Steps of Build model for system chatting by language Arabic for robot

1. Download and install library

```
pip install tensorflow  keras  pickle  nltk PyQt5
```

2. Import and load the data file

Because the chat by language Arabic we add the text in code

```
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)
intents = {"intents": [
        {"tag": "تحية",
         "patterns": ["يوم سعيد" ,"مرحبا" ,"هلا","يا","هل من أحد هناك؟","كيف حالك" ,"أهلا"],
         "responses": ["مرحبًا ، كيف يمكنني المساعدة؟" ,"سررت برؤيتك مجددا" ,"مرحبا شكرا على السؤال"],
         "context": [""]
        },
        {"tag": "وداع",
         "patterns": ["القادمة المرة حتى" ,"لطيفة الدردشة معك ، وداعا" ,"مع السلامة" ,"أراك لاحقا" ,"داعا"],
         "responses": ["أتمنى لك نهارا سعيد" ,"وداعا! عد مرة أخرى قريبا" ,"أرك لاحقا!."],
         "context": [""]
        },
        {"tag": "شكر",
         "patterns": ["شكرا لمساعدتي" ,"رائع شكرا" ,"هذا مفيد" ,"شكرا لك" ,"شكرا"],
         "responses": ["في أي وقت!" ,"من دواعي سروري" ,"سررت بالمساعدة!"],
         "context": [""]
        },
        {"tag": "لا_اجابة",
         "patterns": [],
         "responses": ["لست متأكدا بأني أفهم" ,"من فضلك أعطني المزيد من المعلومات" ,"لا أستطيع أن أفهمك"],
         "context": [""]
        },
        {"tag": "الخيارات",
         "patterns": ["ما هو الدعم المقدم" ,"كيف يمكنك ان تكون مفيدا؟" ,"ما هي المساعدة التي تقدمها؟" ,
         "responses": ["تقديم الدعم للتفاعلات الدوائية الضارة وضغط الدم والمستشفيات والصيدليات" ,"ليات."],
         "context": [""]
        },
```

3. Preprocess data

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

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

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

```
# lemmaztize and lower each word and remove duplicates
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]
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'))
```

4. Create training and testing data

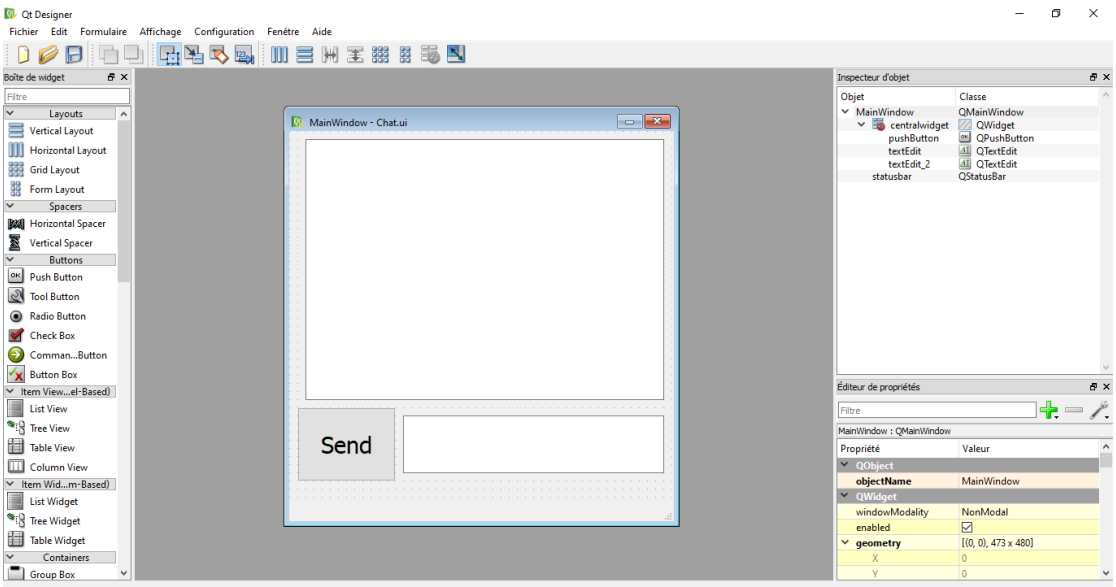
```
# create our training data
training = []
# create an empty array for our output
output_empty = [0] * len(classes)
# training set, bag of words for each sentence
for doc in documents:
    # initialize our bag of words
    bag = []
    # list of tokenized words for the pattern
    pattern_words = doc[0]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

5. Build the model

```
# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd output layer contains numbe
# 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 mod
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")
```

6. Predict the response (Graphical User Interface)

Create the user interface by library Pyqt design



Add the library of Pyqt in code

```
from PyQt5.QtCore import pyqtSlot
from PyQt5.QtWidgets import *
from PyQt5.uic import loadUi
from PyQt5 import QtWidgets, uic
```

Add code of user interface in code for the chat

```
app = QtWidgets.QApplication([])
dlg = uic.loadUi("Chat.ui")
dlg.show()

dlg.pushButton.clicked.connect(send)
app.exit(app.exec_())
```

7. Run the chatbot

Run code of training

```
python train_chatbot.py
```

Run code of active the chat

```
python chatgui.py
```