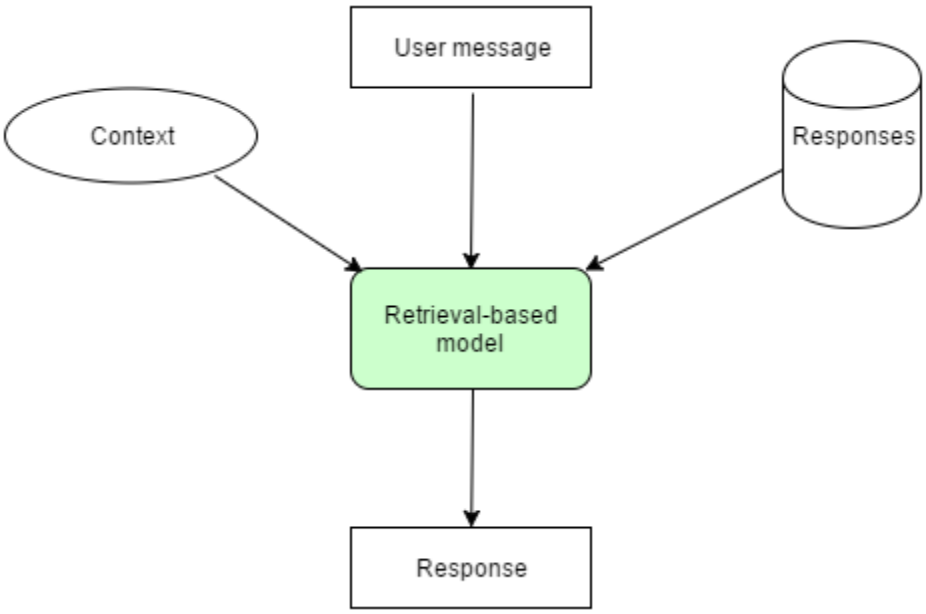
Price Negotiator Chatbot

Project Modeling.

Architecture model of chatbot

The architecture model that is based on to build the chatbot is Retrieval-Based Models. The chatbot uses some heuristic to select an appropriate response from a library of predefined responses. This architectural model of a chatbot is easier to build and much more reliable. Though there cannot be 100% accuracy of responses, you can know the possible types of responses and ensure that no inappropriate or incorrect response is delivered by the chatbot.



Retrieve from:https://dzone.com/articles/understanding-architecture-models-of-chatbot-and-r

This bot considers the message and context of the conversation to deliver the best response from a predefined list of messages.

How do Chatbots Work?

Chatbots are nothing but an intelligent piece of software that can interact and communicate with people just like humans. Interesting, isn't it? So now let's see how they actually work.

All chatbots come under the NLP (Natural Language Processing) concepts. NLP is composed of two things:

- **NLU** (Natural Language Understanding): The ability of machines to understand human language like English.
- **NLG** (Natural Language Generation): The ability of a machine to generate text similar to human written sentences.

Build the Chatbot

Prerequisites

To implement the chatbot, we will be using Keras, which is a Deep Learning library, NLTK, which is a Natural Language Processing toolkit, and some helpful libraries.

Project File Structure

- **Train_chatbot.py** — In this file, we will build and train the deep learning model that can classify and identify what the user is asking to the bot.
- **Chatbot.py** — This file is where we will build functions to connect a graphical user interface to chat with trained chatbot.
- **Intents.json** — The intents file has all the data that we will use to train the model. It contains a collection of tags with their corresponding patterns and responses.
- **Chatbot_model.h5** — This is a hierarchical data format file in which we have stored the weights and the architecture of our trained model.
- **Classes.pkl** — The pickle file can be used to store all the tag names to classify when we are predicting the message.
- **Words.pkl** — The words.pkl pickle file contains all the unique words that are the vocabulary of our model.

Step 1. Import Libraries and Load the Data

Create a new python file and name it as train_chatbot and then we are going to import all the required modules. After that, we will read the JSON data file in our Python program.

Python (train_chatbot.py)

```
import pickle
import json
import numpy as np
from keras.models import Sequential
```

```
from keras.layers import Dense, Activation, Dropout
#from keras.optimizers import SGD
from tensorflow.keras.optimizers import SGD
import random

import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
```

Step 2. Preprocessing the Data

The model cannot take the raw data. It has to go through a lot of pre-processing for the machine to easily understand. For textual data, there are many preprocessing techniques available. The first technique is tokenizing, in which we break the sentences into words.

By observing the intents file, we can see that each tag contains a list of patterns and responses. We tokenize each pattern and add the words in a list. Also, we create a list of classes and documents to add all the intents associated with patterns.

Python (train_chatbot.py)

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

for intent in intents['intents']:
    for pattern in intent['patterns']:
        # tokenize each word
        word = nltk.word_tokenize(pattern)
        words.extend(word)
        # add documents in the corpus
        documents.append((word, intent['tag']))
        # add to our classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])
```

Another technique is Lemmatization. We can convert words into the lemma form so that we can reduce all the canonical words. For example, the words play, playing, plays, played, etc. will all be replaced with play. This way, we can reduce the number of total words in our vocabulary. So now we lemmatize each word and remove the duplicate words.

Python (train_chatbot.py)

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

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

In the end, the words contain the vocabulary of our project and classes contain the total entities to classify. To save the python object in a file, we used the pickle.dump() method. These files will be helpful after the training is done and we predict the chats.

Step 3. Create Training and Testing Data

To train the model, we will convert each input pattern into numbers. First, we will lemmatize each word of the pattern and create a list of zeroes of the same length as the total number of words. We will set value 1 to only those indexes that contain the word in the patterns. In the same way, we will create the output by setting 1 to the class input the pattern belongs to.

Python (train_chatbot.py)

```
# 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 pattern_words]
    # create our bag of words array with 1, if word match found in current pattern
    for word in words:
        bag.append(1) if word 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])
```

Step 4. Training the Model

The architecture of our model will be a neural network consisting of 3 dense layers. The first layer has 128 neurons, the second one has 64 and the last layer will have the same neurons as the number of classes. The dropout layers are introduced to reduce overfitting of the model. We have used the SGD optimizer and fit the data to start the training of the model. After the training of 200 epochs is completed, we then save the trained model using the Keras model.save("chatbot_model.h5") function.

Python (train_chatbot.py)

```
# 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)
```

Step 5. Interacting With the Chatbot

The model is ready to chat. In our GUI file, the chatbot will capture the user message and again perform some preprocessing before we input the message into our trained model.

The model will then predict the tag of the user's message, and we will randomly select the response from the list of responses in our intents file.

Python (chatbot.py)

```
import os
import random
import json
from keras.models import load_model
import numpy as np
import pickle
import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()


model = load_model(os.path.join("./ai/components", 'chatbot_model.h5'))
intents = json.loads(
    open(os.path.join("./ai/components", 'intents.json')).read())
words = pickle.load(open(os.path.join("./ai/components", 'words.pkl'), 'rb'))
classes = pickle.load(
    open(os.path.join("./ai/components", 'classes.pkl'), 'rb'))

print('words', words)
print('classes', classes)
```

```
def clean_up_sentence(sentence):
    # tokenize the pattern - splitting words into array
    sentence_words = nltk.word_tokenize(sentence)
    # stemming every word - reducing to base form
    sentence_words = [lemmatizer.lemmatize(
        word.lower()) for word in sentence_words]
    return sentence_words


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


def predict_class(sentence):
    # filter below  threshold predictions
    p = bag_of_words(sentence, words, show_details=True)
    flag = False
    return_list = []
    for x in p:
        if x == 1:
            flag = True
    if (flag == True):
        res = model.predict(np.array([p]))[0]
        ERROR_THRESHOLD = 0.25
        results = [[i, r] for i, r in enumerate(res) if r > ERROR_THRESHOLD]
        # sorting strength probability
        results.sort(key=lambda x: x[1], reverse=True)

        print("result", results)
        for r in results:
```

```
            return_list.append(
                {"intent": classes[r[0]], "probability": str(r[1])})
    else:
        return_list.append({"intent": "noanswer", "probability": str(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)
    print('ints', ints)
    res = getResponse(ints, intents)
    print('res', res)
    return res
```

References :

Shivashish Thkaur CORE ·, Thkaur, S., 04, M., & Like (31) Comment . (2020, May 4). *Build your first python chatbot project - dzone AI*. dzone.com. Retrieved November 23, 2021, from https://dzone.com/articles/python-chatbot-project-build-your-first-python-pro.

Smith, A. (2020, April 2). *Understanding architecture models of chatbot and Response Generation Mechanisms - DZone AI*. dzone.com. Retrieved November 23, 2021, from https://dzone.com/articles/understanding-architecture-models-of-chatbot-and-r.