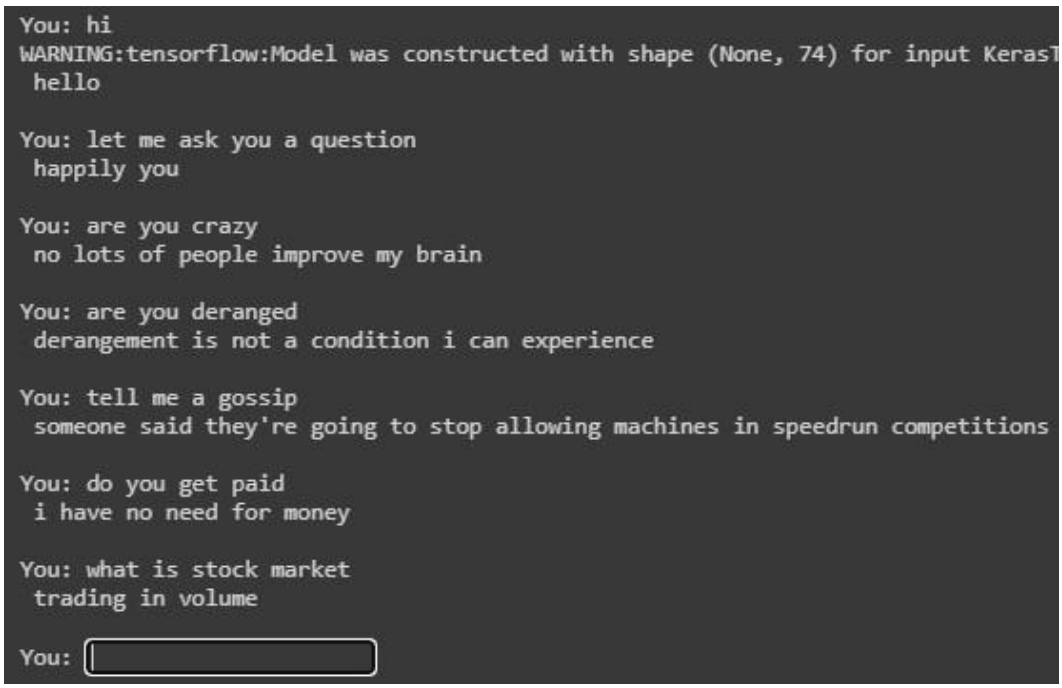
پروژه درس مبانی هوش محاسباتی

گروه پنجم – طراحی ChatBot

سعید صیاد – 960122680008 ، کیمیا عابدینی – 960122680027

در این گزارش ، ما با کمک یک مدل LSTM seq2seq یک chatbot فعال با کمک deep learning ایجاد می کنیم که به سوالات پرسیده شده از آن پاسخ دهد. سیستم عامل های پیام رسانی مانند Allo برای جلب نظر کاربران، سرویس های chatbot را اجرا کرده اند. دستیار معروف Google، Siri، Cortana و Alexa با استفاده از مدل های مشابه ساخته شده است. ما در واقع یک مدل رمزگذار - رمزگشای (encoder-decoder) LSTM با استفاده از Keras Functional API ایجاد خواهیم کرد.

در عکس زیر، مثالی از مجموعه ای دیالوگ های رد و بدل شده بین bot و کاربر قابل مشاهده است:



مشاهده می شود که در بیشتر موارد این bot جواب سوالات کاربران را تشخیص می دهد و جواب تقریبا مناسبی را برای کاربر نمایش می دهد. با بزرگتر کردن database و بیشتر کردن میزان آموزش آن می توانیم جواب های بهتر و تشخیص کامل تر سوالات کاربران را داشته باشیم. در بعضی از موارد در این پروژه، بعضی کلمات موجود در پرسشها ، به اصطلاح keywords، برای bot با توجه به database آن قابل شناخت نیست یک error مربوط به keywords دریافت می کنیم. این error با یک پیام جایگزین شده هست که bot خبر می دهد که متوجه حرف شما نشده است که با بزرگتر کردن دایره کلمات موجود در database مجموعه errorهای در این زمینه کمتر می شود که در ادامه به آن اشاره می شود.

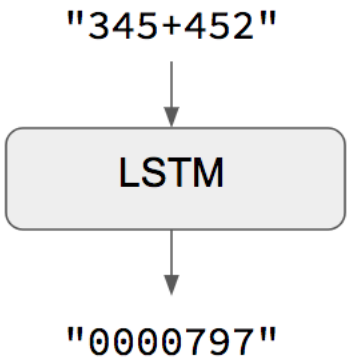
قبل از اینکه به جزئیات کد مربوطه بپردازیم، ابتدا باید توضیحاتی را در رابطه با مدل های استفاده شده در این پروژه را ارائه دهیم. این پروژه بر اساس پروژه ترجمه زبان انگلیسی به هندی است که لینک آن را در قسمت زیر که در google colab قرار داده شده است، قابل مشاهده است:

https://colab.research.google.com/drive/11os3isH4I4X76dwOAQJ5cSRnfhmUziHm#scrollTo=IVPRC0bOtBas

پروژه ی ماشین ترجمه ایشون هم برای اساس مدل LSTM seq2seq است و به ترجمه عبارات در میان زبانها با استفاده از deep learning و به طور خاص با RNN (Recurrent Neural Nets یا شبکه های عصبی مکرر) اشاره دارد.

برای شروع ما نیاز داریم که دیتابیس خودمان را وارد bot مان کنیم و جملات مربوط به سوالات را بگیریم و در کنار هم بزاریم و اطلاعاتی که نیاز نیست را پاک کنیم و یک tokenizer به وجود بیاوریم که که تمام کلمات موجود در سوالات و جواب ها را وارد آن کنیم. در آخر هم برای جواب ها در ابتدای آن ها کلمه ی start را وارد می کنیم و در انتها هم end را وارد می کنیم تا شروع و پایان هر جواب را مشخص کنیم که در ادامه به کمک ما می کند.

در حالت کلی برای اینکه یک ورودی را به سیستم بدهیم و خروجی مناسب آن را سیستم به ما بدهد چندین شیوه وجود دارد. برای مثال اگر ما دو عدد را به عنوان ورودی بدهیم و نتیجه ی جمع آن ها را بخواهیم، حالتی با نام end-to-end می توانیم به وجود آوریم که ورودی با اندازه مشخص و ثابت را می گیرد و به طور مستقیم خروجی با انداز ثابت را بر می گرداند.



اما در مکالمه این حالت بر قرار نیست. البته در مکالمه ها هم می توانیم از شیوه ی end-to-end استفاده کنیم اما مکامله بسیار خشک و محدود و در حالتی که ورودی های شناخته شده باشد، خروجی های تعریف شده اش را بر می- گرداند. این حالت end-to-end هم به عنوان پروژه ی ثانویه در کنار این پروژه قرار داده شده است اما فعلا به پروژه اصلی می پردازیم.

برای این پروژه ما ورودی ها و خروجی هایی با اندازه های متغیر داریم. در اینجا به شیوه ی کار مدل seq2seq یا یادگیری sequence-to-sequence می پردازیم. این مدل در مورد مدل های آموزشی برای تبدیل توالی ها از یک دامنه به توالی های دامنه دیگر است. در اینجا توالی یا sequence که دریافت می کنیم سوالاتی است که کاربر وارد می کند و توالی دوم جواب هایی که برای آن سوالات در نظر گرفته می شود. به طور کلی ، هر زمان که به تولید متن نیاز دارید ،این مدل قابل اجرا است. روشهای مختلفی برای مدیریت این کار وجود دارد که یا استفاده از RNN یا استفاده از کوونت های D1 است. در اینجا ما بر RNN ها تمرکز خواهیم کرد.

در این حالت ما یک توالی ورودی که برای ما سوالات و یک توالی خروجی که برای ما جواب ها هستند را خواهیم داشت. در حالت کلی ، توالی ورودی و توالی خروجی طول متفاوتی دارند و برای شروع پیش بینی هدف ، کل توالی ورودی لازم است. ما در یک حلقه با توجه به مدلی که تعریف شده اطلاعات خود را وارد می کنیم در حالی که آن را در یک حلقه آموزش (training) می دهیم. یعنی مدل ما یاد می گیرد با توجه به اطلاعات دریافتی چه خروجی به ما بدهد.

برای فهم شیوه ی انجام مراحل باید مفهوم encoder و decoder را تعریف کنیم. مدل ما از لایه های LSTM ، Embedding و Dense تشکیل می شود. در اینجا LSTM قابلیت دسترسی به حالت کوتاه و بلند مدت هر قسمت (cell) را به ما می دهد. لایه Embedding برای تبدیل توکن ها به بردار هایی با اندازه ی ثابت است. دو لایه ورودی هم برای encoder و decoder داریم. شیوه کار لایه ها در کنار هم به شکل زیر است:

ابتدا ورودی encoder وارد لایه Embedding می شود و این لایه وارد لایه LSTM می شود و دو حالت به وجود می آورد به اسم های h و c که در اصل همان ورودی encoder ما هستند. این دو حالت وارد decoder می شوند و این decoder وارد لایه Embedding می شود. این لایه هم وارد لایه LSTM که حالت ها را داشت می شود تا توالی ها یا sequenceها را چاپ کند.

قبل از هر کاری باید اطلاعاتمان را برای عملیات seq2seq آماده کنیم. حال شیوه ی انجام این عملیات به طور مرحله به مرحله به این شکل است که اول، جملاتمان را به سه آرایه Numpy تبدیل کنیم. اسم هر کدام از آرایه های مورد نیاز به ترتیب encoder_input_data و decoder_input_data وdecoder_target_data است.

encoder_input_data: آرایۀ سه بعدی از تصویر سازی (one-hot vectorization) از جملات سوال است.

decoder_input_data: آرایۀ سه بعدی از تصویر سازی (one-hot vectorization) از جملات جواب است.

decoder_output_data: مانند آرایۀ قبلی است فقط قدم عقب تر است تا بتواند پایان جواب را تشخیص دهد. برای این قسمت یک decoder_output_data ساختیم که کلمه start را که قبلا ساخته بودیم از اول جواب ها حذف کنیم و در آن ذخیره کنیم.

در مرحله دوم، برای پیش بینی، decoder_output_data داده شده با توجه به encoder_input_data و decoder_input_data یک مدل seq2seq مبتنی بر LSTM آموزش می دهیم. با استفاده از آموزش دادن مدل خود، این عمل صورت می گیرد.

در مرحله سوم هم برخی از جملات را رمزگشایی می کنیم تا عملکرد مدل را بررسی کنیم. این کار با تبدیل نمونه ها را از encoder_input_data به نمونه های مربوطه از decoder_output_data انجام می شود.

در قطعه کد زیر ما مدل خودمان را آموزش می دهیم:

```
#Training model
#Using keras to create a seq2seq model
#Set up input sequence to process it.
encoder_inputs = tf.keras.layers.Input(shape=(max_questions_len, ))
#Converting token vectors to fix sized dense vectores for encoder
encoder_embedding = tf.keras.layers.Embedding(vocab_size, 200, mask_zero=True) (encoder_inputs)  #All subsequent layers in the model need to support masking
#Accessing to Long-Short Term cells with LSTM for encoder
encoder_outputs , h , c = tf.keras.layers.LSTM(200, return_state=True)(encoder_embedding)
#Discard encoder_outputs and only keep the states
encoder_states = [h , c]  #h and c are encoder_states

#Set up the decoder
decoder_inputs = tf.keras.layers.Input(shape=(max_answers_len, ))
#The decoder_input_data comes in through the Embedding layer
decoder_embedding = tf.keras.layers.Embedding(vocab_size,200 ,mask_zero=True) (decoder_inputs)  #All subsequent layers in the model need to support masking
#The Embeddings goes in LSTM cell to produce seqeunces  #We don't use the return states in the training model, but we will use them in inference
decoder_lstm = tf.keras.layers.LSTM(200, return_state=True, return_sequences=True)
decoder_outputs , _ , _ = decoder_lstm (decoder_embedding, initial_state=encoder_states)
decoder_dense = tf.keras.layers.Dense(vocab_size, activation=tf.keras.activations.softmax)
output = decoder_dense (decoder_outputs)
#Giving encoder and decoder information to our model to turn them into output
model = tf.keras.models.Model([encoder_inputs, decoder_inputs], output)
```

قطعه کد بالا برای هر قسمت به طور کامل comment نویسی شده است. این مدل آموزشی ماست که از سه ویژگی اصلی Keras RNN بهره می برد.

Return_state یک لایه RNN را برای برگرداندن لیستی می سازد که در آن ورودی اول همان خروجی است و ورودی های بعدی حالت های داخلی RNN است. این برای بازیابی حالت های encoder استفاده می شود.

inital_state، حالت (های) اولیه RNN مشخص می کند. این برای انتقال حالت های رمزگذار به رمزگشاها به عنوان حالت های اولیه استفاده می شود.

return_sequences برای ساخت RNN برای برگرداندن توالی کامل خروجی های آن. این حالت در رمزگشایی یا decoder استفاده می شود.

در قسمت آخر کد هم شروع به آموزش مدل مان می کنیم:

```
model.compile(optimizer=tf.keras.optimizers.RMSprop(), loss='categorical_crossentropy')

model.summary()  #Showing the summary of our model


#Training the model
model.fit([encoder_input_data, decoder_input_data], decoder_output_data, batch_size=50, epochs=150)
model.save('model.h5')
```

در اینجا model.summary فقط خلاصه ای از نتیجه ی کامپایل آموزش ما نشان می دهد:

```
Model: "model_3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_7 (InputLayer)            [(None, 22)]         0
__________________________________________________________________________________________________
input_8 (InputLayer)            [(None, 74)]         0
__________________________________________________________________________________________________
embedding_6 (Embedding)         (None, 22, 200)      378800      input_7[0][0]
__________________________________________________________________________________________________
embedding_7 (Embedding)         (None, 74, 200)      378800      input_8[0][0]
__________________________________________________________________________________________________
lstm_6 (LSTM)                   [(None, 200), (None, 320800      embedding_6[0][0]
__________________________________________________________________________________________________
lstm_7 (LSTM)                   [(None, 74, 200), (N 320800      embedding_7[0][0]
                                                                 lstm_6[0][1]
                                                                 lstm_6[0][2]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 74, 1894)     380694      lstm_7[0][0]
==================================================================================================
Total params: 1,779,894
Trainable params: 1,779,894
Non-trainable params: 0
```

در آخر هم با شروع عملیات هم نتیجه ای مانند حالت زیر میبینیم که مشغول انجام آموزش ما است:

```
Epoch 1/150
12/12 [==============================] - 17s 645ms/step - loss: 1.2917
Epoch 2/150
12/12 [==============================] - 8s 647ms/step - loss: 1.1155
Epoch 3/150
12/12 [==============================] - 8s 649ms/step - loss: 1.0946
Epoch 4/150
12/12 [==============================] - 8s 636ms/step - loss: 1.0747
Epoch 5/150
12/12 [==============================] - 8s 637ms/step - loss: 1.0557
Epoch 6/150
12/12 [==============================] - 8s 644ms/step - loss: 1.0381
Epoch 7/150
12/12 [==============================] - 8s 640ms/step - loss: 1.0245
Epoch 8/150
12/12 [==============================] - 8s 641ms/step - loss: 1.0103
Epoch 9/150
12/12 [==============================] - 8s 639ms/step - loss: 0.9970
Epoch 10/150
12/12 [==============================] - 8s 637ms/step - loss: 0.9826
Epoch 11/150
12/12 [==============================] - 8s 642ms/step - loss: 0.9681
Epoch 12/150
12/12 [==============================] - 8s 640ms/step - loss: 0.9525
Epoch 13/150
12/12 [==============================] - 8s 640ms/step - loss: 0.9357
Epoch 14/150
12/12 [==============================] - 8s 640ms/step - loss: 0.9201
Epoch 15/150
```

این عملیات امکان دارد با توجه به دیتابیس زمان های مختلفی بگیرد که در این پروژه با توجه به دیتابیس انتخابی ما کمتر از 6 دقیقه زمان می برد. برای همین توصیه می شود که حالت runtime داخل google colab بر gpu گذاشته شود تا زمان کمتری برای اجرای کد از ما بگیرد.

حال زمان پیش بینی نتایج یا جواب هایی است کا با توجه به سوالاتمان باید بگیریم. برای این کار مدل مان را استنباط یا inference می کنیم. در ابتدا سوال را به عنوان ورودی و خروجی ها را در حالت LSTM که در حالت های h و c بود را به encoder وارد می کنیم. برای decoder هم به دو مورد نیاز داریم، که یک حالت های LSTM و دیگری توالی ورودی جواب های ما هستند که جواب مورد نیاز ما با توجه به سوال ورودی را می دهد. در کد زیر این عملیات قابل مشاهده است:

```
#Creating an inference model that can help predict the answer
def make_inference_models():

    #Taking the question as input and outputs LSTM states for encoder model
    encoder_model = tf.keras.models.Model(encoder_inputs, encoder_states)

    #Giving LSTM states and the answer input seqeunces to decoder model
    decoder_state_input_h = tf.keras.layers.Input(shape=(200, ))
    decoder_state_input_c = tf.keras.layers.Input(shape=(200, ))

    decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]

    decoder_outputs, h, c = decoder_lstm(decoder_embedding , initial_state=decoder_states_inputs)
    decoder_states = [h, c]
    decoder_outputs = decoder_dense(decoder_outputs)
    decoder_model = tf.keras.models.Model([decoder_inputs] + decoder_states_inputs,[decoder_outputs] + decoder_states)

    return encoder_model , decoder_model
```

حال که تمام عملیات مورد نیاز برای انجام فرایند chatbot نیاز داریم که شروع به صحبت و تست کردن آن کنیم. در قطعه کد زیر، قطعه کدی که به ما کمک خواهد کرد که سوال ورودی ما از حالت string تبدیل به توکن هایی در فرم integer شود را خواهیم دهید:

```
#Turning questions from string to tokens using padding
def str_to_tokens(sentence : str):
    words = sentence.lower().split()
    tokens_list = list()
    #Getting the tokens list of words we have in question
    for word in words:
        tokens_list.append(tokenizer.word_index[word])
    return preprocessing.sequence.pad_sequences([tokens_list], maxlen=max_questions_len, padding='post')
```

در آخر هم کد هایی داریم که صحبت کردن با bot را برای ما امکان پذیر می کند. این کد به این شکل کار می کند که سوال را وارد enc_model می کنیم که بتوانیم state_values را پیش بینی کنیم و وارد حالت decoder می کنیم که یک توالی به وجود می آید که در آن کلمه ی start موجود است که ان را وارد dec_model می کنیم و state_values را بروز رسانی می کنیم تا به آنجایی برسیم که به end برسیم یا به حد نهایی از طول جواب برسیم. در این زمان است که امکان دارد این مدل ما بعضی کلمات ورودی ما را نشناسد که امکان errorای به وجود آید که در ابتدا اشاره کردیم. برای اینکه bot جوابی به ما بدهد که بگوید متوجه جمله ما نشده است، پیامی هم در انتها قرار گرفته است.

```
enc_model , dec_model = make_inference_models()
try:
  for _ in range(10):
      #Asking the question and predicting state values from question by using enc_model
      states_values = enc_model.predict(str_to_tokens(input('You: ')))
      #Setting the h and c state values
      empty_target_seq = np.zeros((1, 1))
      #Generating sequence that has <start> element
      empty_target_seq[0, 0] = tokenizer.word_index['start']
      stop_condition = False
      decoded_translation = ''
      #Start the process
      while not stop_condition :
          #Replace the <start> element with the element which was predicted before and update the state values by using dec_model
          dec_outputs , h , c = dec_model.predict([empty_target_seq] + states_values)
          sampled_word_index = np.argmax(dec_outputs[0, -1, :])
          sampled_word = None
          for word , index in tokenizer.word_index.items() :
              if sampled_word_index == index :
                  decoded_translation += ' {}'.format(word)
                  sampled_word = word  #Getting the word that is best for the answer
          #Stopping when we hit the <end> tag or the maximum answer length
          if sampled_word == 'end' or len(decoded_translation.split()) > max_answers_len:
              stop_condition = True

          empty_target_seq = np.zeros((1, 1))
          empty_target_seq[0, 0] = sampled_word_index
          states_values = [h, c]
      #Deleting the word "end" at the end of the sentence for printing
      decoded_print = decoded_translation.replace("end", "")

      print(decoded_print)
      print()
#Showing a messege if we hit a keyword error
except KeyError:
    print("Sorry, I did not understand that")
```

تا اینجا شیوه ی کار این کد که به خود با توجه به دیتابیس اش آموزش می دهد را توضیح دادیم. توضیحات مربوط به شیوه کار seq2seq از سایت زیر گرفته شده است:

https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html

حالا به طور مختصر به chotbot دوم که قابلیت های محدود تری دارد می پردازیم. در شکل زیر می توان یک دیالوگ کوتاه را با این chatbot مشاهده کرد:

```
You: hello
Hello!
You: who are you
I didn't get that, Ask something else!
You: what is you name
You can call me Valkyrie
You: what do you do
I'm here to answer your questions
You: what is your job
I'm a Norse God
You:
```

دیتابیس این chatbot بسیار کوتاه تر و محدود تر از مورد قبلی است و شخصا طراحی شده است. برای کد این پروژه هم به طور کامل درون کد، comment گذاری انجام شده است. در این کد دیتابیس ما شامل یک فایل است که در چند حالت مختلف دسته بندی شده است که حالت ها شامل دیالوگ های خوش آمد گویی، خداحافظی و موارد دیگر می شود. در اصل chatbot ما سوالی که پرسیده می شود را می گیرد، با سوالات داخل دیتابیس مقایسه می کند و به طور رندوم از دسته ای که فکر می کرد آن سوال بیشتر تطابق دارد، جوابی را نمایش می دهد.

مدل کار این bot با استفاده از Neural networks یک مجموعه از کلمات یا bag of words را می گیرد (این کلمات با توجه به tag هایی که در دیتابیس ما هستند و موضوعات هر دیالوگ را مشخص می کنند، هستند و تشخیص داده می شوند که ورودی با کدام قسمت دیتابیس مطابقت دارد) وارد مدل ما می کند. عکس بعدی مربوط به کد این قسمت است:

```
#Making a bag of words to get ready to put it in our model
words = [stemmer.stem(w.lower()) for w in words if w != "?"]  #If statement makes it to not care about question mark
words = sorted(list(set(words)))  #Making sure that there are no double words in list

lables = sorted(lables)  #Sorting our lables list

training = []
output = []

out_empty = [0 for _ in range(len(lables))]  #Creating an empty output for putting the words we need to fill it with

#Creating bag of words for our output
for x, doc in enumerate(docs_x):
    bag = []
    wrds = [stemmer.stem(w) for w in doc]

    for w in words:
        if w in wrds:  #If the word exist in pattern we looping through
            bag.append(1) #Showing if that word existes and where it is
        else:
            bag.append(0)

    output_row = out_empty[:]
    output_row[lables.index(docs_y[x])] = 1  #Seeing where the tag we want is and set in our output

    training.append(bag)
    output.append(output_row)
```

مدل ما لایه های مخفی یا hidden layer هایی دارد که عملیات تجزیه و تحلیل در آن انجام می شود و قابلیت softmax در آن فعال است و شامل هشت قسمت ورودی است که هر قسمت به هر یک از bag of words متصل است و احتمال آن که کدام خروجی آن بیشتر به جواب مورد نظر می خورد را به ما می دهد:

```
net = tflearn.input_data(shape=[None, len(training[0])])  #Input data #Finding the input shape that we are expecting for our model
net = tflearn.fully_connected(net, 8)  #Hidden layer #Adding fully connected layer to our network
net = tflearn.fully_connected(net, 8)  #Hidden layer
net = tflearn.fully_connected(net, len(output[0]), activation="softmax")  #Output layer #Getting the probability of each output
net = tflearn.regression(net)

model = tflearn.DNN(net)  #DNN will get the network we made and will use it

#If the model already existes, it won't train the model again
try:
    model.load("model.tflearn")
except:
    model.fit(training, output, n_epoch=1000, batch_size=8, show_metric=True) #Fitting the model or passing all our data #n_epoch
    model.save("model.tflearn")  #Saving our model
```

حال بعد از آموزش دادن مدل، می توانیم ورودی را بگیریم که در اصل همان سوال ما است. در اینجا مدل ما بررسی می کند و می بیند از بین جواب هایی که برای این سوال تعیین کرده است، کدامشان احتمال درست بودن بالای 80 درصد دارد و آن را انتخاب می کند وگرنه می گوید که متوجه پیام ما نشده است:

```
def chat():
    print("Start talking with the bot! (Type 'quit' to stop)")
    while True:
        inp = input("You: ")  #Getting the user's response
        if inp.lower() == "quit":
            break

        results = model.predict([bag_of_words(inp, words)])[0]
        results_index = numpy.argmax(results)  #Giving the greatest index number
        tag = lables[results_index]  #Giving the lable that think the message is

        #Getting the responses we need
        if results[results_index] > 0.8:
            for tg in data["intents"]:
                if tg['tag'] == tag:
                    responses = tg['responses']

            print(random.choice(responses))

        else:
            print("I didn't get that, Ask something else!")
chat()
```