```
import nltk
import json
import pickle
import random
import keras
from google.colab import files
uploaded = files.upload()
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()


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


words=[]
classes = []
documents = []
ignore_words = ['?', '!']
# The file was uploaded to the current working directory, not the parent
# Use 'intents.json' instead of '../intents.json'
data_file = open('intents.json').read()
intents = json.loads(data_file)

# ... (rest of your code remains the same) ...
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'))

# 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])

#Before converting to numpy array, ensure all bags have the same length
bag_len = len(words)  # Get the expected length of the bag of words
for i in range(len(training)):
    if len(training[i][0]) != bag_len:  # If bag length is incorrect
        diff = bag_len - len(training[i][0])  # Calculate the difference
        training[i][0].extend([0] * diff)  # Pad with zeros to match expected length
```

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

# 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(learning_rate=0.01, decay=1e-6, momentum=0.9, nesterov=True) # Changed 'lr' to 'learning_rate'
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")
```

Show hidden output

```
import nltk
import json
import pickle
import random
import keras
from google.colab import files
uploaded = files.upload()  # File upload line remains here

# ... (other imports and code)

# The file was uploaded to the current working directory, not the parent
# Use 'intents.json' instead of '../intents.json'
```

```
data_file = open('intents.json').read()
intents = json.loads(data_file)

# Print the loaded data
print("Uploaded Data:")  # Add this line to print the data
print(json.dumps(intents, indent=4))  # Format the JSON for better readability

# ... (rest of your code)
```

Show hidden output

```
import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import pickle
import numpy as np

from keras.models import load_model
# Make sure 'chatbot_model.h5' exists in the current directory, if not, run the training cell first
model = load_model('chatbot_model.h5')
import json
import random
intents = json.loads(open('intents.json').read())
words = pickle.load(open('words.pkl','rb'))
classes = pickle.load(open('classes.pkl','rb'))


def clean_up_sentence(sentence):
    # tokenize the pattern - split words into array
    sentence_words = nltk.word_tokenize(sentence)
    # stem each word - create short form for word
    sentence_words = [lemmatizer.lemmatize(word.lower()) for word in sentence_words]
    return sentence_words

# 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


#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.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)

!pip install ipywidgets
import ipywidgets as widgets
from IPython.display import display

# ... (rest of your code including imports and functions)

# Create the UI elements
chat_history = widgets.Output()
user_input = widgets.Text(placeholder="Enter your message here")
send_button = widgets.Button(description="Send")

# Function to handle sending messages
def on_send_button_clicked(b):
    global user_message
    user_message = user_input.value

    with chat_history:
        display(widgets.HTML(f"<b>You:</b> {user_message}"))  # Display user message

    bot_response = chatbot_response(user_message)
    with chat_history:
        display(widgets.HTML(f"<b>Bot:</b> {bot_response}"))  # Display bot response

    user_input.value = ""  # Clear the input field

# Connect the send button to the event handler
send_button.on_click(on_send_button_clicked)

# Display the UI elements
display(chat_history)
display(user_input)
display(send_button)
```

Show hidden output

Start coding or generate with AI.