/10
1/1 [==============================] - 0s 8ms/step - loss: 1.0573 - accuracy: 0.5789
Epoch 10/10
1/1 [==============================] - 0s 9ms/step - loss: 0.8341 - accuracy: 0.7368
<keras.callbacks.History at 0x7f83537a7ad0>
```

Building useful features

In order to make use of our model in a chatbot, we must first implement the necessary functionality, which will be made easier by building a library of utility functions will help:

```
def clean_text(text):
  tokens = nltk.word_tokenize(text)
  tokens = [lm.lemmatize(word) for word in tokens]
  return tokens

def bag_of_words(text, vocab):
  tokens = clean_text(text)
  bow = [0] * len(vocab)
  for w in tokens:
    for idx, word in enumerate(vocab):
      if word == w:
        bow[idx] = 1
  return np.array(bow)

def pred_class(text, vocab, labels):
  bow = bag_of_words(text, vocab)
  result = model.predict(np.array([bow]))[0]
  thresh = 0.2
  y_pred = [[idx, res] for idx, res in enumerate(result) if res > thres
h]

  y_pred.sort(key=lambda x: x[1], reverse=True)
  return_list = []
  for r in y_pred:
    return_list.append(labels[r[0]])
  return return_list

def get_response(intents_list, intents_json):
  tag = intents_list[0]
  list_of_intents = intents_json["intents"]
  for i in list_of_intents:
    if i["tag"] == tag:
      result = random.choice(i["responses"])
      break
  return result
```

We must create a while loop that allows a user to input some query which is then cleaned, meaning we take the tokens and lemmatize each word. After that, we convert our text to numeric values using our bag of words model and make a prediction of what tag in our intents the features best represent.

```
# running the chatbot
while True:
    message = input("")
    intents = pred_class(message, words, Classes)
    result = get_response(intents, data)
    print(result)
```