TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN 2**

**TÌM HIỂU VÀ XÂY DỰNG ỨNG DỤNG DỊCH TỰ ĐỘNG ANH – VIỆT DỰA TRÊN THƯ VIỆN TENSORFLOW**

*Người hướng dẫn*: **TS NGUYỄN CHÍ THIỆN**

*Người thực hiện*: **NGUYỄN ĐOÀN DUY – 51403202**

**NGUYỄN KHẮC NHẬT - 51403099**

Lớp **: 14050303**

Khoá  **: 18**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2018**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN 2**

**TÌM HIỂU VÀ XÂY DỰNG ỨNG DỤNG DỊCH TỰ ĐỘNG ANH – VIỆT DỰA TRÊN THƯ VIỆN TENSORFLOW**

*Người hướng dẫn*: **TS NGUYỄN CHÍ THIỆN**

*Người thực hiện*: **NGUYỄN ĐOÀN DUY – 51403202**

**NGUYỄN KHẮC NHẬT - 51403099**

Lớp **: 14050303**

Khoá  **: 18**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2018**

LỜI CẢM ƠN

Chúng em xin cảm ơn thầy Thiện đã hỗ trợ để chúng em có thể hoàn thành đồ án 2 một cách tốt nhất.

**ĐỒ ÁN ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là sản phẩm đồ án của riêng chúng tôi và được sự hướng dẫn của thầy Nguyễn Chí Thiện. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình.** Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày tháng năm*

*Tác giả*

*(ký tên và ghi rõ họ tên)*

*Nguyễn Đoàn Duy*

*Nguyễn Khắc Nhật*

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

**Phần xác nhận của GV hướng dẫn**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

**Phần đánh giá của GV chấm bài**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

Mục Lục

[CHƯƠNG I: CƠ SỞ LÝ THUYẾT 2](#_Toc515617630)

[1.1 DỊCH MÁY (MACHINE TRANSLATION) 2](#_Toc515617631)

[1.1.1 Dịch tự động là gì? 2](#_Toc515617632)

[1.1.2 Khó khăn 2](#_Toc515617633)

[1.1.3 Hạn chế 3](#_Toc515617634)

[1.2 THƯ VIỆN TENSORFLOW 4](#_Toc515617635)

[1.2.1 Tensorflow là gì ? 4](#_Toc515617636)

[1.2.2 Khái niệm cơ bản trong Tensorflow 4](#_Toc515617637)

[1.3 NEURAL MACHINE TRANSLATION 8](#_Toc515617638)

[1.3.1 Mạng hồi quy RNN(Recurrent Neural Network) 8](#_Toc515617639)

[1.3.2 Áp dụng RNN để dịch 11](#_Toc515617640)

[CHƯƠNG II: XÂY DỰNG ỨNG DỤNG DỊCH MÁY VỚI TENSORFLOW 14](#_Toc515617641)

[2.1 Import thư viện: 14](#_Toc515617642)

[2.2 Tải dữ liệu: 14](#_Toc515617643)

[2.3 Tokenizer 15](#_Toc515617644)

[2.4 Tạo RNN 16](#_Toc515617645)

[2.4.1 Tạo phần mã hóa 16](#_Toc515617646)

[2.4.2 Tạo phần giải mã 17](#_Toc515617647)

[2.4.3 Kết nối và tạo mô hình 19](#_Toc515617648)

[2.4.4 Huấn luyện mô hình 19](#_Toc515617649)

[2.4.5 Tiến hành dịch 20](#_Toc515617650)

[2.4.6 Thực nghiệm dịch 21](#_Toc515617651)

[2.4.7 Hướng dẫn sử dụng 21](#_Toc515617652)

[TÀI LIỆU THAM KHẢO 27](#_Toc515617653)

CHƯƠNG I: CƠ SỞ LÝ THUYẾT

1.1 DỊCH MÁY (MACHINE TRANSLATION)

1.1.1 Dịch tự động là gì?

Dịch tự động  hay còn gọi là dịch máy( machine translation) là một nhánh của [xử lý ngôn ngữ tự nhiên](https://vi.wikipedia.org/wiki/X%E1%BB%AD_l%C3%BD_ng%C3%B4n_ng%E1%BB%AF_t%E1%BB%B1_nhi%C3%AAn) thuộc phân ngành [trí tuệ nhân tạo](https://vi.wikipedia.org/wiki/Tr%C3%AD_tu%E1%BB%87_nh%C3%A2n_t%E1%BA%A1o), nó là sự kết hợp giữa [ngôn ngữ](https://vi.wikipedia.org/wiki/Ng%C3%B4n_ng%E1%BB%AF), [dịch thuật](https://vi.wikipedia.org/wiki/D%E1%BB%8Bch_thu%E1%BA%ADt) và [khoa học máy tính](https://vi.wikipedia.org/wiki/Khoa_h%E1%BB%8Dc_m%C3%A1y_t%C3%ADnh). Như tên gọi, dịch tự động thực hiện dịch một ngôn ngữ này (gọi là ngôn ngữ nguồn) sang một hoặc nhiều ngôn ngữ khác (gọi là ngôn ngữ đích) một cách tự động, không có sự can thiệp của con người trong quá trình dịch.

1.1.2 Khó khăn

Khó khăn của việc thiết kế chương trình dịch tự động là làm sao khử [nhập nhằng](https://vi.wikipedia.org/wiki/Nh%E1%BA%ADp_nh%E1%BA%B1ng_(x%E1%BB%AD_l%C3%BD_ng%C3%B4n_ng%E1%BB%AF_t%E1%BB%B1_nhi%C3%AAn)) hiệu quả. Nói về tính nhập nhằng, đây là khái niệm chỉ tính không rõ ràng của ngôn ngữ, chẳng hạn khi viết từ đường kính thì vẫn chưa rõ là nó chỉ một loại "chất ngọt dùng để pha làm đồ uống" hay là "đoạn thẳng đi qua tâm và nối hai điểm của đường tròn, của mặt cầu". Nhập nhằng như ví dụ vừa rồi là do hiện tượng đồng âm (hoặc đồng tự) gây ra, một số kiểu nhập nhằng khác như nhập nhằng [từ loại](https://vi.wikipedia.org/wiki/T%E1%BB%AB_lo%E1%BA%A1i), nhập nhằng [từ đa nghĩa](https://vi.wikipedia.org/wiki/T%E1%BB%AB_%C4%91a_ngh%C4%A9a). Khi dịch tự động từ [tiếng Việt](https://vi.wikipedia.org/wiki/Ti%E1%BA%BFng_Vi%E1%BB%87t) sang [tiếng Anh](https://vi.wikipedia.org/wiki/Ti%E1%BA%BFng_Anh), khó khăn đầu tiên lại là việc xác định ranh giới từ, không giống như tiếng Anh (và nhiều ngôn ngữ khác) mỗi từ đã mang trọn vẹn một nghĩa và được xác định ranh giới qua khoảng trắng, tiếng Việt là [ngôn ngữ đơn lập](https://vi.wikipedia.org/wiki/Ng%C3%B4n_ng%E1%BB%AF_%C4%91%C6%A1n_l%E1%BA%ADp) do vậy có rất nhiều [từ ghép](https://vi.wikipedia.org/w/index.php?title=T%E1%BB%AB_gh%C3%A9p&action=edit&redlink=1), nếu không xác định đúng sẽ xuất hiện kiểu dịch từng từ rồi ghép lại với nhau.

Với tiếng Anh do là [ngôn ngữ biến hình](https://vi.wikipedia.org/w/index.php?title=Ng%C3%B4n_ng%E1%BB%AF_bi%E1%BA%BFn_h%C3%ACnh&action=edit&redlink=1) nên việc xác định [từ loại](https://vi.wikipedia.org/wiki/T%E1%BB%AB_lo%E1%BA%A1i) dễ hơn, ngoài ra nó cũng ít [từ đồng tự](https://vi.wikipedia.org/w/index.php?title=T%E1%BB%AB_%C4%91%E1%BB%93ng_t%E1%BB%B1&action=edit&redlink=1) (hai từ có ký tự hoàn toàn giống nhau nhưng mang nghĩa khác nhau), còn về từ đa nghĩa tiếng Anh cũng như tiếng Việt và hầu như tất cả các ngôn ngữ khác đều rất phức tạp, muốn xác định nghĩa chính xác phải phải thực hiện phân tích [văn cảnh](https://vi.wikipedia.org/w/index.php?title=V%C4%83n_c%E1%BA%A3nh&action=edit&redlink=1).

Sau khi xác định nghĩa phù hợp của từ, công việc tiếp theo là sắp xếp để tạo thành câu hoàn chỉnh. Nếu hai ngôn ngữ có cấu trúc càng khác nhau bao nhiêu công việc này càng phức tạp bấy nhiêu, với những ngôn ngữ gần nhau như tiếng Anh và [tiếng Pháp](https://vi.wikipedia.org/wiki/Ti%E1%BA%BFng_Ph%C3%A1p) công việc tương đối đơn giản, nhưng giữa tiếng Pháp và [tiếng Trung](https://vi.wikipedia.org/wiki/Ti%E1%BA%BFng_Trung_Qu%E1%BB%91c) thì rất khó khăn. Để sắp xếp người ta đưa vào các cấu trúc [ngữ pháp](https://vi.wikipedia.org/wiki/Ng%E1%BB%AF_ph%C3%A1p) hết sức phức tạp, áp dụng nhiều kiến thức toán học nhưng thực tế cho thấy hiệu quả của chúng vẫn không được tốt.

1.1.3 Hạn chế

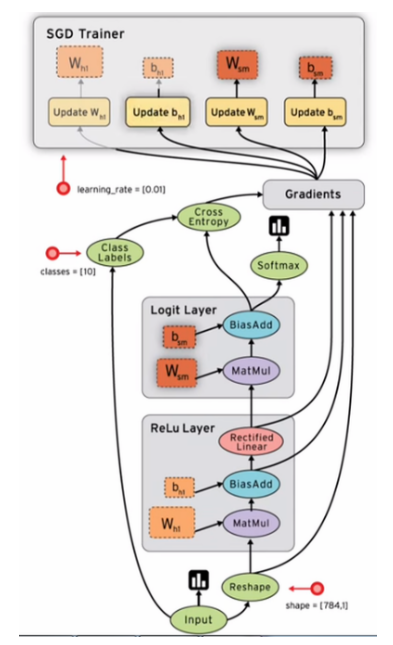
Do chất lượng chưa thật tốt nên hầu hết các sản phẩm dịch tự động đều chỉ mang tính tham khảo, các bản dịch chỉ cho biết đại ý và nó hoàn toàn có thể dịch sai một phần hoặc toàn bộ nội dung cốt lõi của văn bản.

Trong quá trình dịch thuật nếu lạm dụng dịch tự động sẽ làm ảnh hưởng nghiêm trọng đến văn phong, bởi vì người dịch giữ nguyên cách hành văn của bản dịch nhưng cách hành văn này thường không chính xác, máy móc và thiếu "chất người". So sánh giữa các thể loại văn bản khác nhau cho thấy dịch tự động dịch tài liệu chuyên ngành có chất lượng tốt nhất, nguyên nhân là vì các tài liệu này từ vựng có nghĩa rõ ràng (đơn nghĩa), cấu trúc ngữ pháp mạch lạc, đơn giản, ngược lại thể loại [văn học](https://vi.wikipedia.org/wiki/V%C4%83n_h%E1%BB%8Dc) là khó dịch nhất vì từ thường đa nghĩa, nhiều [khẩu ngữ](https://vi.wikipedia.org/w/index.php?title=Kh%E1%BA%A9u_ng%E1%BB%AF&action=edit&redlink=1), cấu trúc ngữ pháp phức tạp, hay sử dụng [nghĩa bóng](https://vi.wikipedia.org/w/index.php?title=Ngh%C4%A9a_b%C3%B3ng&action=edit&redlink=1).

1.2 THƯ VIỆN TENSORFLOW

1.2.1 Tensorflow là gì ?

Tensorflow là một thư viện mã nguồn mở cung cấp khả năng xử lí tính toán số học dựa trên biểu đồ mô tả sự thay đổi của dữ liệu. Tensor được sử dụng khi bạn cần giải quyết các bài toán supervised learning.



1.2.2 Khái niệm cơ bản trong Tensorflow

Khi thực hành với Tensorflow, sẽ có rất nhiều khái niệm phức tạp. Tuy nhiên chỉ ở mức cơ bản, khái niệm quan trọng nhất trong Tensorflow là **Tensor**.

1.2.2.1 Node

Vì Tensorflow mô tả lại dòng chảy của dữ liệu thông qua graph nên mỗi một điểm giao cắt trong graph thì được gọi là Node. Tại sao điều này quan trọng thì là vì các Node chính là điểm đại diện cho việc thay đổi của dữ liệu nên việc lưu trữ lại tham chiếu của các Node này là rất quan trọng.

1.2.2.2 Tensor

Như trong bài viết trước mình có đề cập, để giải được các bài toán Machine Learning, cần phải làm cho máy tính có thể hiểu được dữ liệu của tập nguồn và dữ liệu của tập đích. Tensorflow cung cấp một loại dữ liệu mới được gọi là Tensor[[Tenensorflow]](https://www.tensorflow.org/api_docs/python/tf/Tensor). Trong thế giới của Tensorflow, mọi kiểu dữ liệu đều được quy về một mối được gọi là Tensor hay trong Tensorflow, tất cả các loại dữ liệu đều là Tensor. Vậy nên có thể hiểu được phần nào cái tên Tensorflow là một thư viện mô tả, điều chỉnh dòng chảy của các Tensor.

Tensor là một kiểu dữ liệu dạng mảng có nhiều chiều.

Mảng nhiều chiều này được đính kèm thêm một vài thuộc tính tham chiếu khác. Các thuộc tính của Tensor được mô tả trong tài liệu bao gồm:

**device**: Tên của thiết bị mà Tensor hiện tại sẽ được xuất bản. Có thể None.

**graph**: Đồ thị chứa Tensor hiện tại.

**name**: Tên của Tensor hiện tại.

**shape**: Trả về TensorShape mô tả lại Shape của Tensor hiện tại.

**op**: Phép toán được sử dụng để xuất bản Tensor hiện tại.

**dtype**: Kiểu của các phần tử trong Tensor hiện tại.

1.2.2.3 Rank

Rank là bậc hay độ sâu của một Tensor. Cách nhanh nhất để xác định rank của một Tensor là đếm số lần mở ngoặc vuông cho đến giá trị khác ngoặc vuông đầu tiên. Việc phân rank này khá quan trọng vì nó đồng thời cũng giúp phân loại dữ liệu của Tensor.

Khi ở cách rank đặc biệt cụ thể, Tensor có những tên gọi riêng như sau:

**Scalar**: Khi Tensor có rank bằng 0, Tensor đại diện cho một số hoặc một chuỗi cụ thể.

**Vector**: Vector là một Tensor rank 1. Trong python thì Vector là một list hay mảng một chiềuchứa các số.

**Matrix**: Đây là một Tensor rank 2 hay mảng hai chiều theo khái niệm của Python.

**N-Tensor**: Khi rank của Tensor tăng lên lớn hơn 2, chúng được gọi chung là N-Tensor.

1.2.2.3 Shape

Shape là một tuple có số chiều bằng với rank của Tensor tương ứng dùng để mô tả lại cấu trúc của Tensor đó. Dưới đây là ví dụ về Shape.

Tensor = 1 sẽ có Shape = ().

Tensor = [1] sẽ có Shape = (1).

Tensor = [[[1,1,1],[178,62,74]]] sẽ có Shape = (1,1,2).

Tensor = [[1,1,1],[178,62,74]] sẽ có Shape = (1,2).

Dựa vào cấu trúc của Shape, ta dễ dàng thấy rằng ràng buộc cơ bản của Tensor là chiều của các elements trong Tensor tại mỗi bậc phải bằng nhau.

1.2.2.4 Op

Được viết tắt là op, khái niệm Operator là toán tử được dùng để thực thi Tensor tại node đó. Các toán tử này có thể là const, variable, Add, Mul... Đôi khi mình cảm thấy việc dịch là toán tử cũng không hợp lí bởi lẽ các toán tử này đôi khi lại mô tả Node là Constant hay Variable. Có thể nói, khái niệm operator trong Tensorflow là khái niệm dùng để mô tả lại trạng thái của Node nói chung.

1.2.2.5 DType

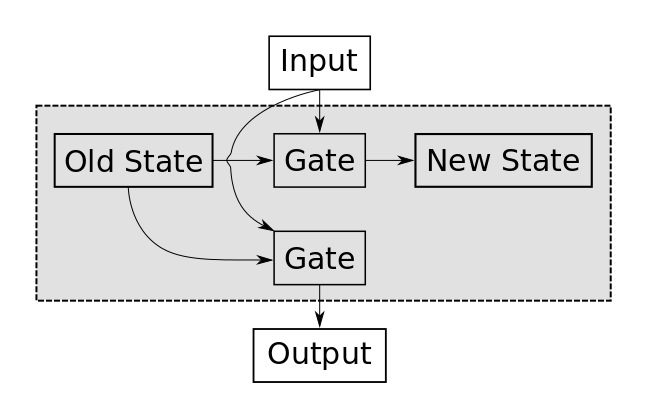
Đây là kiểu dữ liệu của các elements trong Tensor. Vì một Tensor chỉ có duy nhất một thuộc tính DType nên từ đó cũng suy ra là chỉ có duy nhất một kiểu DType duy nhất cho toàn bộ các elements có trong Tensor hiện tại.

1.3 NEURAL MACHINE TRANSLATION

1.3.1 Mạng hồi quy RNN(Recurrent Neural Network)

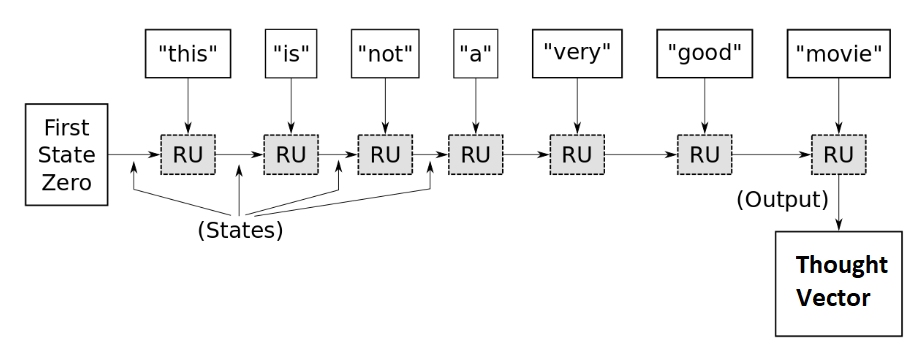
Thứ cơ bản trong RNN là Recurrent Unit (RU). Có rất nhiều các phiên bản khác nhau của RU từ phức tạp như LSTM (Long-Short-Term-Memory) và đơn giản hơn như GRU (Gated Recurrent Unit). Cả 2 mạng LSTM và GRU đều được hổ trợi trong TensorFlow.

Hình dưới đây sẽ cho chúng ta thấy ý tưởng của một RU, nó có một trạng thái ban đầu và sẽ được cập nhật mỗi khi nhận được input mới. Trạng thái ban đầu này đóng vai trò như một bộ nhớ. Nhưng mà nó không lưu trữ bits giống như bộ nhớ của máy tính. Thay vào đó RU chứa các dấu phẩy động trong bộ nhớ của mình. Như vậy mạng có thể được huấn luyện bình thường dùng Gradient Descent.



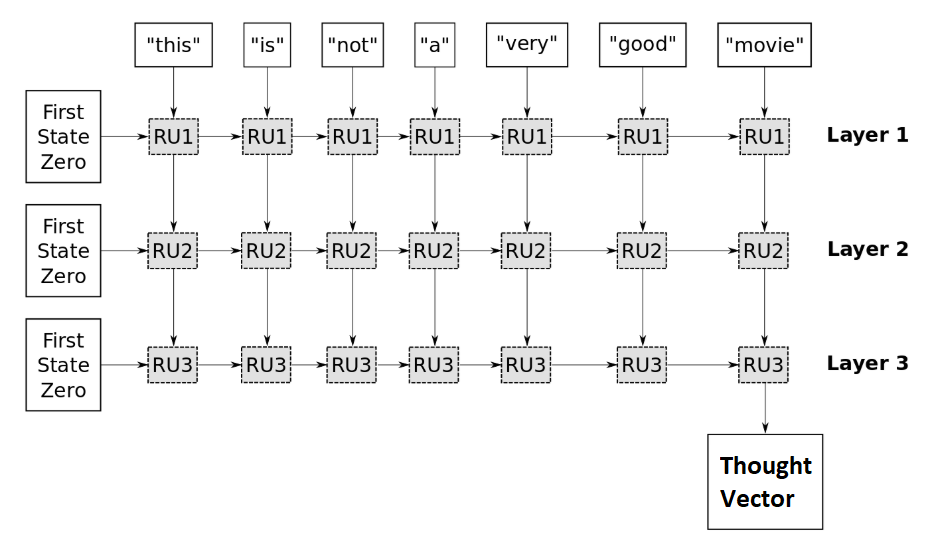
Dữ liệu của trạng thái mới sẽ phụ thuộc vào trạng thái cũ và input hiện tại. RU thông qua một cổng để nối dữ liệu của trạng thái cũ và input với trạng thái mới. Trong RU sẽ còn một cổng nửa để để tính toán output.

Tronh hình dưới đây sẽ là ví dụ về một RU. RU này sẽ nhận được một chuỗi input và xủ lý chuỗi này từng từ một để cho ra một vector thể hiện ý nghĩa cua chuỗi input.



Mỗi lần lặp RU sẽ nhận vào một từ của chuỗi input rồi dùng cổng để tính toán cập nhật lại bộ nhớ cứ lặp đi lặp lại như vậy cho đến hết chuỗi.Ở lần đầu tiên từ “this” sẽ được đưa vào RU, nó sẽ sử dụng trạng thái cũ (lúc này chưa có gì) và dùng cổng để tính toán ra trạng thái mới. RU lúc này chưa dùng đến cổng thứ hai vì nó chỉ cần lúc kết thúc chuổi để tổng hợp. Ở lần thứ hai từ “this” được đưa vào. Lúc này RU dùng bộ nhớ vừa được cập nhật lạ khi từ “this” được đưa vào ở bước trên và dùng cổng để tính toán sau đó cập nhật lại bộ nhớ.Lần lặp thứ 3 dùng bộ nhớ đã được cập nhật tại bước 2 và cổng để tính toán và cập nhật lại bộ nhớ. Cứ như vậy cho đên khi đến cuối chuỗi cổng thứ hai sẽ dùng dữ liệu bộ nhớ từ các bước trước và từ cuối cùng của chuỗi để cho ra một vector thể hiện ý nghĩa của chuỗi input.

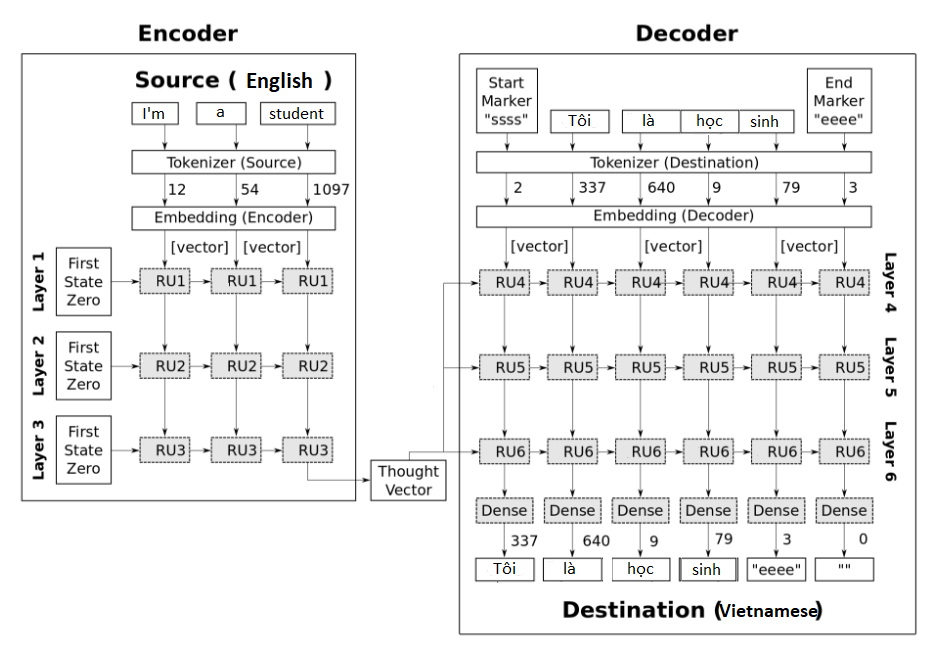
Chúng ta cũng có thể sử dụng cùng lúc nhiểu RU để tạo thành một RNN nhiều lớp. Lớp đầu tiên (RU1) sẽ gần giống như những gì ta đã nói ở trên. Nhưng thay vì đợi đến hết chuỗi để cho ra vector output thì ở đây vector output sẽ được xuất ra ở mỗi lần lặp và vector đó sẽ được dùng làm input cho tầng lớp theo (RU2). Tương tự như vậy đối với lớp thứ 3 (RU3).



1.3.2 Áp dụng RNN để dịch

1.3.2.1 Huấn luyện RNN

Biểu đồ dưới đây sẽ cho chúng ta thấy một RNN 3 lớp được cấu trúc như thế nào. Nó sẽ có 2 phần: Đầu tiên phần mã hóa sẽ mã hóa văn bản được đưa vào thành một vector mô tả ý nghĩ của các văn bản đó, rồi sau đó vector này sẽ được giải mã ở phần tiếp theo để cho ra văn bản theo ngôn ngữ mà ta mong muốn.



RNN không thể làm việc trực tiếp trên văn bản nên đầu tiên chúng ta cần phải chuyển các từ vựng thành số nguyên tương ứng theo tần số xuất hiện của chúng trong tập huấn luyện bằng cách sử dụng lớp tokenizer có sẵn trong TensorFlow . Nhưng mà neural network chỉ có thể làm việc với vector vì thế nên ta cần một tầng để chuyển các số nguyên thành một embedding-vector gồm các giá trị dấu phẩy động gọi là Embedding Layer.

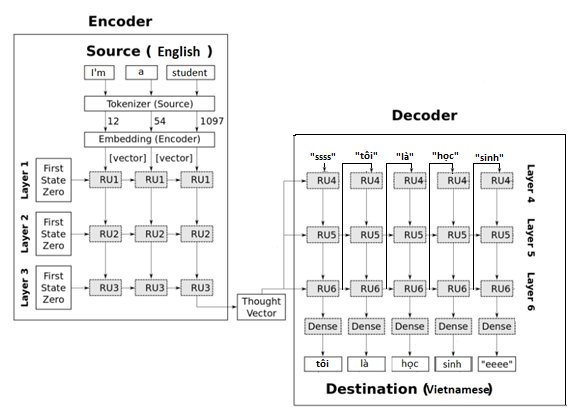
Ví dụ, “I’m a student” có nghĩa tiếng viết là “Tôi là học sinh”. Đầu tiên chúng ta đã chuyển toàn bộ tập huấn luyện sang số nguyên nên câu “I’m a student” sẽ có dạng [54, 12, 1097]. Mỗi số nguyên sẽ được Embedding Layer chuyển thành embedding-vectors. Như 12 sẽ trở thành [0.12, -0.56, ..., 1.19] và 54 sẽ thành [0.39, 0.09, ..., -0.12].Những embedding-vector này sẽ được dùng để làm input cho Recurrent Neural Network. RU cuối cùng của phần mã hóa cho ra một vector mô tả ý nghĩ của câu “I’m a student” tiếp theo vector này sẽ thành input cho các RU của phần giải mã.

Văn bản đích “Tôi là học sinh” sẽ được thêm các ký tự “ssss” và “eeee” để đánh dấu cho sự bắt đầu và kết thúc câu, vậy nên câu sẽ trở thành dãy số [2, 337, 640, 9, 79, 3]. Trong quá trình huấn luyện, phần giải mã sẽ được cho trước dãy số này làm input và output mong muốn sẽ là dãy số [337, 640, 9, 79, 3]. Chúng ta đang cố tình huấn luyện cho phần giải mã nối vector output của phẫn mã hóa và ký tự bắt đầu “ssss” (số 2) với từ tiếp theo là “Tôi” (Số 337), và rồi nối “Tôi” với từ “là” (số 640), tương tự như bậy với các từ phía sau.

Trong quá trình huấn luyện mô hình của chúng ta sẽ được input vào từng chuỗi trong tập huấn luyện. Mỗi chuỗi sẽ được xử lý như ví dụ ở trên. Kết thúc mỗi chuỗi sẽ có một giá trị loss được tính theo thuật toán “cross entropy” để so sánh sự khác nhau của ouput mà ta mong muốn với output mà mô hình đưa ra. Để làm cho giá trị loss này nhỏ đi ta sử dụng thuật toán “Adam Optimizer”

1.3.2.2 Tạo bản dịch

Khi RNN của chúng ta đã được huấn luyện xong ta có thể bắt đầu dùng nó để dịch. Ý tưởng của quá trình dịch được mô tả như hình bên dưới:



Câu “I’m a student” vẫn sẽ được mã hóa như lúc huấn luyện.Tiến trình dịch sẽ bắt ngay khi phần giải mã nhận được ký tự bắt đầu “ssss”. Nhưng thay vì luôn luôn cho các từ dịch chính xác làm input, phần giả mã lúc này dùng các từ được dự đoán bởi mô hình để làn input cho các bước tiếp theo. Quá trình sẽ tiếp tục như vậy cho đến khi phần giải mã nhận được ký tự kết thúc “eeee”.

CHƯƠNG II: XÂY DỰNG ỨNG DỤNG DỊCH MÁY VỚI TENSORFLOW

2.1 Import thư viện:

Đầu tiên chúng ta cần phải import một số thưu viện:

**import** **matplotlib.pyplot** **as** **plt**

**import** **tensorflow** **as** **tf**

**import** **numpy** **as** **np**

**import** **math**

**import** **os**

**from** **tensorflow.python.keras.models** **import** Model

**from** **tensorflow.python.keras.layers** **import** Input, Dense, GRU, Embedding

**from** **tensorflow.python.keras.optimizers** **import** RMSprop

**from** **tensorflow.python.keras.callbacks** **import** EarlyStopping, ModelCheckpoint, TensorBoard

**from** **tensorflow.python.keras.preprocessing.sequence** **import** pad\_sequences

2.2 Tải dữ liệu:

Chúng sẽ sử dụng tập dữ liệu IWSLT'15 English-Vietnamese data với hơn 133000 câu và 10000 từ vựng thông dụng nhất để dùng cho ứng dụng

Để phần giải mã biết được khi nào bắt đầu và kết thúc của một câu, ta cần phải đánh dấu chúng bằng những ký tự đặt biệt không có trong tập dữ liệu.

mark\_start = 'ssss '

mark\_end = ' eeee'

Tiếp theo ta tải dữ liệu cho phần văn bản gốc. Ở đây là tiếng anh:

data\_src = europarl.load\_data(english=**False**,

language\_code=language\_code)

Tải dữ liệu cho phần văn bản đích (tiếng Việt):

data\_dest = europarl.load\_data(english=**True**,

language\_code=language\_code,

start=mark\_start,

end=mark\_end)

Chúng ta sẽ tạo ra một mô hình dịch từ tiếng Anh sang tiếng Việt.

2.3 Tokenizer

Chúng ta sẽ sử dụng lớp Tokenizer có sẵn của Keras.Các chuỗi trong quá trình huấn luyện cần có độ dài như nhau nên chúng ta thêm các số 0 vào với những mẫu ví dụ ngắn và cắt ngẵn đối với các mẫu quá dài. Chúng ta cần phải dùng Tokenizer với cả hai tập dữ liệu gốc và đích vì tập từ vựng của chúng khác nhau

Bây giờ ta tạo tokenizer  cho tập dữ liệu gốc. Trong phần này ta sẽ thêm các số 0 ngay từ đầu. Các chuỗi token cũng sẽ được đảo ngược để cho ra kết quả tốt hơn.

tokenizer\_src = TokenizerWrap(texts=data\_src,

padding='pre',

reverse=**True**,

num\_words=num\_words)

Tiếp theo là tokenizer cho tập dữ liệu đích. Trong phần này chuỗi token sẽ không bị đảo ngược và các số không sẽ được thêm vào phía sau.

tokenizer\_dest = TokenizerWrap(texts=data\_dest,

padding='post',

reverse=**False**,

num\_words=num\_words)

2.4 Tạo RNN

2.4.1 Tạo phần mã hóa

Thành phần đầu tiên trong một RNN chính là phần mã hóa. Chúng ta sẽ dùng các API có sẵn của Keras cho việc này. Đầu tiên ta sẽ tạo cho mỗi tầng một đối tượng riêng biệt và rồi kết chúng lại với nhau sau.

Sau đây sẽ là input cho phần mã hóa. Nhận vào các mẫu ví dụ đã được tokenizer. None thể hiện mỗi mẫu ví dụ có thể có độ dài tùy biến.

encoder\_input = Input(shape=(**None**, ), name='encoder\_input')

Tiếp theo là embedding-layer

embedding\_size = 128

encoder\_embedding = Embedding(input\_dim=num\_words,

output\_dim=embedding\_size,

name='encoder\_embedding')

Tiếp theo ta sẽ tạo ra 3 tầng GRU (Gated Recurrent Units). Các tầng này sẽ ánh xạ từ embedding-vectors thành một vector duy nhất mô tả ý nghĩa của mẫu ví dụ input.

state\_size = 512

encoder\_gru1 = GRU(state\_size, name='encoder\_gru1',

return\_sequences=**True**)

encoder\_gru2 = GRU(state\_size, name='encoder\_gru2',

return\_sequences=**True**)

encoder\_gru3 = GRU(state\_size, name='encoder\_gru3',

return\_sequences=**False**)

Ta sẽ sử dụng một hàm để kết nối các tầng GRU và embedding lại với nhau.

**def** connect\_encoder():

*# Bắt đầu network với tầng nhận input.*

net = encoder\_input

*# Kết nối embedding-layer.*

net = encoder\_embedding(net)

*# Kết nối tất cả các tầng GRU-layers*

net = encoder\_gru1(net)

net = encoder\_gru2(net)

net = encoder\_gru3(net)

*# Đây sẽ là output của phần mã hóa.*

encoder\_output = net

**return** encoder\_output

Và rồi giờ ta kết nối tất cả lại với nhau.

encoder\_output = connect\_encoder()

2.4.2 Tạo phần giải mã

Phần giải mã sẽ có 2 input. Thứ nhất là vector kết quả của phần mã hóa

decoder\_initial\_state = Input(shape=(state\_size,),

name='decoder\_initial\_state')

Phần giải mã còn cần một chuỗi integer-tokens làm input. Trong quá trình huấn luyện chúng ta sẽ cung cấp đầy đủ chuỗi này. Nhưng khi tiến hành dịch một câu mới hoàn toàn, chúng ta sẽ chỉ cung cấp một token là “ssss” thể hiện sự bắt đầu cho một câu và cùng với vector có được từ phần mã hóa, phần giải mã sẽ tự phán đoán từ tiếp theo.

decoder\_input = Input(shape=(**None**, ),name='decoder\_input’)

Tiếp chúng ta sẽ tạo embedding-layer cho phần giải mã.

decoder\_embedding = Embedding(input\_dim=num\_words,

output\_dim=embedding\_size,

name='decoder\_embedding')

Tiếp theo ta tạo 3 tầng GRU cho phần giải mã.

decoder\_gru1 = GRU(state\_size, name='decoder\_gru1',

return\_sequences=**True**)

decoder\_gru2 = GRU(state\_size, name='decoder\_gru2',

return\_sequences=**True**)

decoder\_gru3 = GRU(state\_size, name='decoder\_gru3',

return\_sequences=**True**)

Các tầng GRU output ra một tensor[batch\_size, sequence\_length, state\_size], nơi mà mỗi từ sẽ được mã hóa thành một vector với độ dài là state\_size. Ta cần phải chuyển đổi tensor này thành chuỗi integer-tokens để rồi ánh xạ thành các từ trong tập từ vựng.

decoder\_dense = Dense(num\_words,

activation='linear',

name='decoder\_output')

Tiếp theo tương tự với phần ta sẽ tạo ra một hàm để các nối các tầng GRU và embedding.

**def** connect\_decoder(initial\_state):

*# Bắt đầu phần giải mã với tầng input.*

net = decoder\_input

*# Kết nối với embedding-layer.*

net = decoder\_embedding(net)

*# Kết nối các tầng GRU.*

net = decoder\_gru1(net, initial\_state=initial\_state)

net = decoder\_gru2(net, initial\_state=initial\_state)

net = decoder\_gru3(net, initial\_state=initial\_state)

*# Kết nối với tầng dense chuyển đổi GRU ouput*

*# thành one-hot encoded arrays.*

decoder\_output = decoder\_dense(net)

**return** decoder\_output

2.4.3 Kết nối và tạo mô hình

Đầu tiên ta kết nối trực tiếp phần mã hóa và giải mã. Như vậy trạng thái ban đầu của GRU bên phần giả mã sẽ là output của phần mã hóa.

decoder\_output = connect\_decoder(initial\_state=encoder\_output)

model\_train = Model(inputs=[encoder\_input, decoder\_input],

outputs=[decoder\_output])

sa

Sau đó ta tạo ra một mô hình riêng cho phần mã hóa và riêng cho phần giải mã. Giúp dễ dàng hơn khi tạo ra bản dịch.

model\_encoder = Model(inputs=[encoder\_input],

outputs=[encoder\_output])

decoder\_output = connect\_decoder(initial\_state=decoder\_initial\_state)

model\_decoder = Model(inputs=[decoder\_input, decoder\_initial\_state],

outputs=[decoder\_output])

2.4.4 Huấn luyện mô hình

Mô hình huấn luyện sử dụng thuật toán sparse cross entropy để tính giá trị loss, thuật toán tối ưu RMSprop.

model\_train.compile(optimizer=optimizer,

loss=sparse\_cross\_entropy,

target\_tensors=[decoder\_target])

Ta sẽ huấn luyện mô hình với 10 lần lặp lại toàn bộ tập huấn luyện. Mỗi lần lặp tốn 2 giờ với GPU Nvidia GeForce 940MX.

model\_train.fit(x=x\_data,

y=y\_data,

batch\_size=640,

epochs=10,

validation\_split=validation\_split,

callbacks=callbacks)

2.4.5 Tiến hành dịch

Đầu tiên ta chuyển chuỗi input thành token.

Input\_tokens = tokenizer\_src.text\_to\_tokens(text=input\_text,

reverse=**True**)

Tiếp theo ta lấy kết quả từ mô hình mã hóa làm trạng thái bắt đầu cho mô hình giải mã.

initial\_state = model\_encoder.predict(input\_tokens)

Đây là vòng lặp để lấy ra kết quả dịch. Khi chưa gặp ký hiệu kết thúc “eeee” và số lượng token nhỏ hơn số lượng token tối đa.

**while** token\_int != token\_end **and** count\_tokens < max\_tokens:

*# Cập nhật token tiếp theo cho mô hình để giải mã*

*# Trong vòng lặp đầu tiên token sẽ là “ssss”*

decoder\_input\_data[0, count\_tokens] = token\_int

x\_data = \

{

'decoder\_initial\_state': initial\_state,

'decoder\_input': decoder\_input\_data

}

*# Lấy kết quả dự đoán từ mô hình giải mã cho token input.*

decoder\_output = model\_decoder.predict(x\_data)

*# Lấy token được dự đoán sau cùng dưới dạng one-hot array.*

token\_onehot = decoder\_output[0, count\_tokens, :]

*# Chuyển đổi thành integer-token.*

token\_int = np.argmax(token\_onehot)

*# Ánh xạ integer-token thành từ.*

sampled\_word = tokenizer\_dest.token\_to\_word(token\_int)

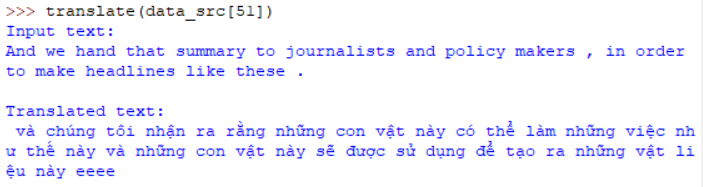
*# Thêm từ vừa được ánh xạ vào output-text.*

output\_text += " " + sampled\_word

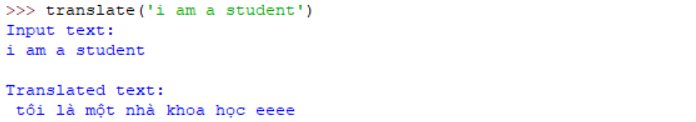
count\_tokens += 1

2.4.6 Thực nghiệm dịch

Dịch một mẫu ví dụ trong tập huấn luyện.

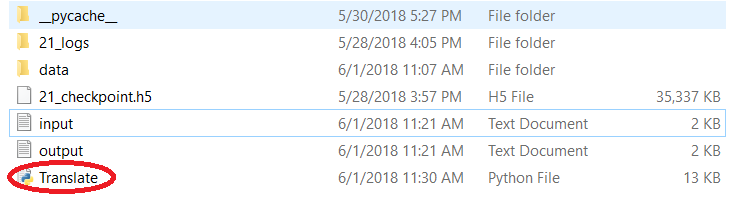


Dịch một câu không có trong tập huấn luyện.

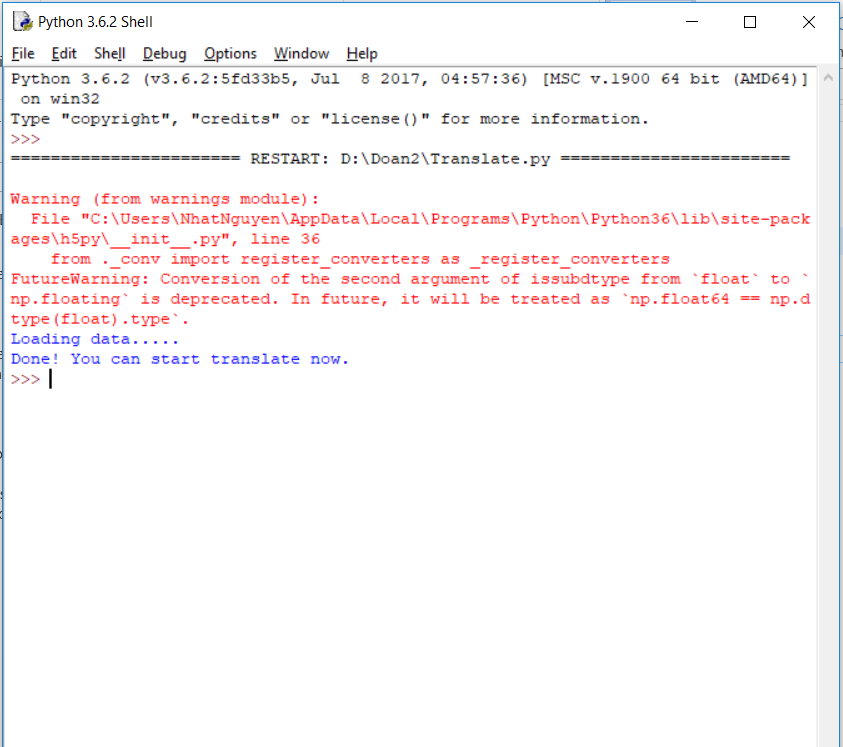


2.4.7 Hướng dẫn sử dụng

Chạy file Translate.py:

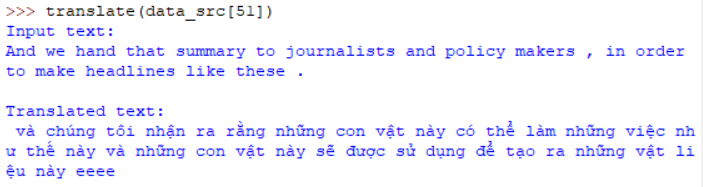


Chờ một lúc để chương trình load dữ liệu:

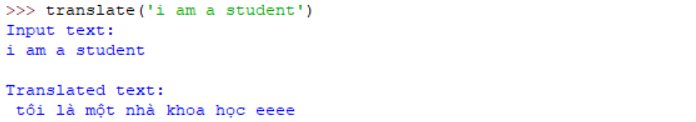


Dịch một câu nhập từ bàn phím:

- Dịch một mẫu ví dụ trong tập huấn luyện.

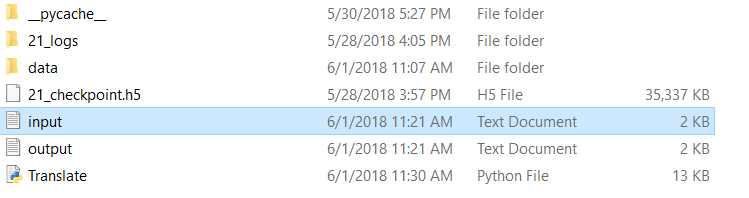


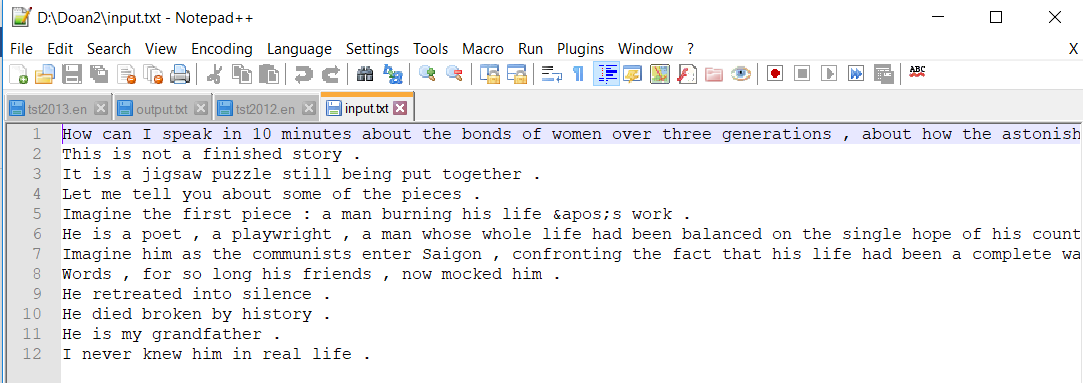
- Dịch một câu không có trong tập huấn luyện.



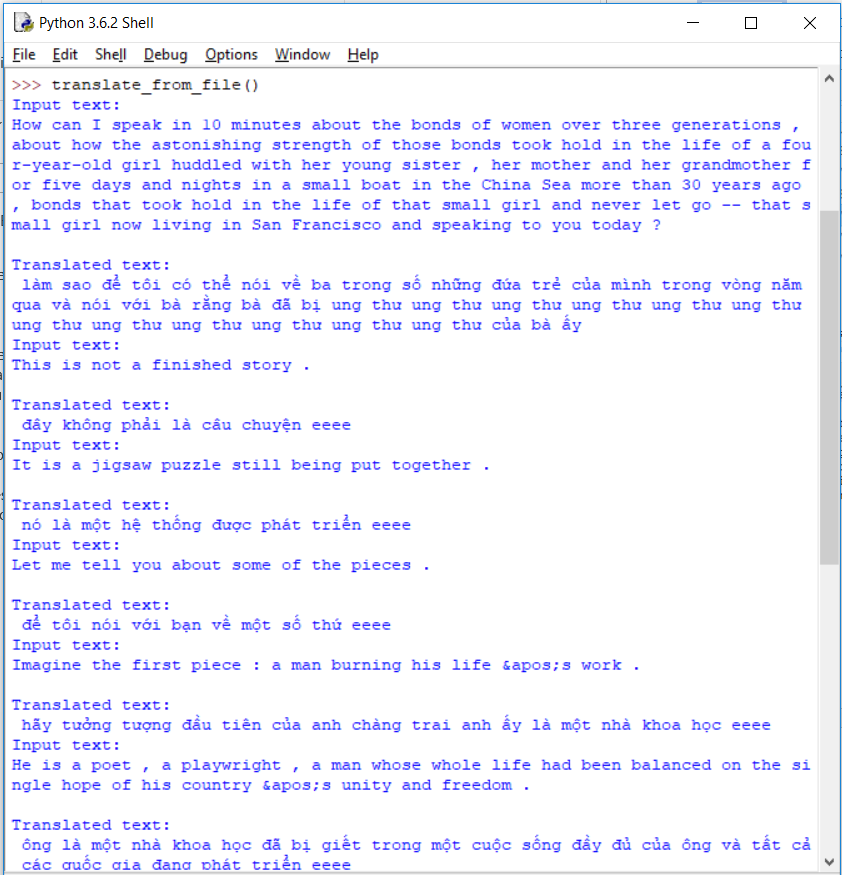
Dịch từ file

- Nhập dữ liệu vào file input.txt.

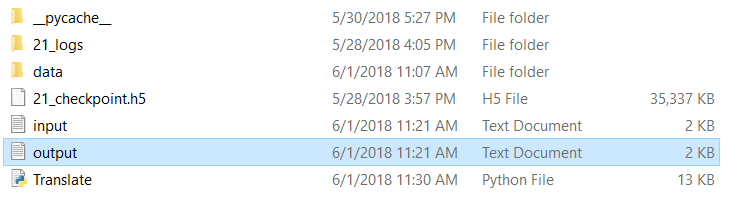


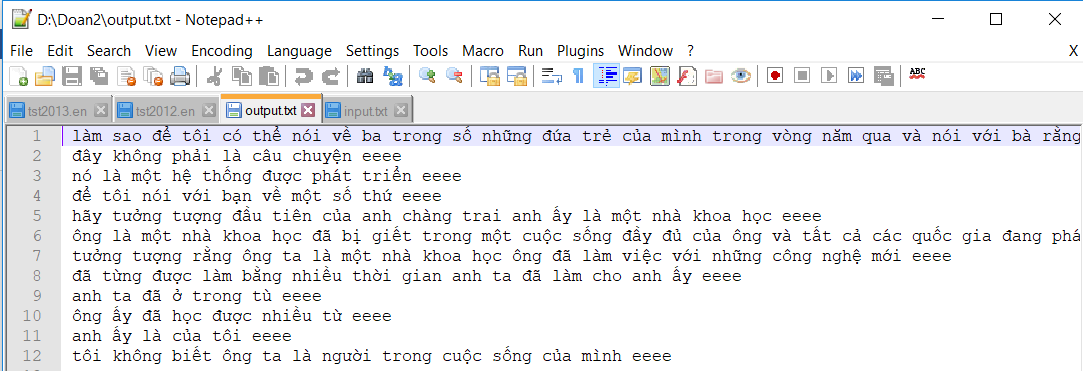


- Gọi hàm translate\_from\_file().



- Dữ liệu dịch sẽ được ghi ra file output.txt.





TÀI LIỆU THAM KHẢO

<https://nlp.stanford.edu/projects/nmt/>

<https://github.com/tensorflow/nmt/tree/master/nmt>

<https://www.tensorflow.org/tutorials/seq2seq>

<https://github.com/Hvass-Labs/TensorFlow-Tutorials>