BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC SƯ PHẠM KĨ THUẬT

HCMUTE

BÁO CÁO CUỐI KỲ

ĐỀ TÀI: Chatbot

GVHD: Vũ Quang Huy

SVTH: Nông Quốc Đạt **MSSV: 18146100**

Nguyễn Đức Hoàng **MSSV:18146119**

Huỳnh Nhật Quang **MSSV:18146196**

TP.Hồ Chí Minh tháng 06 năm 2021

1 . Xử lý văn bản đầu vào

1.1 Xóa các ký tự đặc biệt dấu câu

Sử dụng **Regular Expression** để xử lý văn bản ký tự thừa

```
import re
```

```
def xoa_ky_tu_dac_biet(sentence):
  sentence = sentence.lower() # Bỏ ghi in hoa
  sentence = re.sub(r'[.,~`!@#$%\^&*\(\)\[\]\\|:;\'"]+', ' ', sentence)
  sentence = re.sub(r'\s+', ' ', sentence).strip()
  return sentence
```

Thử nghiệm

```
s = 'Sư Phạm@^ Kỹ Thuật@:.'
x = xoa_ky_tu_dac_biet(s)
print(s)
print(x)
```

```
Sư Phạm@^ Kỹ Thuật@:.
sư phạm kỹ thuật
```

Để tối ưu chatbot , thêm vào thuật toán dự đoán dấu câu văn bản khi mà văn bản đầu vào không có dấu

Ý tưởng thực hiện

1. Từ câu tiếng Việt không dấu, mình sẽ sinh ra tất cả các khả năng có thể đánh dấu cho câu đó.
2. Sử dụng mô hình ngôn ngữ (language model) để xác định điểm số (phân bố xác suất) của các từ, chuỗi từ.
3. Áp dụng thuật toán beam search để tìm kiếm kết chuỗi từ (đã được thêm dấu câu) phù hợp nhất.

Vì việc thêm dấu câu chỉ áp dụng với các từ tiếng Việt (bao gồm các nguyên âm u, e, o, a, i, y và phụ âm d). Do đó, mình sẽ xây dựng bộ từ điển để lưu tất cả các khả năng đánh dấu câu của một từ tiếng Việt bất kỳ theo dạng này:

Đầu tiên mình sẽ bỏ xóa bỏ dấu câu của câu đầu vào bằng **Regular Expression** giống như xóa ký tự đặc biệt như trên

```
[15] def xoa_dau_tieng_viet(s):
         s = re.sub('[áàảãạăắằẳẵặâấầẩẫậ]', 'a', s)
         s = re.sub('[éèẻẽẹêếềểễệ]', 'e', s)
         s = re.sub('[óòỏõọôốồổỗộơớờởỡợ]', 'o', s)
         s = re.sub('[íìỉĩị]', 'i', s)
         s = re.sub('[úùủũụưứừửữự]', 'u', s)
         s = re.sub('[ýỳỷỹỵ]', 'y', s)
         s = re.sub('đ', 'd', s)
         return s
```

Kết quả mình thu được hàm xóa dấu như thế này :

```
[16] s = 'Trí Tuệ Nhân Tạo'
     x = xoa_dau_tieng_viet(s)
     print(s)
     print(x)

     Trí Tuệ Nhân Tạo
     Tri Tue Nhan Tao
```

Ý tưởng: Từ bộ từ điển đầy đủ tất cả các từ tiếng Việt đơn âm tiết *nguphamvietnam.txt* , mình sẽ gom lại thành một cụm nếu các từ đó đem xóa dấu là các từ giống nhau, code thì đơn giản như thế này thôi:

```
map_accents = {}
for word in open('nguphapvietnam.txt', encoding="utf8").read().splitlines():
  word = word.lower()                              # Bỏ viết hoa
  no_accent_word = xoa_dau_tieng_viet(word)        # Xóa dấu
  if no_accent_word not in map_accents:
    map_accents[no_accent_word] = set()
  map_accents[no_accent_word].add(word)
```

Kết quả mình sẽ thu được bộ từ điển đầy đủ như thế này

```
map_accents
```

```
 'khén',
 'khẽn',
 'khến',
 'khền',
 'khển',
 'khễn',
 'khện'},
'kheng': {'kheng', 'khèng', 'khéng', 'khẹng', 'khẻng', 'khẽng'},
'kheo': {'kheo', 'khèo', 'khéo', 'khẹo', 'khẻo', 'khẽo'},
'khi': {'khi', 'khì', 'khí', 'khĩ', 'khỉ', 'khị'},
'khia': {'khia', 'khìa', 'khía', 'khĩa', 'khỉa', 'khịa'},
'khim': {'khim', 'khìm', 'khím', 'khĩm', 'khỉm', 'khịm'},
'khin': {'khin', 'khìn', 'khín', 'khĩn', 'khỉn', 'khịn'},
'khinh': {'khinh', 'khình', 'khính', 'khĩnh', 'khỉnh', 'khịnh'},
'khiu': {'khiu', 'khìu', 'khíu', 'khĩu', 'khỉu', 'khịu'},
'khiem': {'khiêm', 'khiếm', 'khiềm', 'khiểm', 'khiễm', 'khiệm'},
'khien': {'khiên', 'khiến', 'khiền', 'khiển', 'khiễn', 'khiện'},
'khieng': {'khiêng', 'khiếng', 'khiềng', 'khiểng', 'khiễng', 'khiệng'},
'khieu': {'khiêu', 'khiếu', 'khiều', 'khiểu', 'khiễu', 'khiệu'},
'khiec': {'khiếc', 'khiệc'},
'khiep': {'khiếp', 'khiệp'},
'khiet': {'khiết', 'khiệt'},
'kho': {'kho',
 'khò',
 'khó',
 'khô',
 'khõ',
 'khơ',
 'khọ',
 'khỏ',
 'khố',
 'khồ',
 'khổ',
```

Thử kiểm tra vài từ xem sao :

```
[24] print(map_accents['xin'])
     print(map_accents['loi'])
```

```
{'xìn', 'xĩn', 'xín', 'xịn', 'xin', 'xỉn'}
{'lời', 'lỗi', 'lòi', 'lơi', 'lội', 'lới', 'lợi', 'lõi', 'lỗi', 'lói', 'lọi', 'lôi', 'loi', 'lời', 'lời', 'lối', 'lòi', 'lỗi'}
```

Phân tích dữ liệu văn bản tiengviettonghop.txt để dự đoán các từ hợp với ngữ cảnh .

```
lm = {}
for line in open('thuvientiengviettonghop.txt', encoding="utf8"):
  data = json.loads(line)
  key = data['s']
  lm[key] = data
vocab_size = len(lm)
total_word = 0
for word in lm:
  total_word += lm[word]['sum']
```

Ta có được thư viện lm như thế này

```
lm['hoàng']
```

```
{'next': {'sa': 116350,
  'anh': 93785,
  'văn': 84460,
  'gia': 84221,
  'mai': 71188,
  'tử': 69543,
  'thị': 42827,
  'yến': 40768,
  'thùy': 39549,
  'trung': 36978,
```

Có nghĩa là đứng sau từ “hoàng” nhiều nhất trong tệp dữ liệu là từ “sa” với 116350 lần , tiếp sau đó là từ “anh” với 93758 lần v.v….

Tính xác suất một từ trong chuỗi từ

```
[43] def xstk(current_word, next_word):
       if current_word not in lm:
         return 1 / total_word;
       if next_word not in lm[current_word]['next']:
         return 1 / (lm[current_word]['sum'] + vocab_size)
       return (lm[current_word]['next'][next_word] + 1) / (lm[current_word]['sum'] + vocab_size)
```

Do đây là mô hình ngôn ngữ n-gram với n = 2, nên mình sẽ tính xác suất có điều kiện của 1 từ (w2) chỉ phụ thuộc vào 1 từ trước nó (w1):

$$P(w1, w2) = Count(w1w2) / Count(w1)$$

Ví dụ với từ “hoàng sa” P(hoàng, sa) = Count(“hoàng sa”) / Count(“hoàng”) = 0.054111534377852925 ≈ 5.41%

```
xstk('hoàng','sa')

0.054111534377852925
```

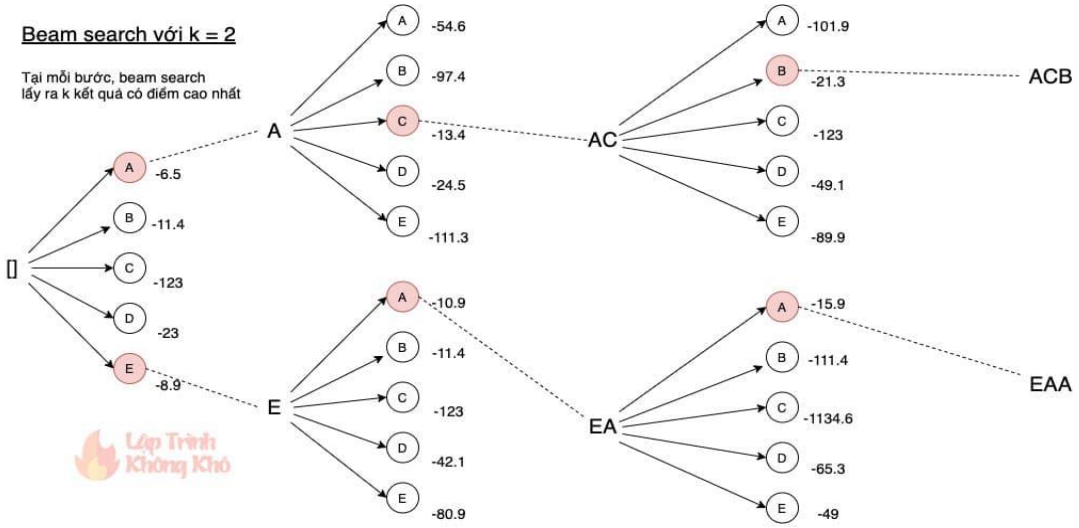
Tương tự với từ “hoàng anh”

```
[46] xstk('hoàng','anh')

     0.04361719592578761
```

2 . Tìm cách đánh dấu câu tốt nhất

Giải thuật toán kiếm beam search cũng khởi đầu với chuỗi rỗng. Tại mỗi bước, nó thực hiện tìm kiếm toàn bộ trên không gian của bước đó và lấy ra k kết quả có điểm số cao nhất thay vì chỉ lấy 1 kết quả cao nhất. Hình ảnh dưới đây sẽ cho bạn thấy rõ nhất cách hoạt động của beam search với k=2.



Chúng ta sẽ xây dựng một hàm tìm kiếm beam search để tìm ra cách đánh dấu câu phù hợp nhất. Hàm sẽ nhận vào chuỗi từ không dấu và tham số k. Tại mỗi bước lặp, từ hiện tại sẽ được ghép cặp với tất cả các khả năng đánh dấu của từ phía sau nó. Ở mỗi bước đó thì điểm số của chuỗi từ sẽ được tính bằng cách nhân các xác suất với nhau và chỉ giữ lại (lấy) k chuỗi từ có điểm số cao nhất. Quá trình tiếp tục cho tới khi kết thúc câu.

Do mô hình ngôn ngữ mình dùng thiếu token bắt đầu, từ đầu tiên sẽ không có phân bố xác suất 2-gram nên bước đầu tiên mình lấy tất cả các khả năng. Nói là tất cả nhưng không nhiều đâu, làm vậy cho chính xác hơn!

Do giá trị xác suất của các cặp từ là các giá trị rất bé, nên nếu ta nhân chúng với nhau liên tục sẽ có thể bị tràn số. Để khắc phục vấn đề này, người ta thường sử dụng hàm log để giữ cho giá trị xác suất đủ lớn.

```python
def beam_search(words1, k=3):
  ss = []
  for idx, word in enumerate(words1):
    if idx == 0:
      ss = [([x], 0.0) for x in map_accents.get(word, [word])]
    else:
      all_sequences = []
      for seq in ss:
        for next_word in map_accents.get(word, [word]):
          current_word = seq[0][-1]
          proba = xstk(current_word, next_word)
          # print(current_word, next_word, proba, log(proba))
          proba = log(proba)
          new_seq = seq[0].copy()
          new_seq.append(next_word)
          all_sequences.append((new_seq, seq[1] + proba))
      # print(all_sequences)
      all_sequences = sorted(all_sequences,key=lambda x: x[1], reverse=True)
      ss = all_sequences[:k]
  return ss
```

Thử kết quả với một đoạn văn bản với số k = 5`

```
s = 'dai hoc su pham ky thuat'
x = xoa_ky_tu_dac_biet(s)
y = xoa_dau_tieng_viet(x)
z = y.split()
w = beam_search(z, k=5)
inp = ' '.join(w[0][0])
print(s)
print(inp)
```

```
dai hoc su pham ky thuat
đại học sư phạm kỹ thuật
```

Khi đã xử lý được văn bản đầu vào , bây giờ tiến hành xây dựng hệ thống chatbot

3 . Ý tưởng thực hiện

Xây dựng một chatbot bằng cách sử dụng các kỹ thuật học sâu. Chatbot sẽ được đào tạo về tập dữ liệu chứa các danh mục (ý định), mẫu và phản hồi. Sử dụng mạng nơ-ron (ANN) để phân loại thông điệp của người dùng thuộc danh mục nào và sau đó chúng tôi sẽ đưa ra phản hồi ngẫu nhiên từ danh sách phản hồi.

Tập dữ liệu training cho chatbox JSON

```
{
  "intents": [
    {
      "tag": "Lời chào",
      "patterns": [
        "Hi",
        "Xin chào",
        "Chào bạn",
        "chào cậu"
      ],
      "responses": [
        "Xin chào mình là bot trả lời tự động"
      ],
      "context_set": ""
    },
    {
      "tag": "Tạm biệt",
      "patterns": [
        "tạm biệt",
        "bye bye",
        "gặp lại sau",
        "bye"
      ],
      "responses": [
        "Goodbye"
      ],
      "context_set": ""
    }

  ]
}
```

Với tag: thẻ nhãn loại câu hỏi

Patterns: Dữ liệu câu hỏi

Responses: Dữ liệu câu trả lời của bot

Tải và đọc tệp JSON

```
with open("training.json",encoding="utf8") as file:
    data = json.load(file)
```

Trích xuất dữ liệu từ file JSON , đầu tiên gọi 4 chuỗi rỗng tương ứng với 4 lớp files JSON

```
words = []
labels = []
docs_x = []
docs_y = []
```

Dùng vòng lặp for để trích xuất với từng lớp , trích xuất từng từ riêng biệt bằng cách dùng nltk.word_tokenize

```
for intent in data["intents"]:
    for pattern in intent["patterns"]:
        wrds = nltk.word_tokenize(pattern)
        words.extend(wrds)
        docs_x.append(wrds)
        docs_y.append(intent["tag"])

        if intent["tag"] not in labels:
            labels.append(intent["tag"])
```

Như chúng ta đã biết mạng nơ-ron và các thuật toán học máy yêu cầu đầu vào là số. Vì vậy, trong danh sách các chuỗi sẽ không cắt nó. Chúng ta cần một số cách để biểu diễn các câu của chúng ta bằng các con số và đây là nơi chứa nhiều từ. Việc chúng ta sẽ làm là biểu diễn mỗi câu bằng danh sách độ dài của lượng từ trong từ vựng mô hình của chúng ta. Mỗi vị trí trong danh sách sẽ đại diện cho một từ trong vốn từ vựng của chúng ta. Nếu vị trí trong danh sách là 1 thì điều đó có nghĩa là từ đó tồn tại trong câu của chúng ta, nếu là 0 thì từ đó cũng không có mặt. Đây là một túi từ vì thứ tự xuất hiện của các từ trong câu bị mất .

Cũng như định dạng đầu vào, chúng ta cần định dạng đầu ra của mình để phù hợp với mạng nơ-ron. Tương tự như vậy với một loạt các từ, chúng tôi sẽ tạo danh sách đầu ra là độ dài của số lượng nhãn / thẻ mà chúng tôi có trong tập dữ liệu của mình. Mỗi vị trí trong danh sách sẽ đại diện cho một nhãn / thẻ riêng biệt, số 1 trong bất kỳ vị trí nào trong số đó sẽ hiển thị nhãn / thẻ nào được đại diện.

```
words = [stemmer.stem(w.lower()) for w in words if w != "?"]
words = sorted(list(set(words)))
labels = sorted(labels)
training = []
output = []

out_empty = [0 for _ in range(len(labels))]

for x, doc in enumerate(docs_x):
    bag = []

    wrds = [stemmer.stem(w) for w in doc]

    for w in words:
        if w in wrds:
            bag.append(1)
        else:
            bag.append(0)

    output_row = out_empty[:]
    output_row[labels.index(docs_y[x])] = 1

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

Cuối cùng, chuyển đổi dữ liệu đào tạo và đầu ra của chúng tôi thành các mảng numpy.

```
training = numpy.array(training)
output = numpy.array(output)
```

4 . Phát triển một mô hình

Bây giờ đã xử lý trước tất cả dữ liệu của mình, bắt đầu tạo và đào tạo một mô hình. Chúng ta sẽ sử dụng một mạng nơ-ron chuyển tiếp khá tiêu chuẩn với hai lớp ẩn. Mục tiêu của mạng sẽ là xem xét một loạt các từ và đưa ra một lớp mà chúng cũng thuộc về (một trong các thẻ của chúng tôi từ tệp JSON).

Bắt đầu bằng cách xác định kiến trúc của mô hình của chúng ta .

```
ops.reset_default_graph()

net = tflearn.input_data(shape=[None, len(training[0])])
net = tflearn.fully_connected(net, 8)
net = tflearn.fully_connected(net, 8)
net = tflearn.fully_connected(net, len(output[0]), activation="softmax")
net = tflearn.regression(net)

model = tflearn.DNN(net)
```

5 . Đào tạo & Lưu mô hình

Bây giờ chúng ta đã thiết lập mô hình của mình, đã đến lúc đào tạo nó trên dữ liệu ! Để làm được điều này ta sẽ "fit" dữ liệu của ta cho mô hình. Số kỷ nguyên mà chúng tôi đặt là số lần mà mô hình sẽ nhìn thấy cùng một thông tin trong khi đào tạo với số lần đào tạo n_epoch = 1000

```
model.fit(training, output, n_epoch=1000, batch_size=8, show_metric=True)
model.save("model.tflearn")
```

Sau khi huấn luyện xong mô hình, lưu mô hình đó vào tệp **model.tflearn** để sử dụng trong các tập lệnh khác.

6 . Đưa ra dự đoán

Bây giờ đã đến lúc thực sự sử dụng mô hình! Lý tưởng nhất là chúng ta muốn tạo phản hồi cho bất kỳ câu nào mà người dùng nhập vào. Để làm được điều này, chúng ta cần nhớ rằng mô hình của chúng ta không sử dụng đầu vào chuỗi, nó cần một túi từ. Chúng ta cũng cần nhận ra rằng mô hình của chúng ta không tạo ra các câu, nó tạo ra một danh sách các xác suất cho tất cả các lớp của chúng ta. Điều này làm cho quá trình tạo phản hồi giống như sau:

- Nhận một số thông tin đầu vào từ người dùng
- Chuyển nó thành một nhóm từ
- Nhận dự đoán từ mô hình
- Tìm lớp có khả năng xảy ra cao nhất
- Chọn phản hồi từ lớp đó

```python
def bag_of_words(s, words):
    bag = [0 for _ in range(len(words))]

    s_words = nltk.word_tokenize(s)
    s_words = [stemmer.stem(word.lower()) for word in s_words]

    for se in s_words:
        for i, w in enumerate(words):
            if w == se:
                bag[i] = 1

    return numpy.array(bag)


def chat():
    print("Start talking with the bot (type quit to stop)!")
    while True:
        inp = input("You: ")
        if inp.lower() == "quit":
            break

        results = model.predict([bag_of_words(inp, words)])
        results_index = numpy.argmax(results)
        tag = labels[results_index]

        for tg in data["intents"]:
            if tg['tag'] == tag:
                responses = tg['responses']

        print(random.choice(responses))

chat()
```

Kết quả training với chỉ số đào tạo n_epoch = 1000 số lớp train 40

```
Training Step: 4996  | total loss: 1.11851 | time: 0.003s
| Adam | epoch: 1000 | loss: 1.11851 - acc: 0.9097 -- iter: 08/40
Training Step: 4997  | total loss: 1.03812 | time: 0.006s
| Adam | epoch: 1000 | loss: 1.03812 - acc: 0.9187 -- iter: 16/40
Training Step: 4998  | total loss: 0.94754 | time: 0.010s
| Adam | epoch: 1000 | loss: 0.94754 - acc: 0.9269 -- iter: 24/40
Training Step: 4999  | total loss: 0.86601 | time: 0.013s
| Adam | epoch: 1000 | loss: 0.86601 - acc: 0.9342 -- iter: 32/40
Training Step: 5000  | total loss: 0.82179 | time: 0.016s
| Adam | epoch: 1000 | loss: 0.82179 - acc: 0.9408 -- iter: 40/40
```

Ta thấy mô hình độ chính xác khá cao = 0.9408 (94.08%)
Bây giờ chúng ta sẽ thử test một câu

```
You: chào bạn
[2.8339453e-02 6.8668932e-02 1.7637786e-02 1.2186435e-03 6.1449301e-01
 4.0231179e-02 1.9163817e-01 1.1839450e-02 1.8284735e-03 1.5958598e-04
 3.9727185e-03 1.9876845e-02 9.5806528e-05]
Bot Dz: Chào
```

Ở đây tôi đã train có 13 lớp câu hỏi , hãy xem kết quả tất cả các lớp câu hỏi và thấy từ “chào bạn” có tỉ lệ gần lớp thứ 5 nhất 0.6144930 (61.44 %)

Như vậy mô hình xác định câu “chào bạn” gần với phân lớp câu hỏi thứ 5 , và từ đó lấy câu trả lời của phân lớp 5 để print

Bây giờ thay vì xác định lớp của câu hỏi để đưa ra câu trả lời , ta sẽ dùng để làm nó như một trợ lý ảo .

Ví dụ , khi chatbot xác định một tag: “Youtube” nó sẽ thực hiện lệnh mở trình duyệt Youtube , chúng ta có thể phát triển thêm như hỏi muốn tìm kiếm gì trên youtube

Ta sẽ viết thêm 3 lớp câu hỏi vào JSON


```
{
  "intents": [
    {
      "tag": "Google",
      "patterns": [
        "Google",
        "Tìm kiếm Google",
        "Mở Google"
      ],
      "responses": [
        "Google"
      ],
      "context_set": ""
    },
    {
      "tag": "Ytb",
      "patterns": [
        "youtube",
        "tìm kiếm youtube",
        "mở youtube",
        "xem video",
        "tôi muốn xem video"
      ],
      "responses": [
        "Youtube"
      ],
      "context_set": ""
    },
    {
      "tag": "fb",
      "patterns": [
        "facebook",
        "fb",
        "mở facebook",
        "mở fb"
      ],
      "responses": [
        "Facebook"
      ],
      "context_set": ""
    },
    {
```

Khi kết qua responses trả về ‘Google’ , ‘Youtube’ , ‘Facebook’ thì ta sẽ dùng lệnh để mở các trình duyệt này lên

```
if results[results_index] > 0.5:
    for tg in data["intents"]:
        if tg['tag'] == tag:
            responses = tg['responses']
            hoang = random.choice(responses)
            if hoang == 'Google':
                inp3 = input("Bot Dz: Bạn muốn tìm kiếm gì ? \nYou: ")
                url=f"https://www.google.com/search?q="+inp3
                wb.get().open(url)
            if hoang == 'Youtube':
                inp3 = input("Bot Dz: Bạn muốn tìm kiếm gì ? \nYou: ")
                url=f"https://www.youtube.com/search?q="+inp3
                wb.get().open(url)
            if hoang == 'Facebook':
                url=f"https://www.facebook.com/"
                wb.get().open(url)
```

Tương tự ta cũng có thể thêm 1 lớp để mở 1 file hoặc 1 ứng dụng trên máy tính bằng cách truy cập đường link file đó

```
if hoang == 'Ảnh':
    link = r"C:\Users\PC\Pictures\poster.png"
    os.startfile(link)
```

7 . Hệ thống tự training

Đây như một phiên bản dành cho người muốn đào tạo chatbot mà không cần phải chỉnh sửa thêm trên file JSON

Ý tưởng : Khi một câu hỏi đầu vào tỉ lệ dưới 0.5 (50%) đối với tất cả các phân lớp câu hỏi , nghĩa là không có phân lớp nào gần giống với câu hỏi . Máy sẽ chuyển sang chế độ ttheemcaau hỏi đó vào file JSON và bạn cần nhập thêm vào câu trả lời . Sau đó câu hỏi và trả lời của bạn sẽ tự update vào file JSON gốc .

```
# Cập nhật thêm câu hỏi và trả lời vào tệp để training
def update_json(filepath, var1, var2):
    now = datetime.datetime.now() # Gọi timenow đặt tên cho tag
    t = now.strftime("%y-%m-%d %H:%M:%S")
    with open(filepath,'r', encoding='utf-8') as fp:
        information = json.load(fp)
    information["intents"].append({
        "tag": t,
        "patterns": [var1],
        "responses": [var2],
        "context_set": ""
    })
```

Đây là chương trình chính:

```
def chat():
    print("Noi gi di may?")
    while True:
        inp = input("You: ")
        if inp.lower() == "quit":
            break

        sentence = xoa_ky_tu_dac_biet(inp)
        _sentence = xoa_dau_tieng_viet(sentence)
        words1 = _sentence.split()
        results1 = beam_search(words1, k=5)
        inp = ' '.join(results1[0][0])


        results = model.predict([bag_of_words(inp, words)])[0]
        results_index = numpy.argmax(results)
        tag = labels[results_index]
        print(results)

        if results[results_index] > 0.5:
            for tg in data["intents"]:
                if tg['tag'] == tag:
                    responses = tg['responses']
                    hoang = random.choice(responses)
                    if hoang == 'Google':
                        inp3 = input("Bot Dz: Bạn muốn tìm kiếm gì ? \nYou: ")
                        url=f"https://www.google.com/search?q="+inp3
                        wb.get().open(url)
                    if hoang == 'Youtube':
                        inp3 = input("Bot Dz: Bạn muốn tìm kiếm gì ? \nYou: ")
                        url=f"https://www.youtube.com/search?q="+inp3
                        wb.get().open(url)
                    if hoang == 'Facebook':
                        url=f"https://www.facebook.com/"
                        wb.get().open(url)
                    if hoang == 'Ảnh':
                        link = r"C:\Users\PC\Pictures\poster.png"
                        os.startfile(link)

            print("Bot Dz: "+random.choice(responses))
        else:
            inp2 = input("Hãy nhập câu trả lời: ")
            update_json("training.json",inp,inp2)


chat()
```