TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÁO CÁO MÔN HỌC
HỌC SÂU

ỨNG DỤNG MÔ HÌNH BILSTM TRONG BÀI TOÁN GÁN NHÃN TỪ LOẠI

Giảng viên hướng dẫn: **TS. Trần Việt Trung**

Học viên thực hiện: **Nguyễn Đức Thắng**

Mã số học viên: **20166769**

HÀ NỘI, 4/2020

Mục lục

Lời mở đầu

Trong những năm gần đây, trí tuệ nhân tạo ngày càng phát triển và đi vào cuộc sống hàng ngày. Trong đó, xứ lý ngôn ngữ tự nhiên (natural language processing - NLP) là một nhánh của trí tuệ nhân tạo tập trung vào các ứng dụng trên ngôn ngữ của con người. Trong trí tuệ nhân tạo thì xử lý ngôn ngữ tự nhiên là một trong những phần khó vì nó liên quan đến việc phải hiểu ý nghĩa của ngôn ngữ - công cụ hoàn hảo nhất của tư duy và giao tiếp. Các bài toán ứng dụng của xử lý ngôn ngữ tự nhiên bao gồm: nhận dạng chữ viết, nhận dạng tiếng nói, dịch máy, tìm kiếm thông tin, tóm tắt văn bản tự động, khai phá dữ liệu...

Gán nhãn từ loại là một trong những bài toán cơ bản của xử lý ngôn ngữ tự nhiên, nó là bước tiền xử lý quan trọng có vai trò ảnh hưởng trực tiếp tới các phân tích văn phạm ở mức sâu. Trong bài báo cáo này, với mục đích nghiên cứu về nhánh xử lý ngôn ngữ tự nhiên và ứng dụng framework Tensorflow, tôi đề xuất đề tài: *Ứng dụng mô hình BiLSTM trong bài toán gán nhãn từ loại.*
Bố cục báo cáo gồm 3 chương:

- **Chương 1: Các kiến thức cơ sở** - Giới thiệu về bài toán và cung cấp các kiến thức cơ sở liên quan đến đề tài.
- **Chương 2: Mô hình BiLSTM** - Giới thiệu về tư tưởng và cách thức hoạt động của mô hình Bidirectional LSTM - Mô hình chính được sử dụng trong báo cáo này.
- **Chương 3: Thực nghiệm mô hình cho bài toán gán nhãn từ loại** - Quy trình và các bước thực hiện triển khai bài toán.

Chân thành cảm ơn TS. Trần Việt Trung - người đã hướng dẫn tôi trong suốt môn học, cung cấp cho tôi những kiến thức nền tảng về học sâu để tôi có thể hoàn thiện báo cáo này.

Mặc dù đã cố gắng rất nhiều, nhưng với kiến thức còn hạn chế nên không thể tránh khỏi những thiếu sót. Rất mong được sự góp ý của thầy cô và bạn bè để báo cáo này được hoàn thiện hơn.

Chương 1

Các kiến thức cơ sở

1.1 Giới thiệu về Tensorflow

Tensorflow là một thư viện mã nguồn mở dùng để xử lí cách phép tính toán số học bằng cách mô tả một mô hình biểu đồ thể hiện sự thay đổi về giá trị của dữ liệu. Đây là một thư viện toán học được sử dụng nhiều trong các ứng dụng của học máy chẳng hạn như xây dựng các mạng nơ ron, các thuật toán phân loại kNN (k-Nearest Neighbor), SVM (Support Vector Machine),...
Tiền thân của Tensorflow là DistBelief - dự án về hệ thống học máy của Google Brain được phát triển vào năm 2011. Tensorflow là dự án thứ 2, được phát hành dưới dạng mã nguồn mở vào 09/11/2015.
Một số project nổi tiếng sử dụng Tensorflow như:

- Phân loại ung thư da - Dermatologist-level classification of skin cancer with deep neural networks (Esteva et al., Nature 2017).
- WaveNet: Text to speech – Wavenet: A generative model for raw audio (Oord et al., 2016).
- Vẽ hình – Draw Together with a Neural Network (Ha et al., 2017).
- Image Style Transfer Using Convolutional Neural Networks (Gatys et al., 2016) Tensorflow adaptation by Cameroon Smith (cysmith@github)

Tensorflow là luông của các dữ liệu thể hiện qua một đồ thị tính toán. Những đặc điểm nổi bật của tensorflow có thể kể đến như khả năng sử dụng trên các platform khác nhau từ smartphone, PC tới distributed servers. Thư viện này có thể được chạy với CPU hoặc cùng với GPU, cùng với danh sách API rất dễ sử dụng, cho phép tính toán nhanh chóng trên mạng neural.
Cũng chính Google phải nhận định: *"Tensorflow nhanh hơn, thông minh hơn và linh hoạt hơn hệ thống cũ của chúng tối (Disbelief), do đó nó có thể được điều chỉnh dễ dàng hơn với các sản phẩm mới và giúp quá trình nghiên cứu diễn ra thuận lợi hơn".*

Vì những sức mạnh như vậy của Tensorflow, nên trong báo cáo này, tôi lựa chọn tensorflow là core cho bài toán của mình.

1.2 Giới thiệu bài toán

Trong nhiều bài toán xử lý ngôn ngữ tự nhiên (NLP), ta mong muốn xây dựng được một mô hình mà chuỗi các quan sát (câu, từ ngữ...) đi kèm với chuỗi các nhãn đầu ra (từ loại, ranh giới từ, tên thực thể,...) gọi là pairs of sequences. Trong việc gán nhãn, mục tiêu của chúng ta là xây dựng mô hình đâu vào là một câu, ví dụ như câu "con ruồi đậu mâm xôi đậu":

con ruồi đậu mâm xôi đậu

thì chuỗi đầu ra (tag sequences) sẽ là:

B-NP I-NP B-VP B-NP I-NP I-NP

Trong đó NP (Noun pharse) là cụm danh từ, VP (Verb pharse) là cụm động từ, B đại diện cho bắt đầu thẻ, I đại diện cho từ đó nằm trong thực thể. Trong phần 3 của báo cáo này, chúng ta sẽ tìm hiểu kỹ hơn về ý nghĩa các thẻ.
Gán nhãn (sequence label problem, tagging problem) có các bài toán thường gặp:

- POS tagging (gán nhãn từ loại): Là sở cở phục vụ cho các bài toán về ngữ nghĩa cao hơn.
- Named-Entity recognition (gán nhãn tên thực thể): Ví dụ: bà Ba [CON NGƯỜI] bán bánh mì [THỰC PHẨM] ở quận 1 [ĐỊA ĐIỂM] có giá trị ngữ nghĩa ở mức trung bình, thường dùng để phân loại văn bản.
- Machine translation (dịch máy): Đầu vào là một câu của ngôn ngữ A, đầu ra là câu của ngôn ngữ B tương ứng. Bài toán này từng rất cấp thiết trong chiến tranh thế giới thứ 2, khi mà thông tin tình báo của địch cần được dịch trong thời gian ngắn nhất, giúp cho các lãnh dạo có thể đưa ra những chiến lược cần thiết.
- Speech recognition (nhận diện tiếng nói): Đầu vào là âm thanh tiếng nói, đầu ra là câu dạng văn bản. Ngày nay, theo thống kê của Apple, người dùng thích sử dụng tiếng nói của mình để nhập văn bản hơn là cách nhập dữ liệu bằng bàn phím như truyền thống, đồng thời tương tác giữa người và máy theo cách này có tốc độ nhập liệu nhanh hơn.

Khi làm việc với các bài toán này, chúng ta sẽ đối mặt với hai thách thức chính:

- Nhập nhằng (ambiguity): Một từ có thể có nhiều từ loại, hay một từ có thể có nhiều nghĩa. Ví dụ "con ruồi đậu mâm xôi đậu", từ "đậu" có lúc là động từ (hành động đậu lên một vật thể) hoặc có lúc là danh từ (tên của một loài thực vật).

- Trong thực tế, có nhiều từ không xuất hiện trong ngữ liệu huấn luyện (training corpus) nên khi xây dựng mô hình gán nhãn sẽ gặp nhiều khó khăn. Độ chính xác của mô hình gán nhãn phụ thuộc 2 yếu tố:
  - Bản thấn từ đó sẽ có xu hướng (xác suất lớn) về từ loại nào. Ví dụ từ "đậu" có xu hướng là động từ nhiều hơn là danh từ (phụ thuộc vào ngữ liệu đang xét).
  - Ngữ cảnh trong câu. Ví dụ "con ruồi đậu mẫm xôi đậu". Từ "đậu" có xu hướng là động từ khi theo xâu từ "ruồi" và từ "đậu" có xu hướng là danh từ khi theo sau từ "xôi".

Với những khó khăn trên, mạng hồi quy (Recurrrent Neural Network - RNN) ra đời giúp khắc phục. Sau này, các biến thể cao cấp hơn như BiLSTM góp một phần quan trọng không thể thiếu trong việc xử lý các bài toán trên. Trong báo cáo này, tôi sẽ trình bày mô hình BiLSTM trong gán nhãn từ loại (POS tagging) sử dụng framework Tensorflow 1 trên nền ngôn ngữ lập trình Python.

1.3 Neural Network

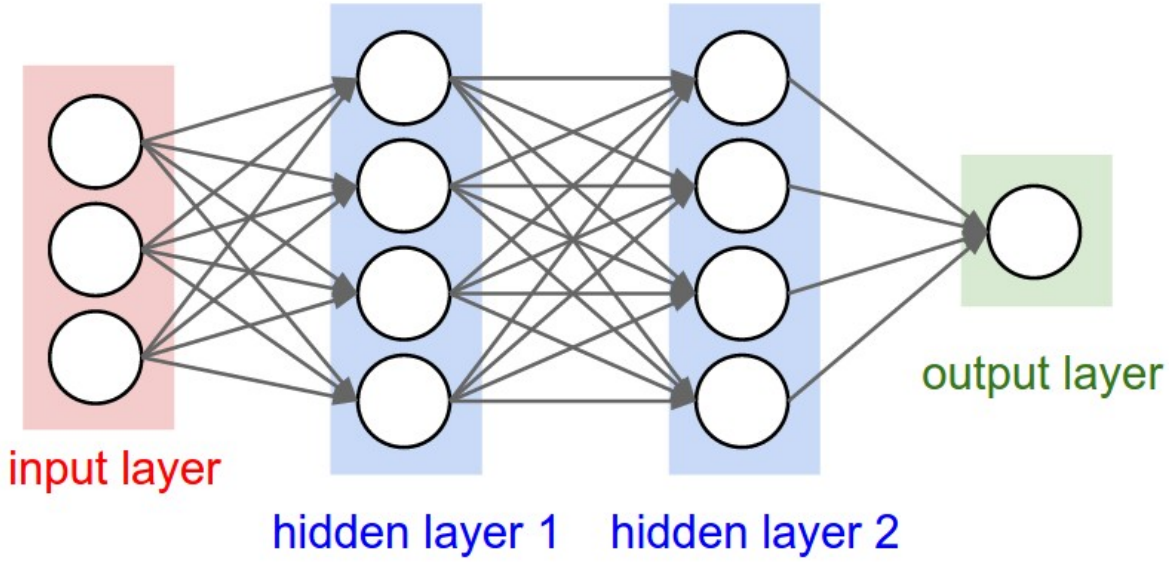
Mạng nơ-ron nhân tạo (Neural Network - NN) là một mô hình lập trình rất đẹp lấy cảm hứng từ mạng nơ-ron thần kinh. Kết hợp với các kĩ thuật học sâu (Deep Learning - DL), NN đang trở thành một công cụ rất mạnh mẽ mang lại hiệu quả tốt nhất cho nhiều bài toán khó như nhận dạng ảnh, giọng nói hay xử lý ngôn ngữ tự nhiên.

1.3.1 Kiến trúc mạng neural

Mạng Neural có kiến trúc như hình 1.1, sẽ gồm có 3 kiểu tầng chính:

- **Tầng vào** *(Input layer)*: Là tầng bên trái cùng của mạng thể hiện cho các đầu vào của mạng.
- **Tầng ra** *(Ouput layer)*: Là tầng bên phải cùng thể hiện cho các đầu ra của mạng.
- **Tầng ẩn** *(Hidden layer)*: Là tầng nằm giữa giữa tầng vào và tầng ra thể hiện cho việc suy luaayjn logic của mạng.

Một mạng neural chỉ có 1 tầng vào và 1 tầng ra, nhưng có thể không có hoặc có thể có nhiều tầng ẩn. Các đặc trưng của input sẽ được đổ vào tầng vào, mỗi nốt của tầng vào tương ứng với 1 đặc trừng của input. Sau đó được lan truyền tiến qua mạng để tính toán ra các đầu ra ở tầng ra, từ tầng đầu ra này, hàm mất mát đánh giá và lại lan truyền ngược lại để cập nhật lại các tham số mạng để cho mạng càng ngày càng dự đoán tốt cho mô hình. Các tham số của mạng chính là các ma trận trọng số liên kết giữa các tầng mạng với nhau. Trong mạng neuron, các nút mạng thường có một



Hình 1.1: Kiến trúc mạng Neural

hàm kích hoạt nào đó và có thể khác nhau. Ở mỗi tầng, số lượng các nút mạng có thể khác nhau tuỳ thuộc vào bài toán và cách giải quyết. Mạng neural giải quyết được các bài toán phân lớp không tuyến tính nhờ vào cơ chế của các hàm kích hoạt.

1.3.2 Hàm kích hoạt

Có rất nhiều hàm kích hoạt trong xây dựng mạng neural. Trong phạm vi của báo cáo này, chúng ta xét đến hai hàm kích hoạt sau:

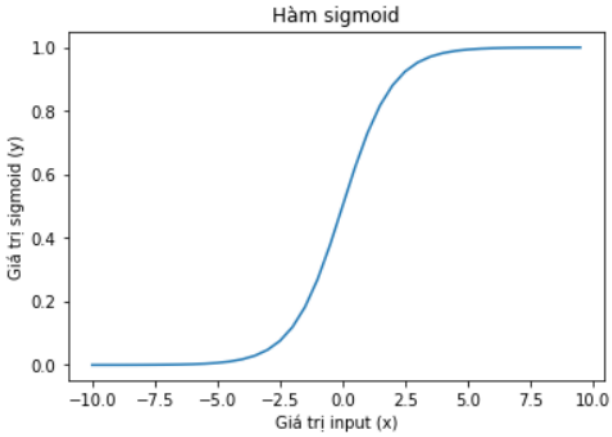
Sigmoid

Hàm sigmoid có đồ thị như hình 1.2 và có công thức như sau:

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

Phân tích hàm sigmoid:

- Giá trị $x \in \mathbb{R}$ và $y \in (0, 1)$. Chúng ta có thể hình dung hàm sigmoid giống như hàm ánh xạ từ $(-\infty, \infty) \to (0, 1)$.
- Hàm sigmoid là hàm không giảm, nghĩa là nếu $x_1 \leq x_2$ thì $f(x_1) \leq f(x_2)$.
- Hàm sigmoid có độ dốc (tương ứng với đạo hàm) lớn xung quanh giá trị $x = 0$ và độ dốc rất nhỏ ở 2 đầu. Điều này gây ra vấn đề khi tính toán dựa vào giá trị đạo hàm. Vì giá trị đạo hàm quá nhỏ nên hầu như không có hữu ích gì.
- Hàm sigmoid không có trung tâm là 0 và gây khó khăn cho việc hội tụ. Tuy nhiên, chúng ta có thể giải quyết vấn đề này bằng cách chuẩn hoá dữ liệu về dạng có trung tâm là 0 (zero centered) với các thuật toán batch/layer normalization.



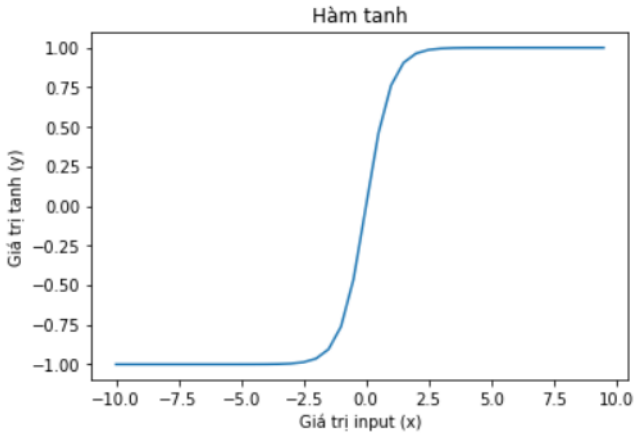
Hình 1.2: Hàm sigmoid

Hành tanh

Hàm tanh có đồ thị như hình 1.3 và có công thức như sau:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Hàm tanh nhận đầu vào là một số thực và chuyển thành một giá trị trong khoảng



Hình 1.3: Hàm tanh

$(-1, 1)$. Cũng như sigmoid, hàm tanh bị bão hoà ở 2 đầu (gradient thay đổi rất ít ở 2 đầu). Tuy nhiên hàm tanh lại đối xứng qua 0 nên khắc phục được một nhược điểm của sigmoid.
Hàm tanh còn có thể biểu diễn bằng hàm sigmoid như sau:

$$tanh(x) = 2\sigma(2x) - 1$$

1.3.3 Lan truyền tiến

Ký hiệu:

- $l^{(i)}$: Số node trong hidden layer thứ i.
- $W^{(k)}$: Kích thước $l^{(k-1)} * l^{(k)}$ là ma trận hệ số giữa layer $k-1$ và layer k, trong đó $w_{ij}^{(k)}$ là hệ số kết nối từ node thứ i của layer $k-1$ đến node thứ j của layer k.
- $b^{(k)}$: Kích thước $l^{(k)} * 1$ là hệ số bias của các node trong layer k, trong đó $b_i^{(k)}$ là bias của node thứ i trong layer k.
- $a^{(l)}$: Các node mạng của tầng thứ l, trong đó $a_j^{(l)}$ là node mạng thứ j của tầng l.

Việc lan truyền tiến trong mạng neural thực hiện từ tầng vào đến tầng ra được thực hiện như sau:

$$z_i^{(l+1)} = \sum_{j=1}^{l^{(l)}} w_{ij}^{(l+1)} a_j^{(l)} + b_i^{(l+1)}$$
$$a_i^{(l+1)} = f(z_i^{(l+1)})$$

Với f là hàm kích hoạt, ta cũng có thể viết lại công thức trên như sau:

$$z^{(l+1)} = W^{(l+1)}.a^{(l)} + b^{(l+1)}$$
$$a^{(l+1)} = f(z^{(l+1)})$$

1.3.4 Lan truyền ngược

Cũng tương tự như các bài toán học máy khác thì quá trình học vẫn là một mô hình tối thiểu hàm lỗi để có được các trọng số hợp lý nhất. Chúng ta cần tìm ra các trọng số của ma trận W và bias b. Hàm mất mát tuỳ vào bài toán mà lựa chọn, trong các bài toán phân lớp thì hàm mất mát thường là hàm *cross entropy*. Để tối ưu hàm mất mát, ta sử dụng các phương pháp như gradient descent hoặc biến thể mạnh mẽ của nó gần đây như: Adam, AMSGrad ...
Sau khi lan truyền tiến thì cần tính đạo hàm hàm lỗi để lan truyền ngược lại và cập nhật lại các tham số của mô hình bằng các phương pháp tối ưu đã kể trên.
Gọi J là hàm mất mát của mô hình. Quá trình lan truyền ngược được thực hiện như sau:

Tính toán đạo hàm theo z ở tầng đầu ra:

$$\frac{\partial J}{\partial z^{(L)}} = \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}}$$

Lan truyền ngược:

Tính đạo hàm theo z ngược lại từ $l = (L-1) \to 2$ theo công thức:

$$\begin{aligned}\frac{\partial J}{\partial z^{(l)}} &= \frac{\partial J}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial z^{(l)}} \\ &= \left(\left(W^{(l+1)} \right)^T \frac{\partial J}{\partial z^{(l+1)}} \right) \frac{\partial a^{(l)}}{\partial z^{(l)}}\end{aligned}$$

Tính đạo hàm hàm mất mát theo các tham số:

Tính đạo hàm theo tham số W bằng công thức:

$$\begin{aligned}\frac{\partial J}{\partial W^{(l)}} &= \frac{\partial J}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial W^{(l)}} \\ &= \frac{\partial J}{\partial z^{(l)}} \left(a^{(l-1)} \right)^T\end{aligned}$$

Tính toán đạo hàm theo tham số b theo công thức:

$$\begin{aligned}\frac{\partial J}{\partial b^{(l)}} &= \frac{\partial J}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial b^{(l)}} \\ &= \frac{\partial J}{\partial z^{(l)}}\end{aligned}$$

1.3.5 Dropout

Mạng neural hoàn toàn có thể bị overfitting (quá khớp) nếu như mô hình thiết lập quá lớn quá mức cần thiết. Overfitting là hiện tượng mạng chạy tốt và hiệu quả trên tập train nhưng lại kém ở trên tập test. Để khắc phục vấn đề này, một trong số những phương pháp phổ biến gọi là dropout (bỏ nút mạng) được đưa ra. Đây là một kỹ thuật rất đơn giản và cho kết quả rất khả quan. Ý tưởng của phương pháp này là trong quá trình huấn luyện, ta bỏ ngẫu nhiên đi một vài nút mạng nhằm giảm độ phức tạp của mạng.

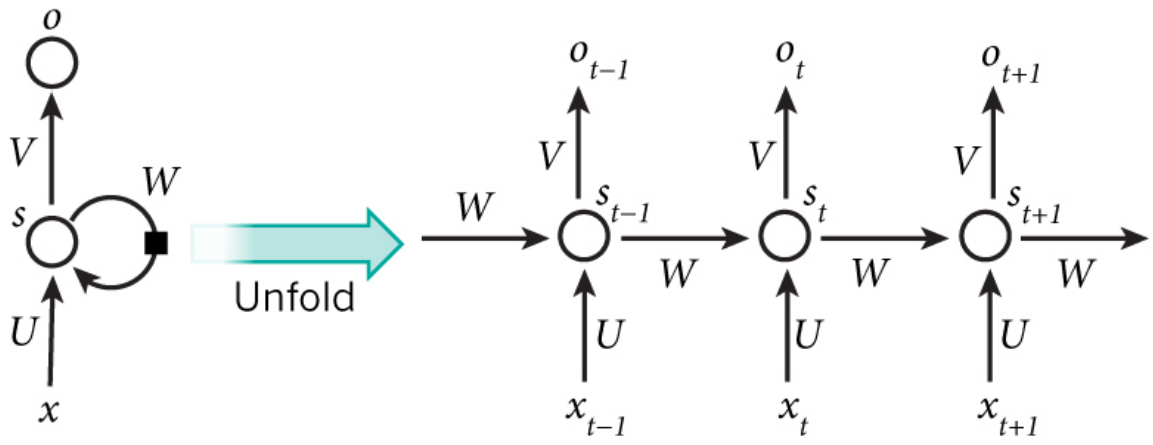
Ta có thể coi mạng sau khi bỏ đi các nút đó là một mạng mới tinh gọn hơn mạng gốc. Như vậy, Với mỗi các lô dữ liệu huấn luyện khác nhau mà ta thực hiện với các mạng tinh giản khác nhau thì kết quả ta thu được sẽ là một mạng trung bình của các mạng tinh gọn đó. Bằng việc lấy mạng trung bình đó, thì ta có thể hi vọng rằng mạng của ta có thể tổng quát được nhiều trường hợp hơn hay nói cách khác là bớt được vấn đề quá khớp.

Tuy nhiên một điểm cần lưu ý là ta không được bỏ bất kì nút mạng nào ở tầng ra, bởi đầu ra của ta cần phải ở dạng mã hoá đầy đủ.

1.4 Recurrent Neural Network (RNN)

1.4.1 Kiến trúc mạng RNN

Ở phần trước, tôi đã giới thiệu về mạng neural, mô hình hoạt động tốt trong nhiều bài toán và là bước nhảy vọt trong lĩnh vực học sâu. Tuy nhiên, các mô hình này vẫn có hạn chế vì các đầu vào chúng ta đưa ra hoàn toàn độc lập với nhau. Trên thực tế, rất nhiều dữ liệu cũng như sự vật trong cuộc sống có mối tương quan chứ không hề độc lập với nhau. Ví như như một video gồm nhiều frame nối tiếp nhau, để có thể nhận dạng một người đang có hành động gì trong video thì không thể chỉ dựa vào một frame ảnh mà cần dựa vào 1 chuỗi các frame ảnh để kết luận. Âm nhạc cũng là một dạng chuỗi tuần tự dạng âm thanh, các văn bản xử lý dạng chuỗi tuần tự,... Câu hỏi đặt ra là vậy làm thế nào chúng ta có thể làm cho các mô hình của chúng ta có khả năng xử lý chuỗi theo cách làm của con người?



Hình 1.4: Mô hình mạng RNN

RNN được tạo ra với mục đích nắm bắt được thông tin dạng chuỗi. Simple Recurrent Network (SRN) được giới thiệu lần đầu bởi Jeff Elman trong paper "Finding structure in time" (Elman, 1990). RNN được gọi là hồi quy (Recurrent) bởi lẽ chúng thực hiện cùng một tác vụ cho tất cả các phần tử của một chuỗi với đầu ra phụ thuộc vào cả các phép tính trước đó. Nói cách khác, RNN có khả năng nhớ các thông tin được tính toán trước đó. Trên lý thuyết, RNN có thể sử dụng được thông tin của một văn bản rất dài, tuy nhiên thực tế thì nó chỉ có thể nhớ được một vài bước trước đó (ta cùng bàn cụ thể vấn đề này sau) mà thôi. Về cơ bản một mạng RNN có dạng như hình 1.4

1.4.2 Lan truyền tiến

Mô hình trên mô tả phép triển khai nội dung của một RNN. Triển khai ở đây có thể hiểu đơn giản là ta vẽ ra một mạng nơ-ron chuỗi tuần tự. Ví dụ ta có một câu gồm 6 chữ "Đại học Bách Khoa Hà Nội", thì mạng nơ-ron được triển khai sẽ gồm 6 tầng nơ-ron tương ứng với mỗi chữ một tầng. Lúc đó việc tính toán bên trong RNN được thực hiện như sau:

- x_t là đầu vào bước t. Ví dụ, x_1 là một vector one-hot tương ứng với từ thứ 2 của câu (học).
- s_t là trạng thái ẩn tại bước t. Nó chính là bộ nhớ của mạng, s_t được tính toán dựa trên cả các trạng thái ẩn phía trước và đầu vào tại bước đó: $s_t = f(Ux_t + Ws_{t-1})$. Hàm f thường là một hàm phi tuyến như tanh hoặc ReLu... Để làm phép toán cho phần tử đầu tiên, ta cần khởi tạo thêm s_{-1}, thường giá trị khởi tạo là 0.
- o_t là đầu ra tại bước t. Ví dụ ta muốn dự đoán từ tiếp theo có thể xuất hiện trong câu thì o_t chính là một vector xác suất các từ trong danh sách từ vựng của ta: $o_t = g(Vs_t)$. g thường là hàm softmax trong bài toán phân loại.

Tóm tắt lại công thức:

$$s_{t+1} = f(Ux_{t+1} + Ws_t)$$
$$o_{t+1} = g(Vs_{t+1})$$

Có thể có thêm các hệ số bias trong công thức trên, trong đó:

- x_t là input vector tại bước t.
- s_t là vector lớp ẩn tại bước t.
- o_t là output vector tại bước t.
- U, V, W là các ma trận tham số ta cần học.
- f, g là các hàm kích hoạt.

1.4.3 Lan truyền ngược

Ở phần trên, chúng ta đã có công thức lan truyền tiến như sau:

$$s_{t+1} = f(Ux_{t+1} + Ws_t)$$
$$o_{t+1} = g(Vs_{t+1})$$

Trong phần này, chúng ta sẽ tính toán đạo hàm cho các tham số để lan truyền ngược cho mô hình. Gọi $o_{t+1} = \hat{y}$ là đầu ra cuối cùng của mạng RNN.

Trong RNN, loss của cả mô hình bằng tổng loss của mỗi output. Các tham số chúng ta cần phải tìm là U, V, W. Ta cần tính:

$$\frac{\partial L}{\partial U}, \frac{\partial L}{\partial V}, \frac{\partial L}{\partial W}$$

Tính đạo hàm với V thì khá đơn giản, ta có o_{t+1} là đầu ra cuối cùng của chúng ta. Do đó:

$$\frac{\partial L}{\partial V} = \frac{\partial L}{\partial o_{t+1}} * \frac{\partial o_{t+1}}{\partial V}$$

Tiếp theo, đạo hàm với U, W thì có:

$$\begin{aligned}
\frac{\partial L}{\partial U} &= \frac{\partial L}{\partial o_{t+1}} * \frac{\partial o_{t+1}}{\partial s_{t+1}} * \frac{\partial s_{t+1}}{\partial U} \\
&= \frac{\partial L}{\partial o_{t+1}} * \frac{\partial o_{t+1}}{\partial s_{t+1}} * \frac{\partial s_{t+1}}{\partial s_t} * ... * \frac{\partial s_1}{\partial U} \\
\frac{\partial L}{\partial W} &= \frac{\partial L}{\partial o_{t+1}} * \frac{\partial o_{t+1}}{\partial s_{t+1}} * \frac{\partial s_{t+1}}{\partial U} \\
&= \frac{\partial L}{\partial o_{t+1}} * \frac{\partial o_{t+1}}{\partial s_{t+1}} * \frac{\partial s_{t+1}}{\partial s_t} * ... * \frac{\partial s_1}{\partial W}
\end{aligned}$$

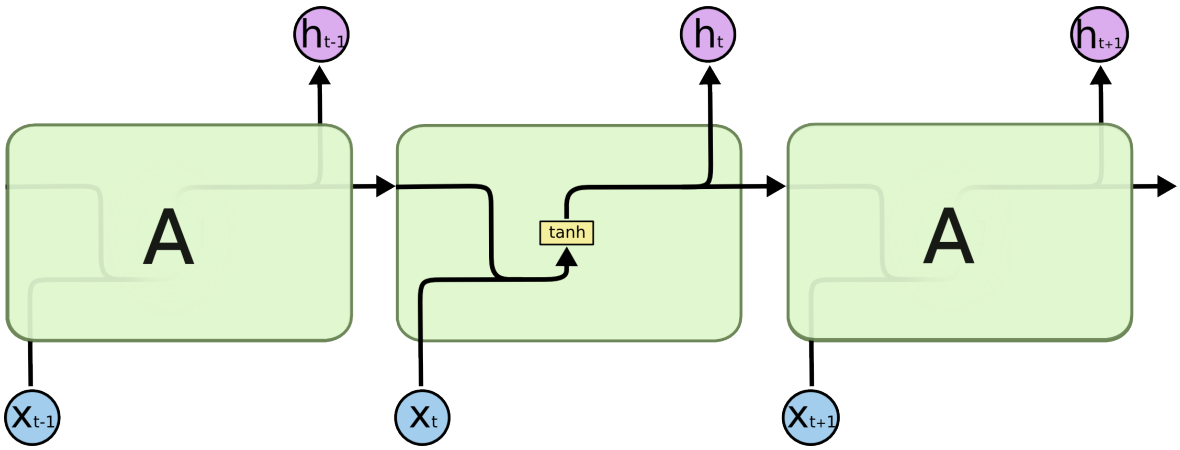
Từ công thức trên, Đầu ra của bài toán chúng ta thường xét dạng xác suất nên giá trị đầu ra từ 0 đến 1. Từ đó, ta thấy có xảy ra hiện tượng vanishing gradient (tức là nhiều đạo hàm liên tiếp nhỏ hơn 1 nhân với nhau - nếu dãy nhân nhau dài sẽ dẫn đến hội tụ về 0 ==> từ đó gây ra hiện tượng "không nhớ"). Còn nếu ngược lại gọi là exploding gradient (tích của rất nhiều số lớn hơn 1 sẽ dẫn đến vô cùng làm cho bước cập nhật hệ số gradient không còn chính xác). Điều này sẽ được cải tiến trong mô hình LSTM sau này.

Chương 2

Mô hình BiLSTM

2.1 Mô hình LSTM

Ở chương trước, chúng ta đã tìm hiểu về mạng RNN. Các mạng hồi quy đều có dạng là một chuỗi các module lặp đi lặp lại của mạng neural. Với mạng neural, các module này có cấu trúc rất đơn giản, thường là một tầng tanh:

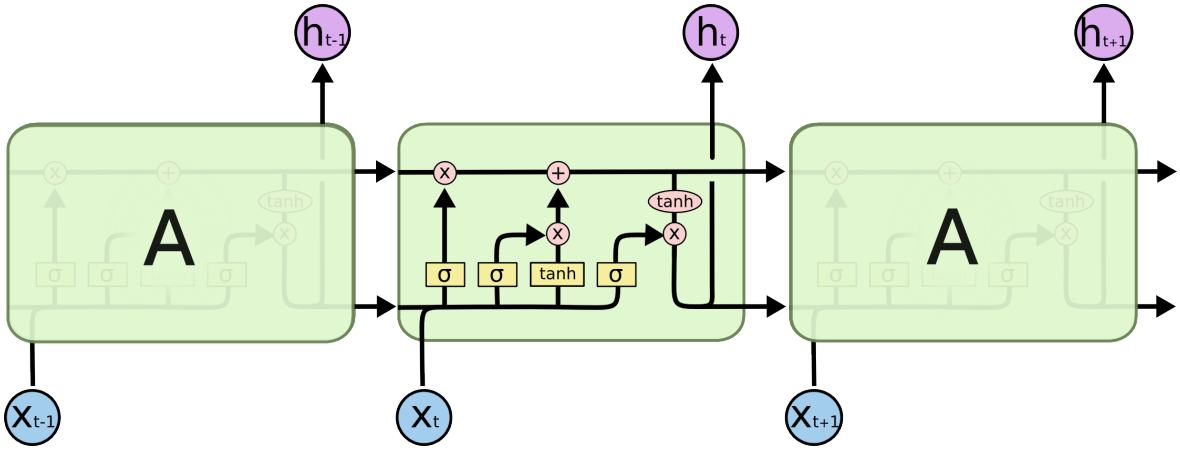


Hình 2.1: Mô hình mạng RNN dạng chuỗi

Tuy nhiên, RNN lại có thể bị vanishing gradient (mất mát đạo hàm), khiến cho mô hình chỉ nhớ được các chuỗi ngắn mà không nhớ được các chuỗi dài. Và LSTM (Long-short term memory) là một biến thể của RNN nhằm mục đích khắc phục nhược điểm này.

LSTM cũng có kiến trúc dạng chuỗi như RNN, nhưng các module có cấu trúc khác với mạng RNN chuẩn. Thay vì chỉ có một tầng mạng neural, chúng có tới 4 tầng tương tác với nhau một cách rất đặc biệt.

LTSM có các cổng giúp lọc thông tin. Trạng thái của cell có thể coi là trí nhớ của toàn hệ thống. Muốn được cập nhất vào hệ thống này, các thành phần thông tin phải đi qua các cổng. Vậy có thể thấy cơ chế này linh hoạt hơn khi mà các thành phần thông tin được chọn lọc mới có thể được đưa vào bộ nhớ, nhờ đó các thành phần có ích từ

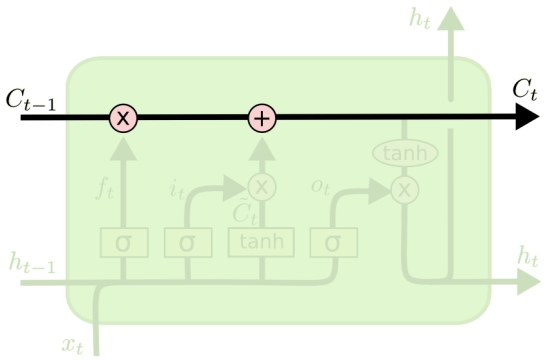


Hình 2.2: Mô hình mạng LSTM

thời điểm rất lâu trong quá khứ vẫn có thể có tác dụng điều chỉnh.
Trong LSTM sử dùng hàm sigmoid, hàm này có tác dụng như hàm tanh, nhưng vùng chiếu của nó chỉ là $(0, 1)$ thay vì $(-1, 1)$. Nếu phần thông tin nào không quan trọng, ta chiếu nó về 0, nó coi như bị "lãng quên".

2.1.1 Ý tưởng chính của mạng LSTM

Chìa khóa của LSTM là trạng thái tế bào (cell state) - chính đường chạy thông ngang phía trên của sơ đồ hình vẽ.

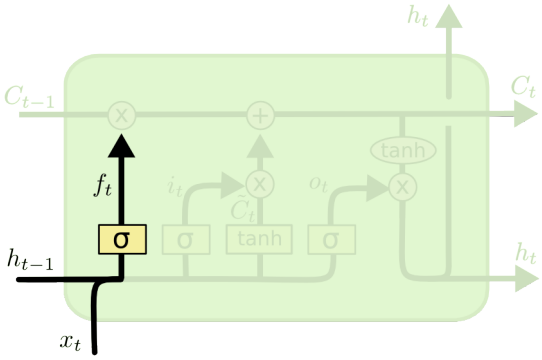


Hình 2.3: Đường trạng thái tế bào của LSTM

Trạng thái tế bào là một dạng giống như băng truyền. Nó chạy xuyên suốt tất cả các mắt xích (các nút mạng) và chỉ tương tác tuyến tính đôi chút. Vì vậy mà các thông tin có thể dễ dàng truyền đi thông suốt mà không sợ bị thay đổi. Các cổng là nơi sàng lọc thông tin đi qua nó. LSTM có 3 cổng là: cổng quên, cổng vào và cổng ra.

2.1.2 Cổng quên

Cổng này quyết định xem thông tin nào trong bộ nhớ hiện tại được giữ và thông tin nào. Đầu vào là h_{t-1} và x_t rồi đưa qua hàm sigmoid sẽ được giá trị f_t trong khoảng [0, 1]. Đầu ra 1 thể hiện việc giữ toàn bộ thông tin, đầu ra 0 thể hiện toàn bộ thông tin sẽ bị bỏ đi.

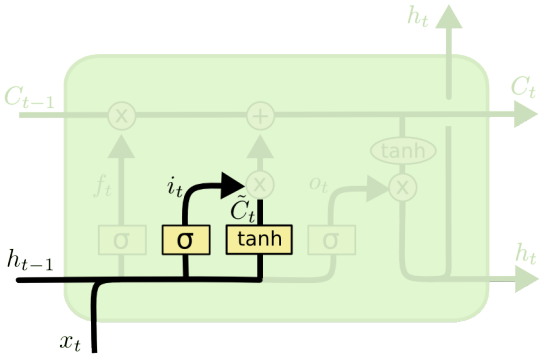


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Hình 2.4: Cổng quên của mạng LSTM

2.1.3 Cổng vào

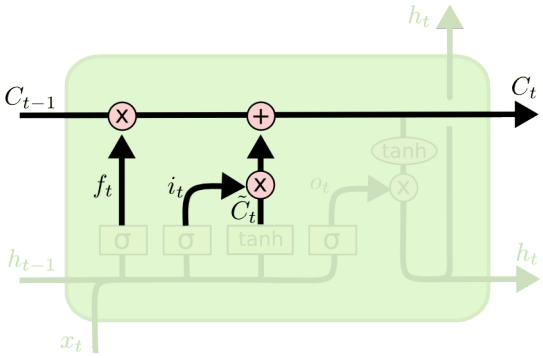
Cổng này dùng để cập nhật thông tin mới. Ở đây có xuất hiện 2 hàm sigmoid và hàm tanh. Tác dụng của chúng cũng như trên. Output từ hàm sigmoid sẽ có tác dụng lọc thông tin đã qua xử lý từ output hàm tanh.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Hình 2.5: Cổng vào của mạng LSTM

Giờ là lúc cập nhật trạng thái tế bào cũ C_{t-1} thành trạng thái mới C_t. Ta sẽ nhân trạng thái cũ với f_t để bỏ đi những thông tin ta quyết định quên lúc trước. Sau đó cộng thêm $i_t * \tilde{C}_t$. Trạng thái mới thu được này phụ thuộc vào việc ta quyết định cập nhập mỗi giá trị trạng thái ra sao.

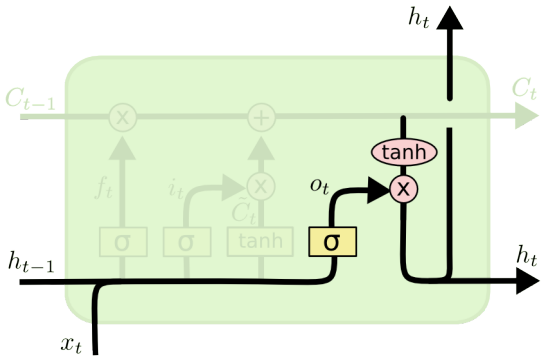


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Hình 2.6: Cập nhật trạng thái tế bào từ cổng vào của mạng LSTM

2.1.4 Cổng ra

Giá trị đầu ra sẽ dựa vào trạng thái tế bào, nhưng sẽ được tiếp tục sàng lọc. Đầu tiên, ta chạy một tầng sigmoid để quyết định phần nào của trạng thái tế bào ta muốn xuất ra. Sau đó, ta đưa nó trạng thái tế bảo qua một hàm tanh để co giá trị nó về khoảng $[-1, 1]$, và nhân nó với đầu ra của cổng sigmoid để được giá trị đầu ra ta mong muốn. Chú ý rằng cả kết quả đầu ra và cả trạng thái cell đều được đưa vào bước tiếp theo.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Hình 2.7: Cổng ra của mạng LSTM

2.2 Mô hình BiLSTM

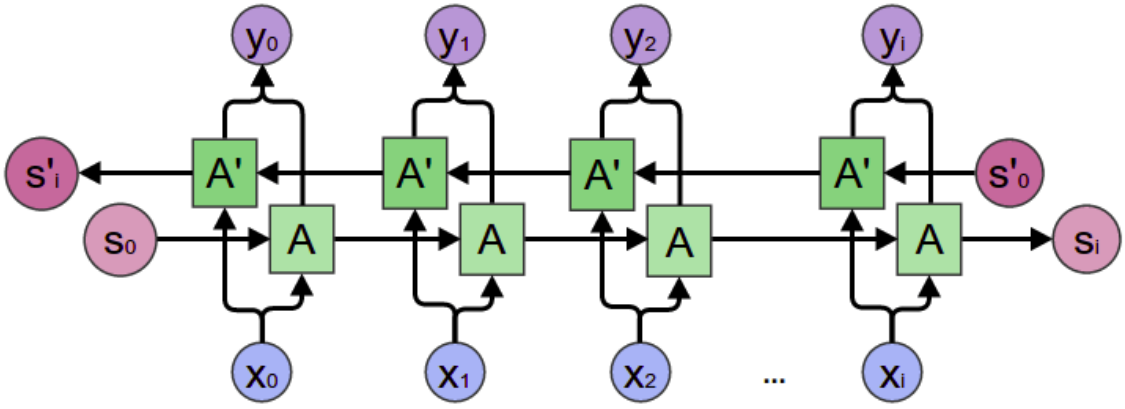
LSTM đơn hướng chỉ lưu giữ thông tin của **quá khứ** vì các đầu vào duy nhất mà nó đã thấy là từ quá khứ. Đặc điểm chung của Vanilla RNN (RNN cơ bản) và LSTM là chúng đều hoạt động theo một chiều nhất định. Hay nói cách khác, các mạng này chỉ mang thông tin tính tới thời điểm hiện tại. Tuy nhiên, trong nhiều bài toán NLP thì việc biết thông tin của các timesteps tiếp theo giúp cải thiện rất nhiều kết quả.
Ví dụ, trong bài toán điền từ còn thiếu vào câu, nếu chúng ta biết 1 câu như sau:

Anh ấy là người ...

Rõ ràng, nếu chỉ biết các từ trước vị trí cần điền thì thật là khó để dự đoán từ tiếp theo trong câu trên, nhưng nếu biết các từ đằng sau đó nữa như:

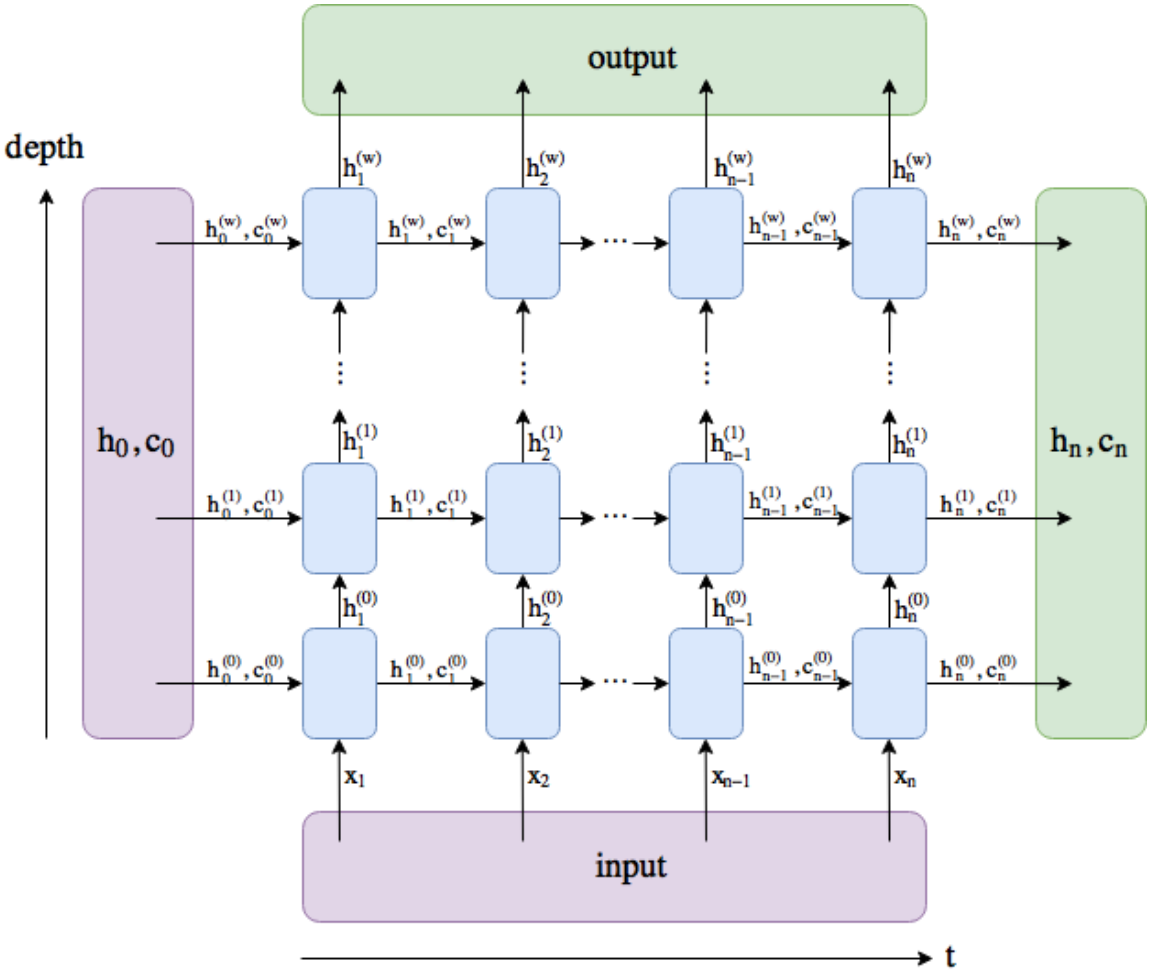
nên anh ấy nói được Tiếng Việt

Nếu biết cả các từ đằng sau vị trí cần điền này thì dễ dàng đoán được từ cần điền là từ *Việt Nam*.
Từ ví dụ trên, ta có thể thấy rằng thông tin các từ ở tương lai cũng có 1 tầm quan trọng nhất định để giúp các mô hình NLP hoạt động tốt hơn. Vì thế, việc sử dụng hai chiều sẽ chạy đầu vào của chuỗi theo hai cách, một từ quá khứ đến tương lai và một từ tương lai đến quá khứ và điều khác biệt trong cách tiếp cận này là có một hướng là LSTM chạy ngược lưu trữ thông tin tương lai, từ đó chúng ta có thể lưu giữ thông tin từ cả từ quá khứ và tương lai. Và đó cũng chính là tư tưởng của bidirectional LSTM hay gọi tắt là BiLSTM.Hoạt động của các bidirectional RNN nói chung có thể được mô tả như hình sau:



Hình 2.8: Hoạt động của Bidirectional RNN

Chúng ta có thể sử dụng Vanilla RNN, LSTM hoặc GRU... làm các cell cho bidirectional và kết quả đầu ra được tính bằng cách kết hợp cả hai hidden state từ hướng thuận và hướng nghịch.
Nhiều kết quả thực nghiệm đã cho thấy BiLSTM thực sự hoạt động tốt hơn LSTM rất nhiều. Ngoài ra, chúng ta có thể kết hợp nhiều mạng RNN để tạo thành một mạng Multi-layer RNN với đầu ra của layer này trở thành đầu vào của layer tiếp theo:

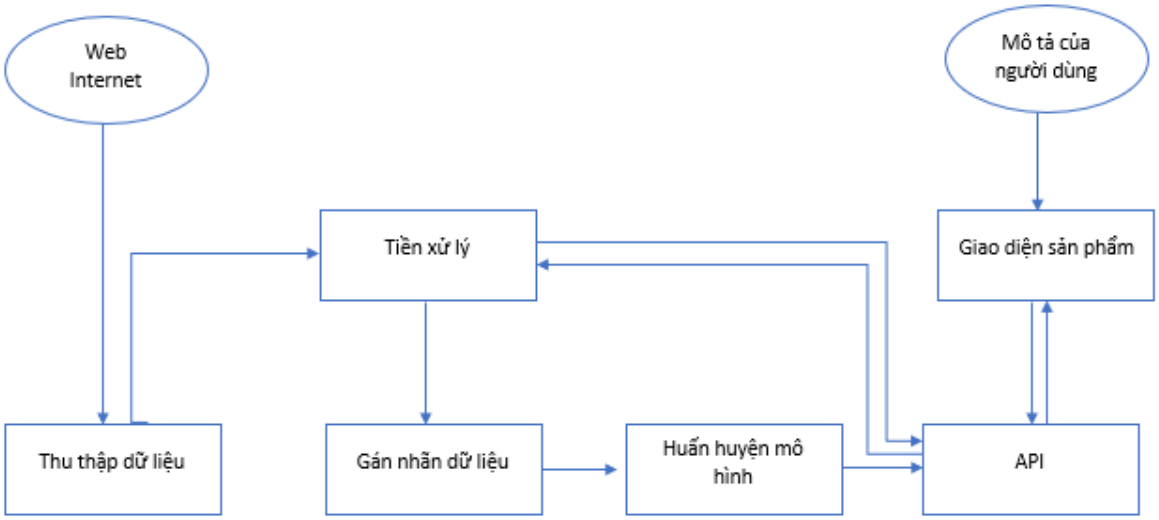


Hình 2.9: Hoạt động của Multi-layer RNN

Chương 3

Thực nghiệm mô hình cho bài toán gán nhãn từ loại

3.1 Quy trình xử lý



Hình 3.1: Quy trình xử lý bài toán

Mô tả quy trình:
Dữ liệu từ internet sẽ được crawl bằng pha thu thập dữ liệu, sau đó tiến hành tiền xử lý để làm đẹp dữ liệu, trong đó bao gồm cả tokenizer câu thành các từ và cụm từ, rồi được gán nhãn và đưa vào mô hình để huấn luyện. Kết quả của mô hình sẽ là lõi của API dùng để dự đoán bài toán.
Khi người dùng nhập vào một câu mô tả gửi đến giao diện sản phẩm, từ giao diện sẽ gửi yêu cầu lên server để API dự đoán câu này, API sẽ đưa câu mô tả đó đến pha tiền xử lý để xử lý dữ liệu từ người dùng và tokenizer câu đó, sau đó dự đoán trên dữ liệu đã xử lý này và trả lại kết quả cho giao diện để hiện lên cho người dùng.

Dựa trên quy trình xử lý, tôi đề xuất những công cụ sau để thực hiện xây dựng mô hình:

Công cụ	**Chú thích**	**Pha sử dụng**
Scrapy	Thư viện hỗ trợ crawl dữ liệu của Python	Thu thập dữ liệu
Spash	Kết hợp với Scrapy để crawl web chạy bằng JS	Thu thập dữ liệu
Pyvi	Công cụ tokenizer cho tiếng Việt	Tiền xử lý
Tensorflow	Framework Tensorflow machine learning	Huấn luyện mô hình và API
Flask	Thư viện của Python	API
Jquery	Thư viện của Javascript	Giao diện sản phẩm

Với giao diện sản phẩm web, tôi thiết lập giao diện bằng HTML, CSS, Bootstrap, giao diện tương tác người dùng và server bằng Javascript và thư viện Jquery của nó. Các pha còn lại đều sử dụng Python hoặc thư viện, framework của Python để xử lý.

3.2 Pha thu thập dữ liệu

Mục đích của pha này là thu thập đủ dữ liệu cho mô hình, dữ liệu có thể là trên internet hoặc các dữ liệu công khai của các tổ chức khác nhau. Dữ liệu phải đảm bảo tính đa dạng và tổng quát để mô hinh hoạt động hiệu quả. Với pha này,vì không có nhiều thời gian và nhân lực để thực hiện nên tôi sử dụng bộ dữ liệu VLSP 2016 - Một bộ dữ liệu tiếng việt cho các bài toán NER, POS đã được gán nhãn sẵn. Ngoài ra, để tăng cường đa dạng hơn dữ liệu, thì có thể tiến hành crawl các mô tả, tiêu đề báo trên các trang mạng để gán nhãn và làm đa dạng dữ liệu. Crawl sử dụng thư viện Scrapy của Python. Tuy nhiên, trong quá trình crawl, tôi nhận ra rằng có một số trang web sử dụng Javascript để đổ dữ liệu khi load web, vì thế khiến thư viện Scapy không hoạt động được với những trang web này. Giải pháp đề xuất là sử dụng thư viện Spash của Python kết hợp với Scapy chạy trên nền Docker để crawl dữ liệu những trang web đặc biệt này.

3.3 Pha tiền xử lý

Dữ liệu được thu thập từ các trang web trên mạng cơ bản là đáp ứng nhu cầu để huấn luyện. Tuy nhiên, dữ liệu này vẫn chứa nhiều rác, đôi khi là bị lặp. Việc xử lý dữ liệu là chọn lọc ra các thuộc tính làm đẹp cho mô hình, loại bỏ các yếu tố gây nhiễu mô hình, làm sạch dữ liệu thô thu thập được từ pha thu thập dữ liệu.

Việc làm sạch dữ liệu bao gồm các thao tác sau:

- Loại bỏ ký tự đặc biệt không phải chữ hoặc số.
- Xoá bỏ ký tự phân tách câu không phải dấu chấm, dấu phẩy hoặc dấu chấm than.
- Loại bỏ các câu giống nhau, bị lặp.
- Xoá bỏ các dòng trắng, khoảng trắng vô nghĩa.

Sau khi dữ liệu được làm sạch xong, sẽ được tiến hành tokenizer để chuẩn bị cho quá trình huấn luyện. Ở bước này, tôi đề xuất công cụ Pyvi của tác giả TS. Trần Việt Trung - Đại học Bách Khoa Hà Nội. Công cụ Pyvi là một tool hiệu quả để tokenizer tiếng Việt với độ chính xác cao. Riêng với bộ dữ liệu của VLSP 2016, do vì đã được đội ngũ VLSP xây dựng và tiền xử lý rất tốt, nên dữ liệu này không cần đi qua pha này.

3.4 Pha gán nhãn

Mục tiêu của chúng ta là cần gán nhãn dữ liệu để cho mô hình học hiệu quả. Dữ liệu của VLSP 2016 đã có sẵn nhãn, tuy nhiên để tương thích với tool pyvi của tác giả TS. Trần Việt Trung, thì trong quá trình đọc các từ, ta cần thêm dấu _ thay vì khoảng trắng để phân tách giữa các cụm từ. Bộ dữ liệu của VLSP 2016 có cấu trúc như sau:

Quân thù	N	B-NP	O
đang	R	O	O
còn	V	B-VP	O
đó	P	B-NP	O
,	CH	O	O
bao nhiêu	P	B-NP	O
bà	Nc	B-NP	O
mẹ	N	B-NP	O
còn	R	O	O
mất	V	B-VP	O
con	N	B-NP	O
,	CH	O	O
bao nhiêu	P	B-NP	O
người	N	B-NP	O
chồng	N	B-NP	O
mất	V	B-VP	O
vợ	N	B-NP	O
.	CH	O	O

Có thể thấy, có những câu kép như "quân thù", "bao nhiêu",... Với những câu này, ta thay khoảng trắng bằng dấu _ để tương thích với tool tokenizer pyvi, vì tool tách 1 câu ra thành các từ, cụm từ, trong đó các cụm từ ngăn cách bởi dấu _ . Kết quả các cụm từ sau khi thay sẽ thành "quân_thù", "bao_nhiêu"...
Với bộ dữ liệu tự crawl, chúng ta tiến hành xử lý và gán nhãn giống với bộ dữ liệu của VLSP để có thể kết hợp chúng lại với nhau. Chúng ta sử dụng cột thứ 3 của dữ liệu cho bài toán gán nhãn từ loại tiếng Việt. Trong đó ý nghĩa các thẻ như sau:

- **AP**: (Adjective pharse) - Cụm tính từ.
- **NP**: (Noun pharse) - Cụm danh từ.
- **PP**: (Prepositional pharse) - Cụm giới từ.
- **VP**: (Verb pharse) - Cụm động từ.
- **O**: Không xác định.
- **B-**: Thẻ bắt đầu nhãn.
- **I-**: Thực thể nằm trong nhãn nào đó.

Ví dụ như câu: *Anh Thắng là cán bộ Uỷ ban nhân dân thành phố Hà Nội* thì sẽ được token thành: *Anh, Thắng, là, cán_bộ, Uỷ_ban, nhân_dân, thành_phố, Hà_Nội*. Sau đó, tương ứng với câu này là dãy thẻ: *B-NP, I-NP, B-VP, B-NP, B-NP, I-NP, I-NP, I-NP*.

3.5 Pha huấn luyện mô hình

Pha đóng vai trò quan trọng nhất trong mô hình đó là pha huấn luyện dữ liệu. Sau bước thu thập dữ liệu, pha tiền xử lý mang lại tác dụng làm cho kết quả mô hình tốt hơn, thì pha huấn luyện này là pha không thể thiếu. Dữ liệu được làm sạch, sẽ được dùng là đầu vào cho huấn luyện mô hình. Ứng dụng mô hình BiLSTM, dữ liệu thu thập được sẽ được huấn luyện để tạo ra mô hình đề xuất.

Chúng ta tạo các dictionary là tập từ vựng, tập các tag, đánh ID cho mỗi từ là 1 số từ tập từ vựng của tập train, và các dictionary hỗ trợ cho việc chuyển đổi từ sang số, tag sang số và ngược lại. Ngoài ra, bổ sung các từ *UNK* - từ không xác định (không có trong bộ từ vựng), và *PAD* - từ viền của câu.

```
def parse(sentences):
    all_words = []
    all_tags = []
    for sentence in sentences:
        for word, tag in sentence:
            if word not in all_words:
                all_words.append(word)
            if tag not in all_tags:
                all_tags.append(tag)
    all_words.sort()
    all_tags.sort()

    word2idx = {word: idx + 2 for idx, word in enumerate(all_words)}
    word2idx['PAD'] = 0
    word2idx['UNK'] = 1
    tag2idx = {tag: idx + 1 for idx, tag in enumerate(all_tags)}
    tag2idx['PAD'] = 0

    idx2word = {idx: word for word, idx in word2idx.items()}
    idx2tag = {idx: tag for tag, idx in tag2idx.items()}

    return all_words, all_tags, word2idx, idx2word, tag2idx, idx2tag
```

Hình 3.2: Tạo các dictionary cần thiết

Trước khi đưa dữ liệu vào train, ta cần chuẩn hoá sao cho độ dài tất cả các câu đưa vào có kích thước bằng nhau (thường chọn dưới 120 - vì trên 120 thì mô hình BiLSTM cũng không còn hoạt động hiệu quả). Nếu câu có độ dài ngắn hơn 120, sẽ được thêm các từ *PAD* vào cuối cùng, ngược lại, nếu câu có độ dài vượt quá 120, sẽ bị cắt bớt đi ở cuối. Trong module *tensorflow.keras.preprocessing.sequence* có hỗ trợ hàm *pad_sequence* thực hiện điều này. Sau khi đã thiết lập đủ dữ liệu, lập hàm next_batch

```
def sentence_to_number(sentences, max_length, word2idx, tag2idx):
    x = [[word2idx[word] for word, tag in sent] for sent in sentences]
    y = [[tag2idx[tag] for word, tag in sent] for sent in sentences]
    x_pad = pad_sequences(x, maxlen=max_length, dtype='int32', padding='post', truncating='post', value=wor
d2idx['PAD'])
    y_pad = pad_sequences(y, maxlen=max_length, dtype='int32', padding='post', truncating='post', value=tag
2idx['PAD'])
    y_categorical = [to_categorical(idx, num_classes=len(list(tag2idx.keys())), dtype='int32') for idx in y
_pad]
    return np.array(x_pad), np.array(y_categorical)
```

Hình 3.3: Pad các câu để đưa vào train

để lấy ra từng batch cho mô hình đưa vào train, tránh tình trạng đưa tất cả dữ liệu vào train làm cho tràn bộ nhớ.

Mô hình thiết lập sẽ gồm các lớp như sau:

Layer	Output shape
Embedding	(None, 120, 100)
BiLSTM 1	(None, 120, 128)
Dropout 1	(None, 120, 128)
BiLSTM 2	(None, 120, 256)
Dropout 2	(None, 120, 256)
Dense	(None, 120, 9)

Ở layer đầu tiên, ta thêm 1 lớp embedding, đầu ra là 1 vector 120 phần tử (độ dài câu) và 100 đặc trưng. Word Embedding là một kỹ thuật cho việc học mật độ dày đặc thông tin đại diện của từ trong một không gian vector có số chiều nhỏ hơn. Mỗi một từ có thể được xem như là một điểm trong không gian này, được đại diện bởi một vector có độ dài cố định. Word Embedding thường được thực hiện trong lớp đầu tiên của mạng: Trong đó lớp embedding sẽ ánh xạ một từ (chỉ số index của từ trong từ điển từ vựng) từ từ điển sang một vector dày đặc với kích thước đã cho.

```
    def _embedding(self, X):
        with tf.variable_scope('embeding'):
            embed_word = tf.contrib.layers.embed_sequence(X, self.VOCAB_SIZE, self.EMBEDDING_SIZE,
                                                          initializer=tf.random_uniform_initializer(seed=2
8))
        return embed_word
```

Hình 3.4: Hàm embedding trong class Model

Các lớp tiếp theo, ta sử dụng 2 BILSTM chồng nhau, trong đó có dùng dropout để giảm overfitting cho mô hình. Lớp BiLSTM thứ nhất, mỗi LSTM gồm 64 unit (tổng 2 chiều thành 128 unit), lớp BiLSTM thứ 2, mỗi LSTM gồm 128 unit (tổng 2 chiều thành 256 unit).

```
    def _bilstm(self, layer_pre, num_unit):
        with tf.variable_scope('BiLSTM' + str(num_unit)):
            lstm_fw_cell = tf.contrib.rnn.LSTMCell(num_unit)
            lstm_bw_cell = tf.contrib.rnn.LSTMCell(num_unit)
            (output_fw, output_bw), _ = tf.nn.bidirectional_dynamic_rnn(lstm_fw_cell, lstm_bw_cell, layer_p
re,
                                                                        dtype=tf.float32)
        output = tf.concat([output_fw, output_bw], axis=-1)
        return output
```

Hình 3.5: Hàm BiLSTM trong class Model

```
def _dropout(self, layer_pre):
    if self.DROPOUT_RATE > 0.0:
        dropout = tf.nn.dropout(layer_pre, 1.0 - self.DROPOUT_RATE)
    else:
        dropout = layer_pre
    return dropout
```

Hình 3.6: Hàm Dropout trong class Model

Tiếp theo là layer đầu ra, là 1 lớp dense 9 unit tương ứng với số thẻ tag chúng ta cần gán: B-AP, B-NP, B-PP, B-VP, I-AP, I-NP, I-VP, O, PAD. (Thẻ PAD là đại diện cho các từ PAD viền của câu).

```
def _logits(self, layer_pre):
    with tf.variable_scope('logits'):
        logits = tf.layers.dense(layer_pre, self.NUM_CLASSES)
    return logits
```

Hình 3.7: Hàm xây dựng lớp Dense trong class Model

Sau khi thiết lập xong các layer cho mô hình, tiến hành định nghĩa hàm loss (chọn hàm softmax cross entropy with logits), metric đánh giá accuracy (chọn tổng số dự đoán đúng/tổng số dự đoán), và phương pháp tối ưu (chọn Adam). Sau đó tiến hành train mô hình theo các step, mỗi step đưa vào 1 batch dữ liệu. Và lưu mô hình và các trọng số kết quả lại để sử dụng cho API dự đoán ở pha tiếp theo. Kết quả train cho thấy mô hình đạt độ chính xác cao trên tập test ($> 99\%$) (trên độ đo đã chọn), đó là một kết quả tốt cho bài toán này.

3.6 Thiết lập API

API đóng vai trò chạy ở phía server, khi người dùng gửi 1 câu lên web, web gửi dữ liệu về server và server gửi trả kết quả lại cho người dùng. Đó là API.Ở trong báo cáo này, tôi chọn Flask làm back-end cho sản phẩm. Trong API, ta cần thiết lập hàm restone lại mô hình đã được train, khi khởi chạy server, hàm này được chạy duy nhất 1 lần và truy trì trong suốt quá trình server chạy.

```
def get_model_api():
    model = Model(VOCAB_SIZE, NUM_CLASSES)
    model.build_model()
    model.saver.restore(model.sess, tf.train.latest_checkpoint(LOAD_CHECKPOINT))

    def model_api(input_data):
        sent_token, tags = model.predict_batch(input_data, all_words, word2idx, idx2tag)
        output_data = {'tokens': sent_token, 'tags': tags}
        return output_data

    return model_api


model_api = get_model_api()
```

Hình 3.8: Hàm respone lại model và thực hiện dự đoán trong API

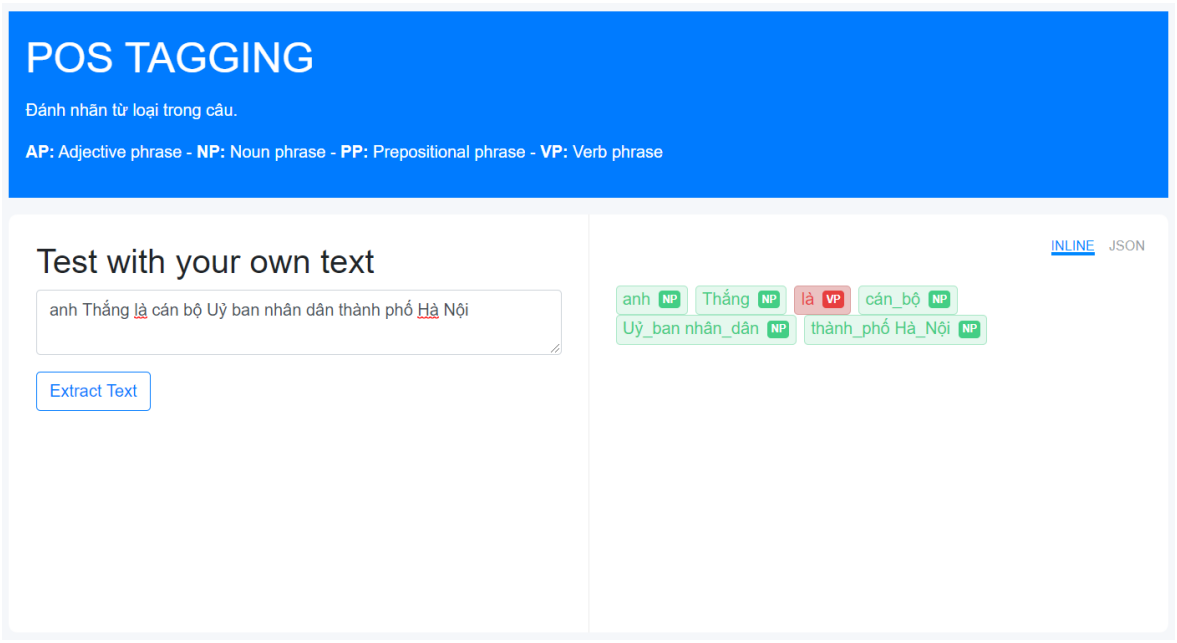
Tiếp đó, thiết lập router **/api** để nhận vào là 1 json dạng **{"text": Câu mô tả}** , và dự đoán và trả về 1 json kết quả.

```
@app.route('/api', methods=['POST', 'OPTIONS'])
def predict():
    api_input = request.json
    if api_input is not None:
        sentence = [str(api_input["text"])]
        response = model_api(sentence)
        return jsonify(response)
    else:
        return "Don't send None data to server"
```

Hình 3.9: Thiết lập router cho API

3.7 Giao diện sản phẩm

Giao diện sản phẩm là nơi tương tác trực tiếp với người sử dụng, đóng vai trò trung gian giữa người dùng và server. Giao diện sản phẩm như sau:



Hình 3.10: Giao diện sản phẩm

Hình 3.11: Giao diện sản phẩm

KẾT LUẬN

Có thể thấy rằng, chỉ với mô hình BiLSTM và dùng framework Tensorflow cùng các công cụ hỗ trợ khác, chúng ta đã có một mô hình hoạt động tương đối tốt cho bài toán gán nhãn từ loại tiếng Việt. Hiện nay, có nhiều nghiên cứu để cải tiến hiệu năng mô hình cũng như có những thuật toán mới cải tiến hơn, hiệu quả hơn. Vì thời gian tìm hiểu chưa nhiều, trong phạm vi của báo cáo môn học, tôi xin dừng báo cáo của mình tại đây.

Dễ thấy rằng, tuy đã hoạt động tốt, nhưng vẫn còn đó những bất cập. Ví dụ như thẻ lồng thẻ (trong thực thể này lại chứa thực thể kia) là chúng ta chưa xử lý được... Và đây cũng là một vấn đề khó trong nghiên cứu các bài toán NER, POS hiện nay. Và đó cũng chính là hướng nghiên cứu tiếp theo để cải tiến cho bài toán này.

Rất mong nhận được sự góp ý của các thầy cô và các bạn để báo này được hoàn thiện hơn.

Tài liệu tham khảo

[1] Hobson Lane, Hannes Hapke, Cole Howard, *Natural Language Processing in Action*

[2] Vũ Hữu Tiệp, *Machine learning cơ bản*, Nhà xuất bản khoa học và kỹ thuật, 2018

[3] Diedelrik P.Kingma, Jimmy Lei Ba, *Adam: A method for stochastic optimization*, ICLR 2015

[4] Báo cáo hội nghị VLSP 2016

[5] Tom Hope, Yehezkel S. Resheff & Itay Lieder, *Learning Tensorflow*