TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**HỌC MÁY**

**SHRINKAGE METHOD**

*Người hướng dẫn*: **Thầy Lê Anh Cường**

*Người thực hiện*: **Nguyễn Duy Hàn Lâm (MSHV: 196005004)**

Khoá  **: 2019**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2020**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**HỌC MÁY**

**SHRINKAGE METHOD**

*Người hướng dẫn*: **Thầy Lê Anh Cường**

*Người thực hiện*: **Nguyễn Duy Hàn Lâm (MSHV: 196005004)**

Khoá  **: 2019**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2020**

**ĐỒ ÁN ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi và được sự hướng dẫn của thầy Lê Anh Cường. 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 30 tháng 04 năm 2020*

*Tác giả*

*(ký tên và ghi rõ họ tên)*

*Nguyễn Duy Hàn Lâm*

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

[PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN ii](#_Toc38785913)

[MỤC LỤC 1](#_Toc38785914)

[CHƯƠNG 1: DỊCH MÁY NEURAL 4](#_Toc38785915)

[1.1 Giới thiệu về dịch máy neural 4](#_Toc38785916)

[1.2 Recurrent Neural Network trong mô hình 5](#_Toc38785917)

[1.2.1 Cổng update 7](#_Toc38785918)

[1.2.2 Cổng reset 7](#_Toc38785919)

[1.2.3 Xuất kết quả 8](#_Toc38785920)

[1.3 Convolutional Neural Network trong mô hình 9](#_Toc38785921)

[1.4 Mô hình Encoder-Decoder 11](#_Toc38785922)

[1.4.1 Word embedding 12](#_Toc38785923)

[1.4.2 Encoder 13](#_Toc38785924)

[1.4.3 Decoder 13](#_Toc38785925)

[1.4.4 Linguistic Input Feature 16](#_Toc38785926)

[1.5 Tóm tắt chương 1 18](#_Toc38785927)

[CHƯƠNG 2: THỰC HIỆN MÔ PHỎNG ENCODER-DECODER 19](#_Toc38785928)

[2.1 Chuẩn bị nguồn dữ liệu và thư viện 19](#_Toc38785929)

[2.2 Xử lý dữ liệu và tạo các đối tượng quản lý 19](#_Toc38785930)

[2.3 Xây dựng encoder-decoder 23](#_Toc38785931)

[2.4 Huấn luyện 27](#_Toc38785932)

[2.5 Đánh giá 33](#_Toc38785933)

[2.6 Kiểm thử 35](#_Toc38785934)

[2.7 Chạy chương trình và kết quả 36](#_Toc38785935)

[2.8 Tóm tắt 37](#_Toc38785936)

[TÀI LIỆU THAM KHẢO 39](#_Toc38785937)

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

**DANH MỤC HÌNH**

[Hình 1.1. Sơ lược cấu tạo hoạt động của RNN [1] 5](#_Toc38785871)

[Hình 1.2. Cấu trúc neural theo GRU [2] 6](#_Toc38785872)

[Hình 1.3. Mô tả cách tính của cổng update [2] 7](#_Toc38785873)

[Hình 1.4. Mô tả cách tính cổng reset [2] 8](#_Toc38785874)

[Hình 1.5. Mô tả cách tính [2] 8](#_Toc38785875)

[Hình 1.6. Cấu trúc của CNN trong encoder [3] 9](#_Toc38785876)

[Hình 1.7. Minh họa cách hoạt động của hàm activation ở mỗi node [3] 10](#_Toc38785877)

[Hình 1.8. Minh họa cho một số cấu trúc của CNN [3] 11](#_Toc38785878)

[Hình 1.9. Mô hình encoder-decoder cơ bản [3] 12](#_Toc38785879)

[Hình 1.10. Cấu trúc của một bộ GRU-RNN encoder [4] 13](#_Toc38785880)

[Hình 1.11. Minh họa cho decoder [4] 14](#_Toc38785881)

[Hình 1.12. Cơ chế attention [5] 15](#_Toc38785882)

[Hình 1.13. Cấu trúc chi tiết của một attention [6] 15](#_Toc38785883)

[Hình 2.1. Các thư viện cần chuẩn bị 19](#_Toc38785884)

[Hình 2.2. Các thư viện phục vụ linguistic input feature 19](#_Toc38785885)

[Hình 2.3. Class quản lý dữ liệu 20](#_Toc38785886)

[Hình 2.4. Xử lý và chuẩn hóa dữ liệu 21](#_Toc38785887)

[Hình 2.5. Hàm đọc dữ liệu 22](#_Toc38785888)

[Hình 2.6. Chuẩn hóa số lượng từ của câu và lemmas hóa dữ liệu 22](#_Toc38785889)

[Hình 2.7. Hàm đọc dữ liệu và thực hiện nạp dữ liệu vào chương trình 23](#_Toc38785890)

[Hình 2.8. Xây dựng bộ encoder 23](#_Toc38785891)

[Hình 2.9. Xây dựng bộ decoder cơ bản 24](#_Toc38785892)

[Hình 2.10. Cấu trúc hoạt động của attention 25](#_Toc38785893)

[Hình 2.11. Class decoder sử dụng attention (hàm khởi tạo của class) 26](#_Toc38785894)

[Hình 2.12. Hàm khởi tạo tiến trình và hàm feedforward 27](#_Toc38785895)

[Hình 2.13. Các hàm tạo vector input và output cho quá trình huấn luyện 28](#_Toc38785896)

[Hình 2.14. Hàm thực hiện huấn luyện 29](#_Toc38785897)

[Hình 2.15. Giai đoạn tiền encoder 29](#_Toc38785898)

[Hình 2.16. Giai đoạn chạy encoder 30](#_Toc38785899)

[Hình 2.17. Giai đoạn hậu encoder và tiền decoder 30](#_Toc38785900)

[Hình 2.18. Giai đoạn chạy decoder 30](#_Toc38785901)

[Hình 2.19. Giai đoạn hậu decoder 31](#_Toc38785902)

[Hình 2.20. Hàm thông báo lượng thời gian chương trình đã và đang chạy 31](#_Toc38785903)

[Hình 2.21. Hàm vẽ biểu đồ thay đổi của giá trị loss 32](#_Toc38785904)

[Hình 2.22. Hàm huấn luyện tổng quát 33](#_Toc38785905)

[Hình 2.23. Hàm thực hiện đánh giá 34](#_Toc38785906)

[Hình 2.24. Hàm thực hiện chọn đánh giá ngẫu nhiên 35](#_Toc38785907)

[Hình 2.25. Các hàm phục vụ trình bày kết quả 35](#_Toc38785908)

[Hình 2.26. Tiến hành huấn luyện mô hình 36](#_Toc38785909)

[Hình 2.27. Kết quả của từng đợt huấn luyện 36](#_Toc38785910)

[Hình 2.28. Chạy thử 1 câu ví dụ 37](#_Toc38785911)

[Hình 2.29. Kết quả chạy thử 1 câu ví dụ 37](#_Toc38785912)

TÓM TẮT NỘI DUNG

Học máy là một lĩnh vực quan trọng trong thời đại công nghệ hiện nay. Trong đó dữ liệu là 1 phần cốt lõi của lĩnh vực vì chương trình sẽ phải dựa trên nguồn tập dữ liệu này mới có thể đào tạo, học và từ đó mới sinh ra được mô hình để có thể từ đó giải quyết bài toán.

Dữ liệu càng lớn, các đối tượng trong dữ liệu càng chứa nhiều thuộc tính (hay còn gọi là chiều dữ liệu) sẽ càng khiến chương trình trở nên nặng nề, khó đào tạo và từ đó sẽ khó có thể đào tạo được ra một mô hình chuẩn xác để giải quyết bài toán. Bằng các kĩ thuật tính toán ta có thể hồi quy đưa số chiều của dữ liệu lớn trở nên nhỏ hơn nhằm giảm bớt gánh nặng trong việc đào tạo ra mô hình. Kĩ thuật tính toán này được gọi là shrinkage method (các phương thức co rút chiều dữ liệu).

Tài liệu này bao gồm các phần chính:

* Giới thiệu lại về mô hình tuyến tính.
* Các phương pháp hồi quy thô.
* Các phương pháp hồi quy chính (shrinkage method).
* Áp dụng vào chương trình mẫu.

CHƯƠNG 1: MÔ HÌNH TUYẾN TÍNH

* 1. Mô hình tuyến tính

Mô hình tuyến tính là mô hình có hàm số biểu diễn cho phương trình đường thẳng:

Trong đó:

* là giá trị mà mô hình tính toán được.
* là giá trị chặn (intercept) hay còn gọi là bias (phần bù đại số).
* là tập vector dữ liệu đầu vào.

Ta có thể thế và rút gọn biểu thức (1) về dạng:

Từ đó ta có thể thấy là tích vô hướng, và kết quả của nó nó tập chứa K các vector, sẽ là một ma trận hệ số có kích thước là . Trong không gian (p + 1) chiều ứng với tập dữ liệu đầu vào và tập dữ liệu đầu ra thì tập sẽ là 1 siêu phẳng. Nếu trong X có chứa hằng số thì siêu phẳng này sẽ chứa cả không gian gốc lẫn các không gian con liên quan. Nếu trong X không có hằng số thì siêu phẳng sẽ là một tập hợp các phép biến đổi cắt trục Y tại điểm .

Xét (1) trong mặt phẳng p chiều, ta có thể thấy nó được tổng quát hóa bằng hàm số , từ đó ta được đạo hàm của nó là là một tập hợp các điểm hình thành nên một nên một đường thẳng có dốc hướng lên.

Việc hiệu chỉnh mô hình đó chính là ta sẽ thay đổi đường thẳng sao cho khoảng cách từ tất các điểm dữ liệu đến đường thẳng là nhỏ nhất hay nói cách khác là làm sao cho giá trị của phương trình sau càng nhỏ càng tốt:

là 1 hàm bậc hai chính vì vậy nó luôn tồn tại giá trị nhỏ nhất, tuy nhiên trong 1 vài trường hợp giá trị này không phải duy nhất. ta có thể biểu diễn RSS về dạng phép tính của ma trận thành:

Trong đó:

* X là ma trận dữ liệu đầu vào có kích thước là . Mỗi dòng là 1 vector dữ liệu đầu vào.
* là tập chứa N vector kết quả mà mô hình tính toán được từ tập dữ liệu huấn luyện.

Ta đạo hàm (3) theo , ta được phương trình:

Từ (4) ta giải phương trình, ta ra được giải pháp ta cần tìm:

Giá trị mà chương trình tính toán được từ input thứ i sau khi hiệu chỉnh sẽ có dạng:

* 1. Mô hình thống kê

Cho là tập dữ liệu đầu vào chứa các giá trị số thực ngẫu nhiên.

Cho là tập dữ liệu kết quả chứa các giá trị số thực ngẫu nhiên.

Đồng thời ta có:

Trong đó:

Ta cần đi tìm một hàm f có khả năng từ các dữ liệu X dự đoán thành các dữ liệu Y. Sai số là điều không tránh khỏi vì vậy ta cần một hàm tính sai số (expected prediction error – EPE):

Ta có thể viết thành:

Kết hợp biểu thức (6) và (8) ta được:

Ta thấy ta có thể giảm bớt giá trị EPE theo từng điểm của f:

Từ đó ta tìm được giải pháp:

Trong đó:

* E là giá trị kỳ vọng

Biểu thức (12) được gọi là biểu thức kỳ vọng có điều kiện, còn được gọi là hàm hồi quy. Các giá trị dự đoán tốt nhất của Y tại bất kỳ điểm X = x nào được gọi là các giá trị trung bình có điều kiện, và giá trị tốt nhất sẽ được tính bằng sai số bình phương trung bình.

Trong bài toán the neareast-neighbor, các phương thức luôn cố gắng tìm cách triển khai trực tiếp quá trình của nó trên dữ liệu huấn luyện. Với mỗi điểm x đầu vào, ta có thể sẽ phải đi tính giá trị trung bình của tất cả các giá trị yi dự đoán được ứng với từng xi của nó.

Trong đó:

* Ave là hàm tính giá trị trung bình.
* là tập chứa các điểm k trong tập dữ liệu gần với x nhất.

Với công thức (13), với , khi thì , dẫn đến .

Khi kích thước số chiều p lớn, sẽ dẫn đến việc kích thước của ma trận sẽ lớn. Mặc dù sự hội tụ vẫn diễn ra như quy luật trên, tuy nhiên tốc độ hội tụ của nó lại diễn ra chậm.

Vì vậy ta đi xây dựng một mô hình hồi quy tuyến tính để giải quyết vấn đề này. Giả sử mô hình hồi quy tuyến tính sẽ có dạng:

Ta kết hợp biểu thức (14) với biểu thức (5) và (8) ta được:

Tuy nhiên về bản chất thì 2 cái hoàn toàn khác nhau, trong biểu thức (5) cố gắng dự đoán giá trị dựa trên một hàm tuyến tính chung tổng thể, còn trong biểu thức (8) lại cố gắng dự đoán bằng các giá trị tính toán nội bộ. Vì thế việc làm này sẽ khiến quá trình tính toán sẽ không ổn định mặc dù ta giải quyết được bớt một phần của vấn đề liên quan đến kích thước của chiều dữ liệu.

Giả sử:

Trong đó:

* là một biến lỗi ngẫu nhiên có và độc lập với X.
* phụ thuộc vào X thông qua điều kiện trung bình của .

Việc thêm sẽ giúp mô hình có giá trị gần hơn với giá trị thật. Vì đối với hầu hết các hệ thống, các cặp (X, Y) sẽ không có mối quan hệ xác định . Nói chung, ngoài giá trị tính toán được từ mô hình thì các yếu tố không liên quan cũng sẽ đóng góp một phần vào giá trị Y. Và chắc chắn giá trị lỗi phát sinh từ mô hình cũng sẽ đóng góp vào Y. Các giá trị này này ta sẽ ước lượng và quy kết bằng giá trị .

* 1. Đánh đổi giữa Bias và Variance

Ta xét lại mô hình phân loại hồi quy k-nearest-neighbor.

Giá trị thật Y được cấu thành như sau:

Trong đó:

* là một biến lỗi ngẫu nhiên có và độc lập với X.

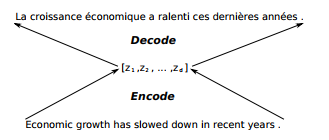
Ta có giá trị lỗi kỳ vọng (Expected Prediction Error – EPE) tại biến đầu vào như sau:

* 1. Mô hình Encoder-Decoder

Nhiệm vụ của dịch thuật có thể được hiểu theo quan điểm của học máy đó là việc học các phân phối có điều kiện của một câu đích (kết quả sau quá trình dịch) f dựa trên một câu nguồn e (câu muốn dịch). Bằng cách lấy mẫu (e,f) từ thực tế hoặc bằng cách sử dụng thuật toán tìm kiếm (gần đúng) mà ta có thể tìm được mức tối đa của phân phối.

Một số bài báo gần đây đã đề xuất sử dụng mạng lưới thần kinh để trực tiếp tìm hiểu phân phối có điều kiện từ một kho văn bản song ngữ (Kalchbrenner và Blunsom, 2013; Cho et al., 2014; Sutskever et al., 2014). Ví dụ, các tác giả (Kalchbrenner và Blunom, 2013) đã đề xuất một cách tiếp cận liên quan đến mô hình n-gram tích chập để trích xuất một vectơ có độ dài cố định của một câu nguồn được giải mã bằng mô hình n-gram ngược được tăng cường bằng RNN. Trong (Sutskever et al., 2014), một RNN với các đơn vị LSTM đã được sử dụng để mã hóa một câu nguồn và bắt đầu từ trạng thái ẩn cuối cùng, để giải mã một câu đích. Tương tự, các tác giả của (Cho et al., 2014) đã đề xuất sử dụng RNN để mã hóa và giải mã một cặp cụm từ.

Tất cả các mô hình này đều được xây dựng dựa trên cấu trúc gồm 2 bộ phận chính là bộ encoder (bộ mã hóa) và bộ decoder (bộ giải mã) (xem hình 1.9). Bộ mã hóa xử lý một câu dữ liệu đầu vào có độ dài thay đổi (câu nguồn) từ đó xây dựng được một vectơ biểu diễn có độ dài cố định (ký hiệu là trong Hình 1.9). Dựa trên kết quả vector này, bộ giải mã sẽ tạo ra được một chuỗi có độ dài thay đổi tương ứng (câu đích).



1. Mô hình encoder-decoder cơ bản [3]

Trước đây trong tài liệu của (Sutskever et al., 2014) phương pháp encoder-decoder này đã được sử dụng chủ yếu như một phần của hệ thống dịch máy thống kê (Statistical Machine Translation – SMT) hiện có. Cách tiếp cận này được sử dụng để xếp hạng lại danh sách n đối tượng tốt nhất được tạo bởi hệ thống SMT trong (Kalchbrenner và Blunom, 2013) và các tác giả của (Cho et al., 2014) đã sử dụng phương pháp này để cung cấp điểm số bổ sung cho phần điểm hiện có của bảng cụm từ.

* + 1. Word embedding

Word embedding trong thực tế là một lớp các kỹ thuật trong đó các từ riêng lẻ được biểu diễn dưới dạng các vectơ có giá trị thực trong một không gian vectơ được xác định trước. Mỗi từ được ánh xạ tới một vectơ và các giá trị vectơ được học theo cách giống với mạng thần kinh, và do đó kỹ thuật này thường được đưa vào lĩnh vực học sâu.

Chìa khóa của cách tiếp cận là ý tưởng sử dụng một đại diện phân phối mật độ cho mỗi từ.

Mỗi từ được đại diện bởi một vectơ có giá trị thực, thường là hàng chục hoặc hàng trăm chiều, khác với one-hot encoding mỗi từ sẽ là một vector chứa các giá trị 0 và 1.

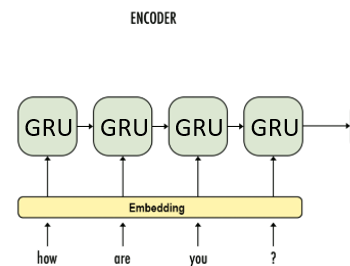
Các đại diện phân phối được học dựa trên việc xuất hiện của các từ. Các từ xuất hiện hay được sử dụng như thế nào thì sẽ có vector biểu diễn tương ứng, vô hình ta có thể tự nhiên nắm bắt ý nghĩa của chúng.

Có một lý thuyết ngôn ngữ sâu hơn đằng sau giúp một phần giải thích cách tiếp cận này. Lý thuyết này có tên là “giả thuyết phân phối” (“distributional hypothesis”) của Zellig S. Harris, có thể được tóm tắt như sau: những từ có ngữ cảnh tương tự sẽ có nghĩa tương tự. (Có thể tham khảo thêm trong tài liệu: Zellig S. Harris, 1954, Distributional Structure).

Trên thực tế, con người giao tiếp với nhau thông qua các kí tự, giọng nói, hình ảnh,… Tuy nhiên điều này vô cùng khó khăn cho máy tính khi xử lý với bất kỳ bài toán. Vì máy tính “quen” với việc xử lý trên các dữ liệu dạng số, vì vậy trước khi ta thực hiện quá trình dịch máy, ta sẽ phải word embedding dữ liệu để đưa dữ liệu về dạng các vector biểu diễn số.

* + 1. Encoder

Cấu trúc của một bộ GRU-RNN encoder có thể được minh họa như sau:



1. Cấu trúc của một bộ GRU-RNN encoder [4]

Đầu vào của neural đầu tiên chỉ có dữ liệu gốc, không có giá trị ẩn (hidden unit bằng 0).

Kết quả đầu ra của encoder sẽ là tập vector có độ dài cố định:

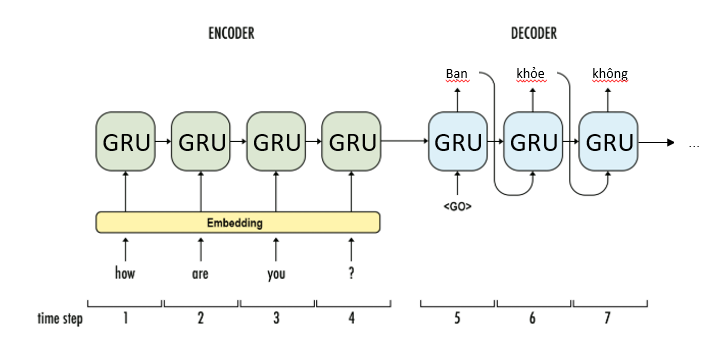
Cấu trúc của grConv encoder giống với phần mô tả ở mục 1.3.

* + 1. Decoder

*Decoder thuần:*

Một bộ decoder đơn thuần sẽ chỉ sử dụng tập kết quả đầu ra của encoder để làm đầu vào cho mình. Tập kết quả của encoder tạo được này đôi khi được gọi là vector bối cảnh vì nó mã hóa bối cảnh từ toàn bộ tập dữ liệu ban đầu. Vectơ bối cảnh này được sử dụng làm trạng thái ẩn (hidden unit) ban đầu cho decoder.

Ở mỗi bước giải mã (decode), input đầu vào, decoder sẽ được cung cấp bởi 2 giá trị: kết quả và trạng thái ẩn của neural phía trước (xem minh họa ở hình 1.11). Chuỗi decoder sẽ bắt đầu bằng chuỗi <GO> (hình 1.11) (do là neural đầu tiên trong decoder) và trạng thái ẩn đầu tiên chính là vectơ ngữ cảnh (kết quả cuối cùng của encoder).



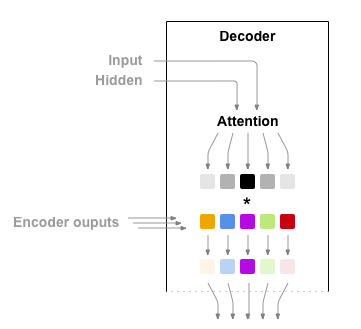
1. Minh họa cho decoder [4]

*Decoder sử dụng attention:*

Attention là một cơ chế kỹ thuật nhằm giúp giải quyết các giới hạn của kiến trúc encoder-decoder trên các chuỗi dài (chuỗi dữ liệu đầu vào). Attention sẽ giúp tăng tốc độ học tập và cải thiện khả năng dự đoán của mô hình.

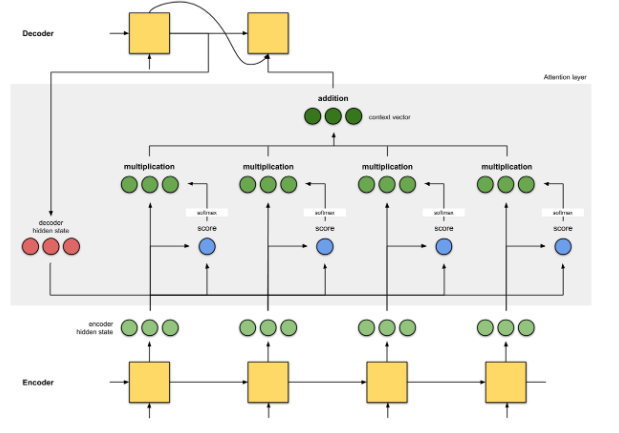
Do vector ngữ cảnh này có độ dài cố định và là kết quả sau quá trình mã hóa từ dữ liệu đầu vào có độ dài thay đổi, nên vector này sẽ chứa rất nhiều thông tin. Việc mã hóa (decode) vector này sẽ là một áp lực lớn đối với decoder.

Cơ chế attention cho phép tại mỗi bước input của decoder, decoder có thể tập trung vào từng phần khác nhau của encoder output. Đầu tiên, chúng ta sẽ tính tập trọng số attention (attention weight). Sau đó ta sẽ lấy tập trọng số này nhân với các vectơ đầu ra của encoder để tạo ra một tổ hợp có trọng số. Kết quả (được gọi là attn\_applied) này giúp nhận biết được từng phần của chuỗi đầu vào, và từ đó sẽ giúp decoder chọn được từ (word) đầu ra đúng.



1. Cơ chế attention [5]

Thay vì mã hóa chuỗi đầu vào thành một vectơ ngữ cảnh cố định duy nhất, mô hình chú ý phát triển một vectơ bối cảnh được lọc riêng cho từng bước thời gian t ở đầu ra.



1. Cấu trúc chi tiết của một attention [6]
   * 1. Linguistic Input Feature

Mỗi nước mỗi vùng sẽ có những ngôn ngữ khác nhau. Trong các ngôn ngữ đó, mỗi ngôn ngữ đều có nét đặc trưng riêng cho từng loại ngôn ngữ. Ta gọi chung các đặc trưng riêng này là linguistic feature (đặc trưng ngôn ngữ).

Chính vì tính đặc trưng của mỗi ngôn ngữ khác nhau như vậy sẽ dẫn đến tình trạng quá trình dịch sẽ không chính xác do phải phụ thuộc vào nhiều yếu tố như ngữ pháp, đồng nghĩa, đồng âm,…

Vì vậy việc xử lý các đặc trưng này trước khi đưa dữ liệu vào quá trình dịch sẽ giúp cho quá trình dịch trở nên một cách chính xác hơn. Điều này đã được thể hiện rõ trong tài liệu: Rico Sennrich, Barry Haddow, 2016, Linguistic Input Features Improve Neural Machine Translation.

Trong tài liệu Rico Sennrich, Barry Haddow, 2016, Linguistic Input Features Improve Neural Machine Translation giới thiệu cách xử lý 5 loại đặc thù của ngôn ngữ mà từ đó có thể giúp ta có thể cải thiện được độ chính xác của quá trình dịch. 5 loại đặc thù này gồm:

* Lemmas.
* Subword tag.
* Morphological feature.
* POS tag.
* Dependency label.

Tài liệu này xin tập trung chủ yếu vào cách xử lý đặc thù về lenmas (bổ đề).

Thông thường các trạng thái ẩn của encoder sẽ được tính như sau:

Trong đó:

* : vector trạng thái ẩn hiện tại của encoder.
* , : vector trọng số.
* : ma trận word embedding.
* : dữ liệu input.
* : vector trạng thái ẩn của encoder ở bước phía trước (t – 1).
* : kích thước của ma trận word embedding.
* : số lượng tế bào GRU (số lượng trạng thái ẩn).
* : kích thước của bộ từ điển.

Khi có thêm các yếu tố đặc thù ngôn ngữ , các trạng ẩn của encoder sẽ được tính như sau:

Trong đó:

* phép nối vector.
* : ma trận word embedding.
* : kích thước của ma trận word embedding.
* : kích thước của bộ từ điển có chứa yếu tố đặc thù ngôn ngữ.

Mô hình tổng quát này có thể sử dụng cho bất kỳ đặc thù ngôn ngữ tùy ý nào mà ta mong muốn.

Trong một ngôn ngữ, một từ có thể được viết lại thành nhiều dạng để phù hợp với ngữ cảnh, ngữ pháp,… Ví dụ trong tiếng anh, ta có từ: interest (thích thú, thú vị,…) đóng vai trò là động từ hoặc danh từ, tuy nhiên khi chuyển về tính từ, ta được: interesting hoặc interested. Ta có thể thấy rằng interesting và interested đều được tạo từ interest. Bổ đề (lemmas – từ gốc) của interesting, interested, interest sẽ là interest.

Dựa trên công thức (1), chính là quá trình ta sẽ thực hiện việc đặc thù hóa dữ liệu trước, sau đó ta sẽ đem đi word embedding.

Vì vậy việc lemmas hóa (Lemmatization) chính là quá trình ta sẽ tìm từ gốc cho toàn bộ các từ trong tập dữ liệu trước khi ta đưa dữ liệu đem đi huấn luyện.

* 1. Tóm tắt chương 1

Trong chương này ta đã tìm hiểu xong cấu trúc cũng như cách thức hoạt động của 1 mô hình dịch máy neural. Mô hình này cơ bản gồm có 2 bộ phận chính là encoder và decoder. Encoder có chức năng dùng để mã hóa một câu văn đầu vào thành 1 chuỗi cố định. Từ chuỗi cố định này decoder có chức năng giải mã và ánh xạ thành từ tương ứng ở ngôn ngữ đích.

Encoder, trong bài viết này gồm 2 dạng chính. Dạng thứ nhất là sử dụng mạng Recurrent Neural Network (RNN) với nhân là GRU. Dạng thứ hai là sử dụng mạng Convolutional Neural Network (CNN). Decoder sử dụng mạng RNN làm mô hình chính.

Ngoài ra ta tìm hiểu được một phương pháp nhằm giúp tối ưu hóa độ chính xác của chương trình đó là sử dụng đặc thù của ngôn ngữ để xử lý dữ liệu đầu vào, ta gọi nó là Lingustic Input Feature. Đặc biệt bài viết giới thiệu cách ta đưa các từ của dữ liệu về dạng gốc của nó dựa trên đặc thù ngôn ngữ của nó là bổ đề (lemmas).

Trong chương sau ta sẽ vận dụng và thực hành viết chương trình thực hiện mô phỏng mô hình encoder-decoder.

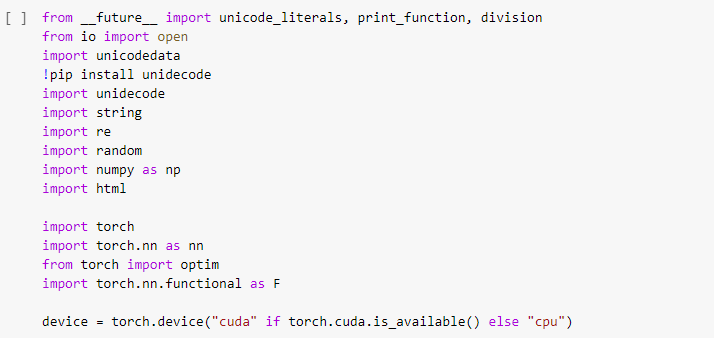
CHƯƠNG 2: THỰC HIỆN MÔ PHỎNG ENCODER-DECODER

2.1 Chuẩn bị nguồn dữ liệu và thư viện

Nguồn dữ liệu được lấy từ trang web: https://github.com/stefan-it/nmt-en-vi/tree/master/data

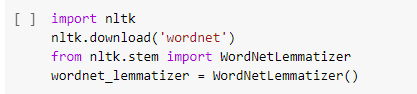
Ta sẽ thực hiện xây dựng và huấn luyện mô hình dựa trên 2 file chính đó là: train.en và train.vi.

Các thư viện ta cần chuẩn bị để chạy chương trình:



1. Các thư viện cần chuẩn bị

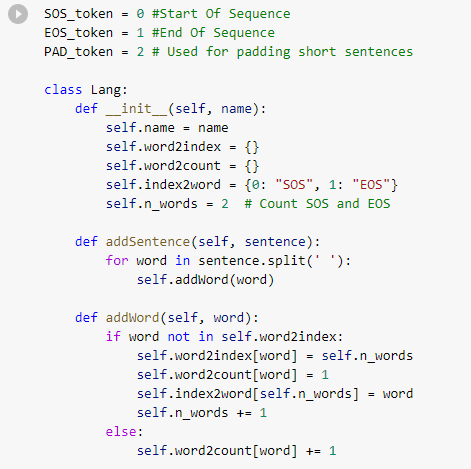
Ngoài ra ta cần chuẩn bị thêm một số thư viện nhằm thực hiện linguistic input feature:



1. Các thư viện phục vụ linguistic input feature

2.2 Xử lý dữ liệu và tạo các đối tượng quản lý

Đầu tiên ta cần tạo 1 class để quản lý các đối tượng dữ liệu:



1. Class quản lý dữ liệu

Class ngôn ngữ gồm có các **thuộc tính** cơ bản sau:

* **name**: tên của ngôn ngữ (English, tiếng Việt, …)
* **word2index**: bộ từ điển.
* **word2count**: bộ đếm từ.
* **index2word**: bộ lưu vị trí các token.
* **n\_words**: đếm tổng các lượng từ.

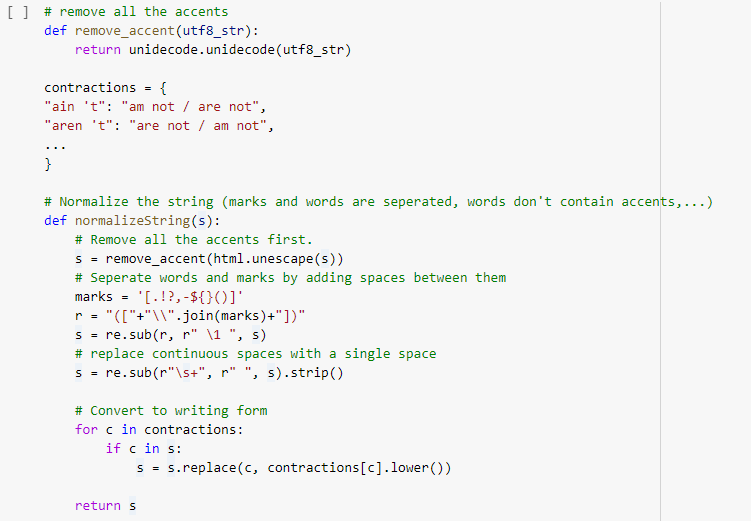
Class ngôn ngữ gồm có các **hàm** cơ bản sau:

* **addSentence**: Có tham số là sentence, dùng để truyền vào 1 câu văn bản. Từ câu văn bản này, ta sẽ tách thành các từ (word). Cuối cùng ta sẽ lưu các từ này vào bộ từ điển bằng hàm addWord.
* **addWord**: Có tham số là word, dùng để truyền vào 1 từ (word), nếu từ này chưa có trong bộ từ điển thì sẽ được thêm (add) mới vào bộ từ điển. Nếu đã có rồi ta sẽ tăng số lần xuất hiện (số đếm) của từ này thêm 1 đơn vị.

Ta cần triển khai thêm một số hàm bổ trợ như:

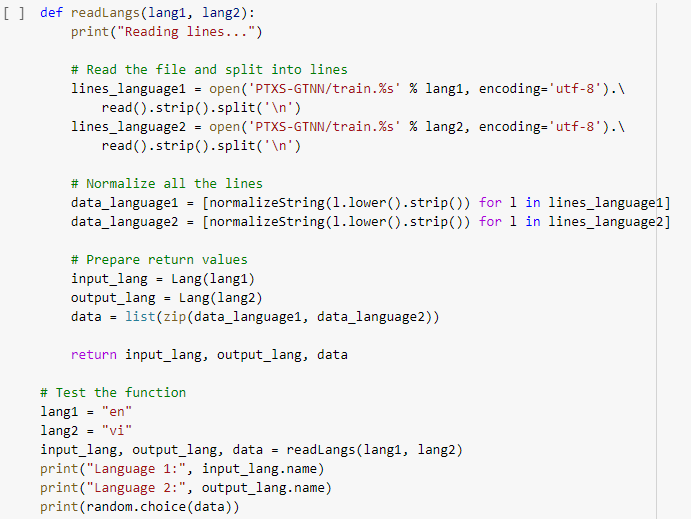
* **remove\_accent**: dùng để loại bỏ các dấu thanh (sắc, hỏi, huyền, ngã, nặng, ...).
* **normalizeString**: Dùng để chuẩn hóa dữ liệu để chạy chương trình. Đầu tiên ta sẽ loại bỏ hết các dấu thanh ra khỏi câu, đồng thời ta chuyển các đối tượng html (html entity) về ngôn ngữ tự nhiên. Sau đó ta sẽ tách các dấu câu (dấu phẩy, dấu chấm, dấu chấm hỏi, dấu chấm thang, ...) ra khỏi từ. Ngoài ra ta sẽ chuyển bớt một số dạng thể ngắn của tiếng anh về dạng thường.

Ví dụ ta có câu: "Ăn quả, nhớ kẻ trồng cây.". Sau khi chạy hàm **normalizeString** ta sẽ được kết quả là: "An qua , nho ke trong cay ."



1. Xử lý và chuẩn hóa dữ liệu

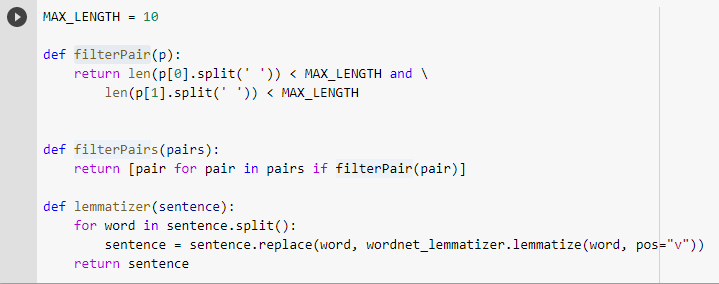
Tiếp theo ta sẽ viết hàm nạp dữ liệu vào chương trình:



1. Hàm đọc dữ liệu

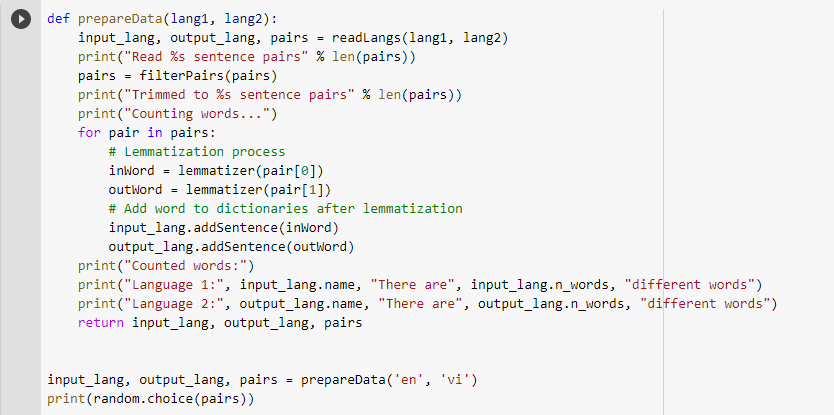
Ta cần viết thêm một số hàm bổ trợ cho việc xử lý dữ liệu như:

* filterPair: phân các câu dữ liệu dài thành các câu dữ liệu ngắn có độ dài giới hạn (MAX\_LENGTH).
* Lemmatizer: thực hiện lemmas hóa dữ liệu.



1. Chuẩn hóa số lượng từ của câu và lemmas hóa dữ liệu

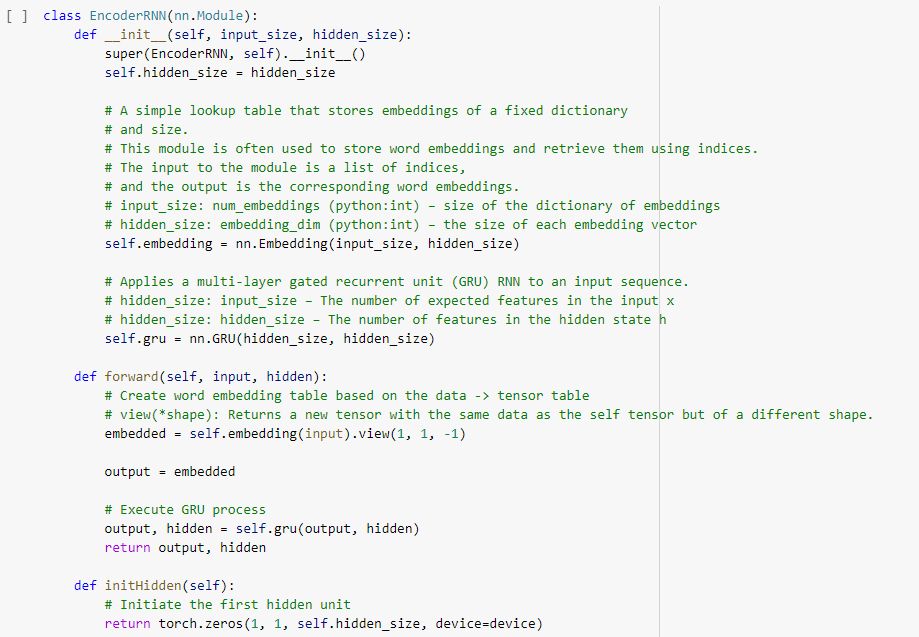
Cuối cùng ta sẽ nạp dữ liệu vào chương trình:



1. Hàm đọc dữ liệu và thực hiện nạp dữ liệu vào chương trình

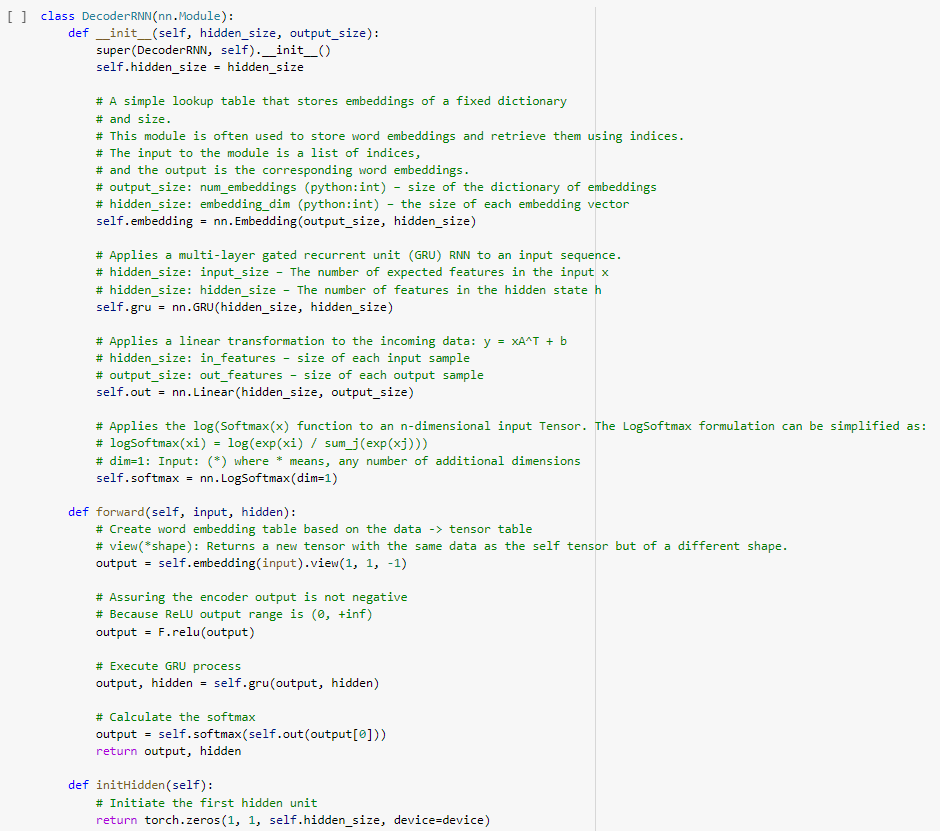
2.3 Xây dựng encoder-decoder

Đầu tiên ta sẽ đi xây dựng bộ encoder:



1. Xây dựng bộ encoder

Tiếp theo ta đi xây dựng bộ decoder:



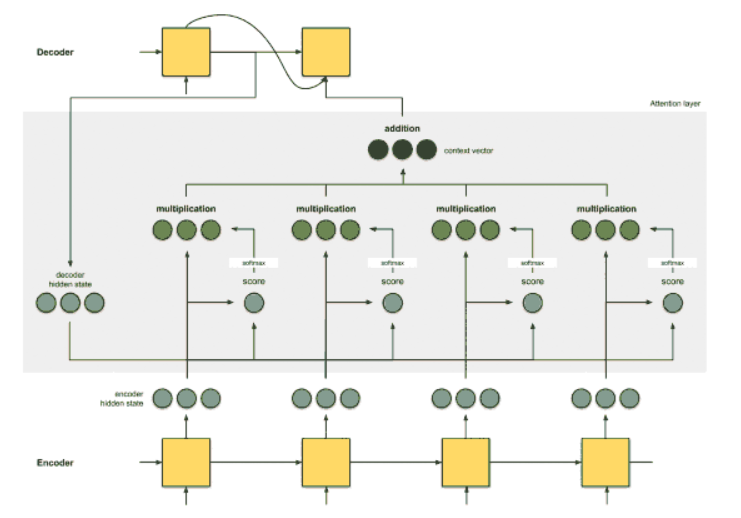
1. Xây dựng bộ decoder cơ bản

Attention là một cơ chế kỹ thuật nhằm giúp giải quyết các giới hạn của kiến trúc encoder-decoder trên các chuỗi dài (chuỗi dữ liệu đầu vào). Attention sẽ giúp tăng tốc độ học tập và cải thiện khả năng dự đoán của mô hình.

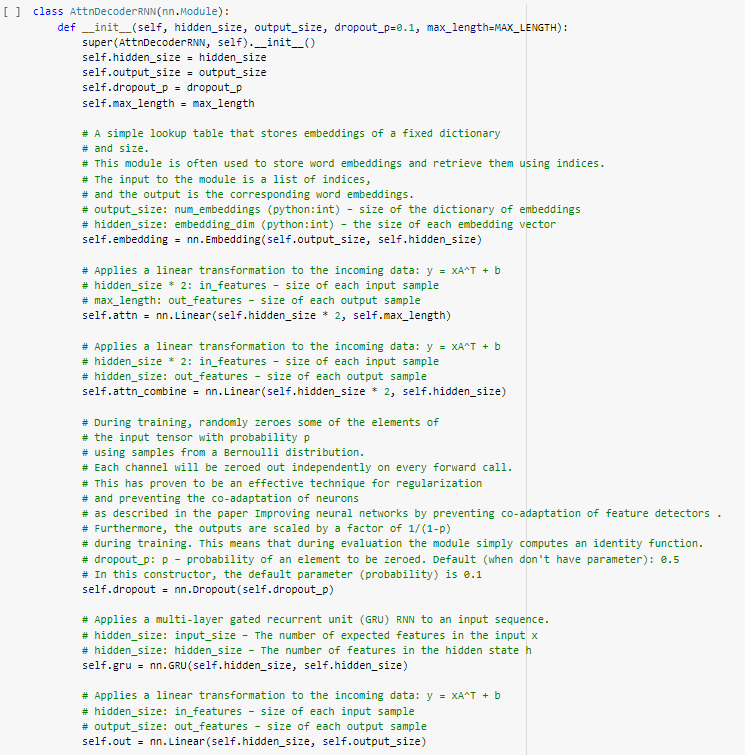
Do vector ngữ cảnh này có độ dài cố định và là kết quả sau quá trình mã hóa từ dữ liệu đầu vào có độ dài thay đổi, nên vector này sẽ chứa rất nhiều thông tin. Việc mã hóa (decode) vector này sẽ là một áp lực lớn đối với decoder.

Cơ chế attention cho phép tại mỗi bước input của decoder, decoder có thể tập trung vào từng phần khác nhau của encoder output. Đầu tiên, chúng ta sẽ tính tập trọng số attention (attention weight). Sau đó ta sẽ lấy tập trọng số này nhân với các vectơ đầu ra của encoder để tạo ra một tổ hợp có trọng số. Kết quả (được gọi là attn\_applied) này giúp nhận biết được từng phần của chuỗi đầu vào, và từ đó sẽ giúp decoder chọn được từ (word) đầu ra đúng.

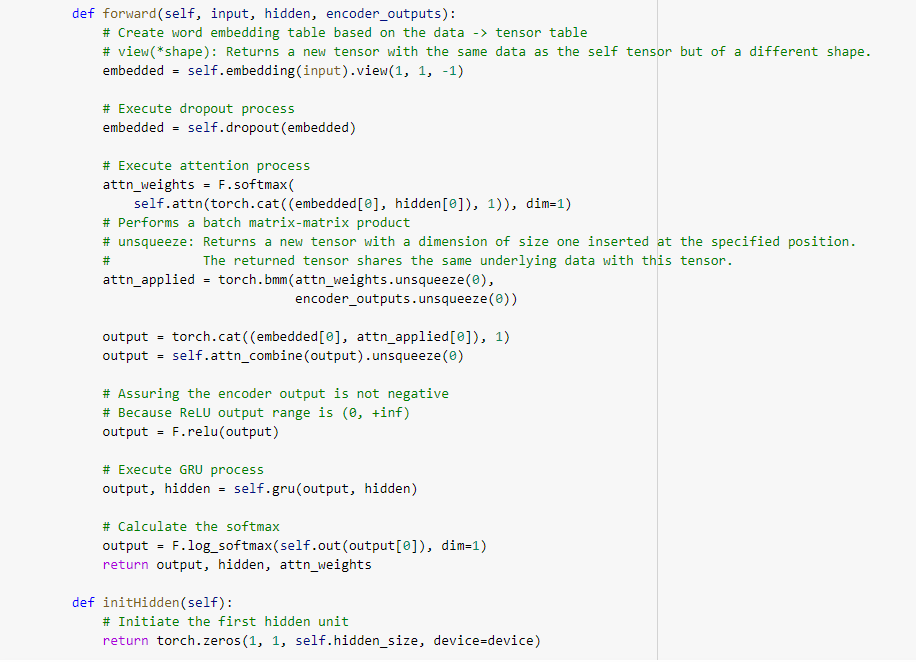
Thay vì mã hóa chuỗi đầu vào thành một vectơ ngữ cảnh cố định duy nhất, mô hình chú ý phát triển một vectơ bối cảnh được lọc riêng cho từng bước thời gian t ở đầu ra.



1. Cấu trúc hoạt động của attention



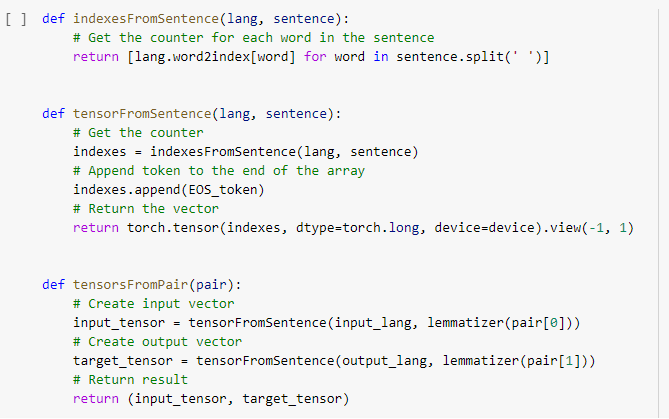
1. Class decoder sử dụng attention (hàm khởi tạo của class)



1. Hàm khởi tạo tiến trình và hàm feedforward

2.4 Huấn luyện

Để huấn luyện, đối với mỗi cặp, chúng ta sẽ cần một tensor (vector) đầu vào (index của các từ trong câu) và tensor đích (index của các từ trong câu đích). Trong khi tạo các vector này, ta sẽ nối thêm các mã thông báo token vào cả hai chuỗi.

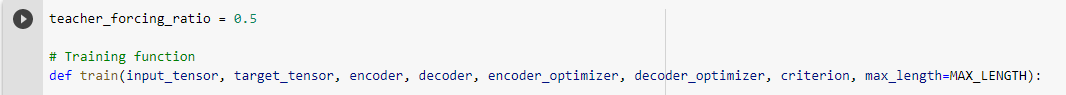


1. Các hàm tạo vector input và output cho quá trình huấn luyện

Để đào tạo, ta sẽ đưa dữ liệu đầu vào đi qua encoder, ta sẽ tiến hành theo dõi các kết quả đầu ra cũng như các giá trị trạng thái ẩn của encoder. Sau đó, ta sẽ cung cấp cho decoder token thông báo làm tín hiệu thông báo cho decoder khởi tạo đầu vào đầu tiên, đồng thời trạng thái ẩn cuối cùng của encoder sẽ làm trạng thái ẩn đầu tiên cho decoder.

"Teacher forcing" là một khái niệm ám chỉ về việc ta sẽ sử dụng các giá trị đầu ra mong muốn cho mỗi đầu vào tiếp theo, thay vì ta sử dụng kết quả mà decoder dự đoán được cho các đầu vào tiếp theo. Việc làm này sẽ khiến cho chương trình hội tụ nhanh hơn, trong một số trường hợp việc làm dụng điều này sẽ khiến cho chương trình không ổn định (tham khảo thêm trong tài liệu của Herbert Jaeger, 10/2008, A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the “echo state network” approach).

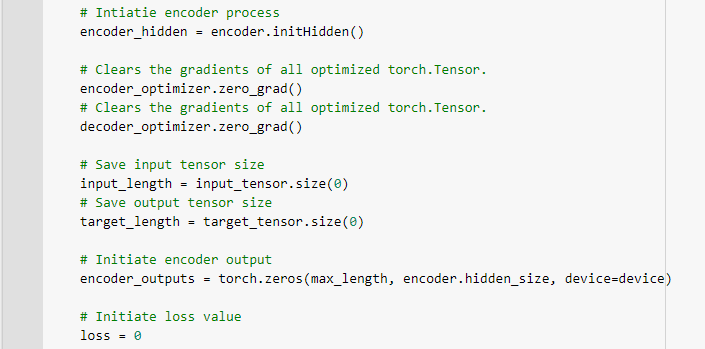
Theo trực giác, ta có thể quan sát kết quả đầu ra của các mạng "teacher-forcing" có cấu trúc ngữ pháp ổn định tuy nhiên kết quả dịch lại không được chính xác. Điều này cho thấy rằng mạng "teacher-forcing" có thể học được cấu trúc ngữ pháp, tuy nhiên việc dịch nghĩa của nó lại không được tốt.



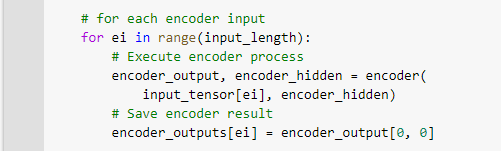
1. Hàm thực hiện huấn luyện

Hàm này gồm 5 giai đoạn cơ bản:

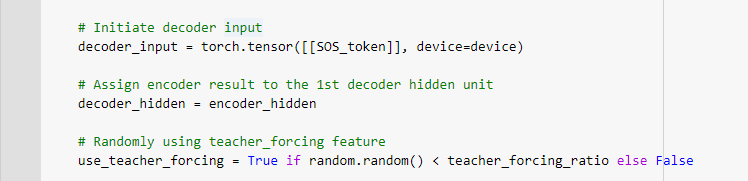
* Tiền encoder: khởi tạo encoder (khởi tạo các vector input và output của encoder), xóa các giá trị gradient descent cũ tồn đọng trong các lần chạy trước, khởi tạo giá trị mất mát (loss).
* Chạy encoder: thực hiện cho dữ liệu chạy vào encoder và tiến hành cho encoder hoạt động.
* Hậu encoder và tiền decoder: khởi tạo các vector input và các trạng thái ẩn của decoder, gán kết quả encoder vào trạng thái ẩn của decoder, khởi tạo giá trị teacher\_forcing.
* Chạy decoder: chạy decoder.
* Hậu decoder: tính toán lỗi và thực hiện lan truyền ngược để tính gradient descent.



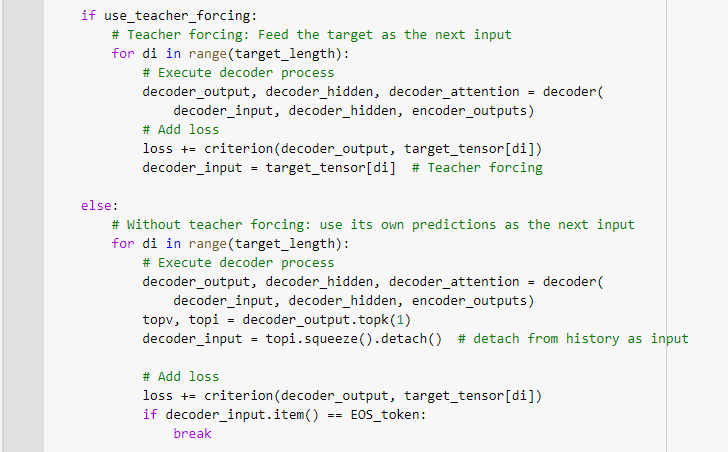
1. Giai đoạn tiền encoder



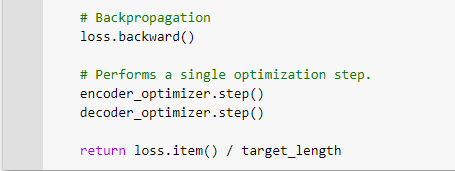
1. Giai đoạn chạy encoder



1. Giai đoạn hậu encoder và tiền decoder



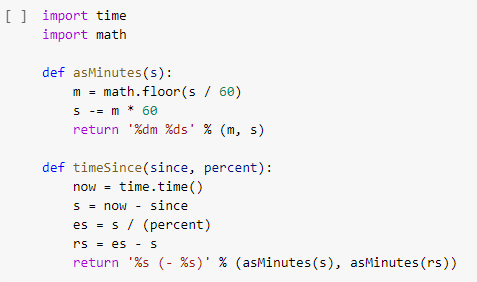
1. Giai đoạn chạy decoder



1. Giai đoạn hậu decoder

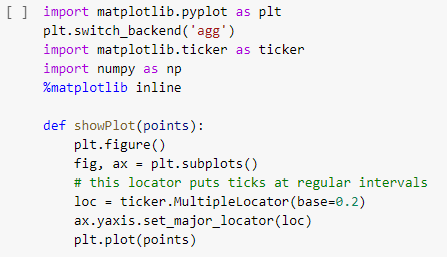
Như vậy ta đã thực hiện xong cho 1 lần huấn luyện, sau 1 lần huấn luyện hàm sẽ trả về kết quả là phép chia giữa số từ mà chương trình dự đoán sai với số lượng từ có trong câu dịch đích.

Ta có thể theo dõi lượng thời gian chương trình tiến hành huấn luyện bằng một số hàm sau:



1. Hàm thông báo lượng thời gian chương trình đã và đang chạy

Ta viết thêm hàm vẽ biểu đồ biểu diễn sự thay đổi của giá trị loss:

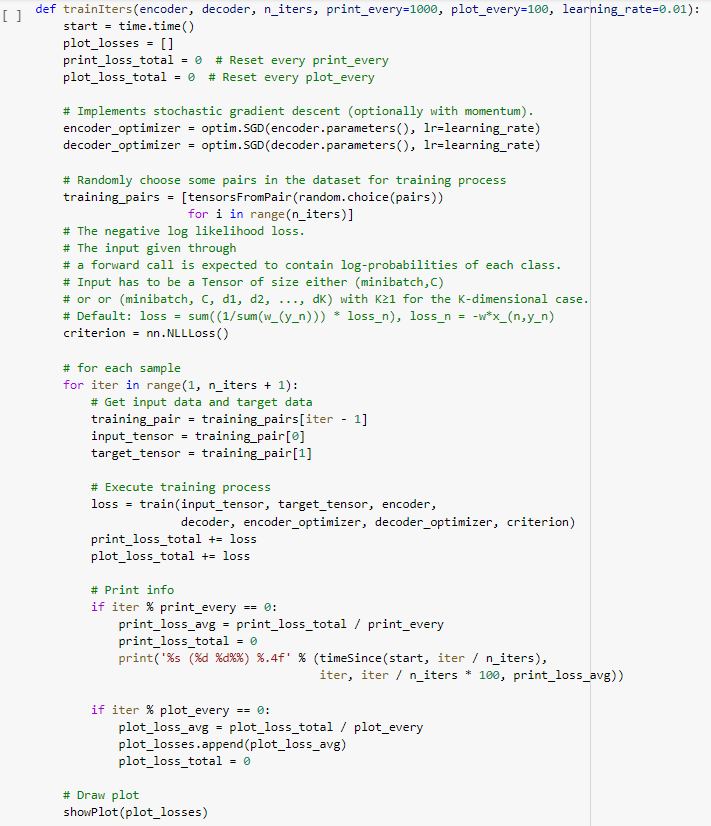


1. Hàm vẽ biểu đồ thay đổi của giá trị loss

Toàn bộ quá trình đào tạo trông như thế này:

* Bắt đầu tính thời gian huấn luyện.
* Khởi tạo bộ tối ưu hóa (optimizer) và bộ tiêu chuẩn (criterion).
* Tạo một tập các cặp đào tạo (training pair).
* Khởi tạo mảng loss lưu từng giá trị loss tại các thời điểm nhằm phục vụ cho việc vẽ biểu đồ.

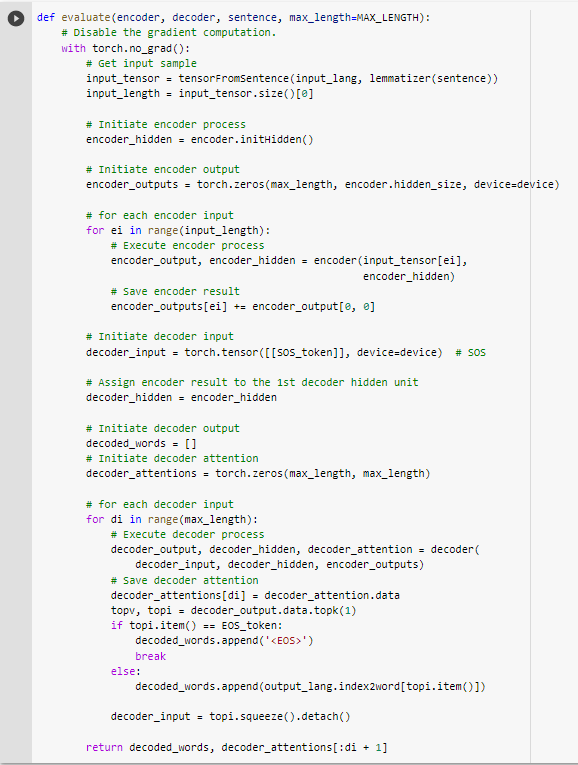
Cuối cùng ta sẽ gọi hàm train để thực hiện quá trình đào tạo. Kết thúc 1 lượt đào tạo ta sẽ thông báo một vài giá trị như giá trị mất mát (loss) trung bình hiện tại, thời gian huấn luyện hiện tại, ...



1. Hàm huấn luyện tổng quát

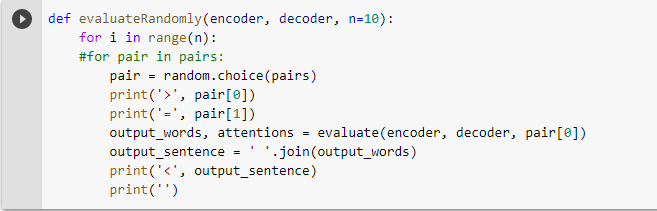
2.5 Đánh giá

Quá trình đánh giá chủ yếu cũng gần giống như đào tạo, tuy nhiên nó khác với quá trình đào tạo ở chỗ nó sẽ không đi so sánh giữa kết quả dự đoán với giá trị mong muốn (target data). Quá trình đánh giá đơn giản chỉ là ta sẽ đem các giá trị dự đoán mà decoder đã thực hiện được nạp vào lại chính nó tại mỗi bước. Mỗi lần decoder dự đoán một từ, ta sẽ thêm từ đó vào chuỗi đầu ra, khi decoder dự đoán mã thông báo (token) EOS, ta sẽ tiến hành dừng quá trình thực hiện. Chúng ta cũng nên cần lưu trữ các giá trị đầu ra của decoder attention nhằm phục vụ cho các yêu cầu hiển thị về sau.



1. Hàm thực hiện đánh giá

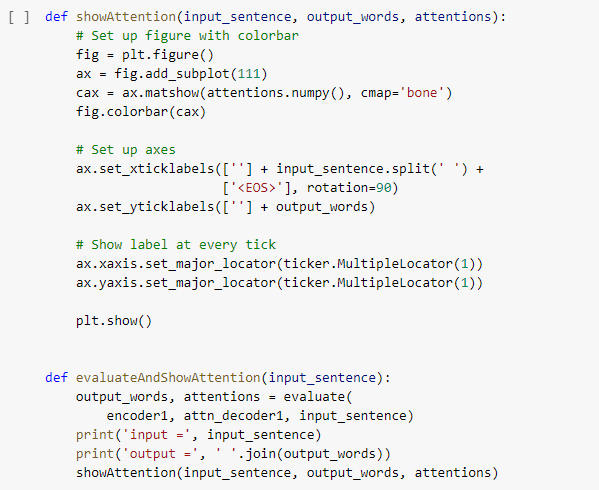
Ta có thể thực hiện đánh giá bằng việc chọn ngẫu nhiên một số dữ liệu từ tập dữ liệu:



1. Hàm thực hiện chọn đánh giá ngẫu nhiên

2.6 Kiểm thử

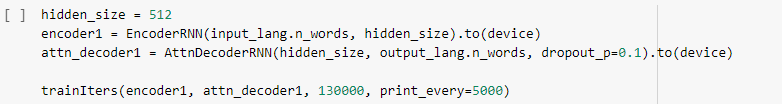
Ta sẽ viết các hàm phục vụ cho kiểm thử như biểu diễn các giá trị của attention, hàm biểu diễn mô phỏng của attention, …



1. Các hàm phục vụ trình bày kết quả

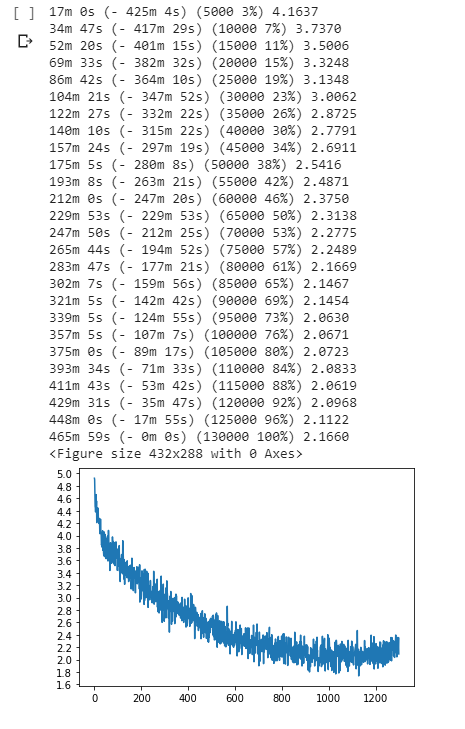
2.7 Chạy chương trình và kết quả

Ta cho chương trình chạy huấn luyện:



1. Tiến hành huấn luyện mô hình

Sau khi chạy ta thu được kết quả:



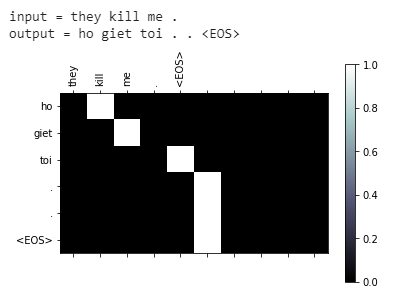
1. Kết quả của từng đợt huấn luyện

Cuối cùng ta sẽ chạy thử trên một số câu ví dụ:



1. Chạy thử 1 câu ví dụ

Kết quả sau khi chạy kiểm thử:



1. Kết quả chạy thử 1 câu ví dụ

2.8 Tóm tắt

Ta đã tìm hiểu xong cách viết chương trình mô phỏng cho quá trình encoder và decoder.

Đầu tiên ta sẽ xử lý và chuẩn hóa dữ liệu. Ta sẽ phải đưa các kí tự Unicode về dạng asci chuẩn. Sau đó ta sẽ lọc ra các cặp câu có lượng từ trong câu nằm trong phạm vi giới hạn của MAX\_LENGTH vì dịch các câu càng ngắn sẽ giúp làm tăng tốc độ thực hiện của chương trình.

Sau khi làm sạch cũng như chuẩn hóa dữ liệu, ta sẽ tiến hành đi xây dựng bộ encoder và decoder. Sau khi xây dựng xong ta sẽ tiến hành đi huấn luyện cho mô hình.

Sau khi huấn luyện mô hình xong ta sẽ tiến hành kiểm tra, đánh giá và kiểm thử kết quả mà chương trình chạy và đạt được.

Như vậy qua 2 chương ta đã tìm hiểu được cấu tạo, nguyên lý hoạt động của quá trình dịch máy dựa trên mô hình encoder-decoder đồng thời ta cũng biết được cách thức vận dụng mô hình vào việc viết chương trình.

TÀI LIỆU THAM KHẢO

**Tiếng Anh:**

1. Trevor Hastie, Robert Tibshirani, Jerome Friedman, 2008, The Elements of Statistical Learning: Data Mining, Inference, and Prediction (Second Edition).