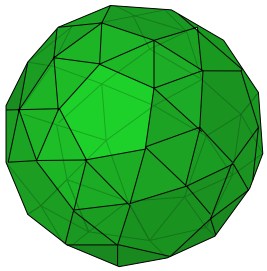
Projects in Mathematics and Applications

SENTIMENT ANALYSIS

Nguyễn Thị Quỳnh Trang [*] Huỳnh Vũ Khôi Nguyên [†]
Võ Minh Quân [‡] Hồ Tiến Đức [§]

Ngày 29 tháng 8 năm 2019



[*]THPT chuyên KHTN - ĐHQG Hà Nội
[†]THPT chuyên Trần Hưng Đạo - Bình Thuận
[‡]THPT chuyên Thăng Long - Đà Lạt
[§]Phổ thông Năng khiếu - ĐHQG TPHCM

Lời cảm ơn

Để hoàn thành bản báo cáo này, nhóm 2 đã nhận được rất nhiều sự giúp đỡ của các thành viên trong trại hè.

Lời đầu tiên chúng em bày tỏ lòng biết ơn sâu sắc đến quan tâm của mọi người trong thời gian qua:

Chúng em xin gửi lời cảm ơn chân thành nhất đến các co-founders anh Cấn Trần Thành Trung, anh Trần Hoàng Bảo Linh, anh Lê Việt Hải cùng ban tổ chức PiMA 2019 đã mang đến trải nghiệm thú vị và bổ ích cho chúng em trong mùa hè này. Không chỉ được tìm hiểu sâu về Toán ứng dụng nói chung và Deep Learning nói riêng mà chúng em còn được rèn luyện những kĩ năng làm việc nhóm, lập trình, nghiên cứu và viết báo cáo. Đó đều là những kinh nghiệm quý báu cho chúng em trên chặng đường tương lai.

Hi vọng trong những năm tiếp theo, PiMA có thể tiếp tục phát triển để lan toả niềm đam mê Toán học cho các bạn học sinh THPT trên khắp đất nước.

Chúng em xin cảm ơn các anh chị trong ban mentors và các diễn giả, đặc biệt là sự giúp đỡ tận tình của anh Lâm Hữu Phúc, anh Trần Thanh Bình và chị Vương Nguyễn Thùy Dương trong quá trình tìm hiểu và chuẩn bị bài báo cáo này. Bên cạnh đó, chúng em cũng muốn bày tỏ lòng biết ơn đến trường Đại Học Khoa Học Tự Nhiên TP.HCM và các nhà tài trợ đã tạo điều kiện để trại hè PiMA 2019 được diễn ra thành công.

Cuối cùng xin cảm ơn các bạn trại sinh khác đã đồng hành cùng chúng mình trong 12 ngày trại vừa qua.

Do thời gian tìm hiểu ngắn, kiến thức còn có hạn nên thiếu sót là điều không thể tránh khỏi. Nhóm chúng em rất mong nhận được đóng góp từ phía các anh chị mentors và bạn đọc để dự án có thể hoàn thiện hơn.

TÓM TẮT NỘI DUNG

Nội dung bản báo cáo này xoay quanh mạng Nơ ron Hồi quy (RNN), các tính chất, các vấn đề liên quan đến RNN như Gradient Vanishing, Gradient Exploding, cách khắc phục các vấn đề trên thông qua cấu trúc LSTM, và ví dụ cài đặt LSTM và so sánh với mô hình RNN cơ bản trong việc giải quyết bài toán Sentiment Analysis .

Mục lục

1 ĐẶT VẤN ĐỀ

Dữ liệu văn bản là một trong những hình thức phổ biến nhất của dữ liệu, được sử dụng trong nhiều bài toán máy học phổ biến:

1. Phân loại văn bản (text classification): Dự đoán xem người sử dụng có hài lòng với sản phẩm hay không dựa vào reviews; phát hiện email spam, phân biệt sắc thái văn bản
2. Dịch tự động (machine translation)
3. Trả lời câu hỏi tự động (question answering) và chatbot
4. Tóm tắt văn bản (text summarization)

Một trong những tính chất đặc thù của dữ liệu dạng văn bản đó chính là tính chất phụ thuộc của các từ vào ngữ cảnh, cũng như nội dung các từ đã từng xuất hiện. Giả sử chúng ta có câu ”Tôi đi ăn”, từ tiếp theo khả năng sẽ là ”cơm” hoặc ”mì”, mà không phải là ”ghế nhựa”. Tính chất này đưa đến cho chúng ta ý tưởng xây dựng một mô hình cho dữ liệu văn bản:

$$h_t = f(x_t, h_{t-1})$$

với h_t được gọi là vector "trạng thái ẩn" (hidden state) tại vị trí t, được tính phụ thuộc vào x_t là từ ở vị trí t và trạng thái ẩn h_{t-1} của từ trước $t-1$. Vector trạng thái ẩn h_{t-1} này có vai trò lưu lại thông tin của các từ trước nó, giúp chúng ta có thể mô hình hoá mối quan hệ giữa mỗi từ và các từ trước trong văn bản. Chúng ta có thể dùng những trạng thái ẩn này làm feature cho các bài toán máy học đã nêu ở trên.
Mô hình này được gọi là là một **mạng nơ-ron hồi quy (Recurrent Neural Network)** truyền thống.

Trong thực tế, mạng nơ ron này thường gặp vấn đề trong việc mô hình những câu có độ dài tương đối lớn. Vấn đề này được giải quyết phần nào bởi mạng nơ ron Long Short-Term Memory của Hochreiter và Schmidhuber. Ở dự án này, chúng ta sẽ tìm hiểu về LSTM và ứng dụng nó vào bài toán **Sentiment Analysis** (nhận diện cảm xúc văn bản).

2 Cơ sở toán học

Để đọc báo cáo này, người đọc cần có các kiến thức cơ bản về Đại số tuyến tính, Giải tích hàm nhiều biến.

Ở đây, chúng tôi xin trình bày một vài định nghĩa cơ bản của các công cụ được dùng trong báo cáo này.

Định nghĩa. Ma trận Jacobi

$$f : \mathbb{R}^n \to \mathbb{R}^m$$
$$(x_1, x_2, \cdots, x_n) \mapsto y_i(x_1, x_2, \cdots, x_n), i = 1, \cdots, m$$

Khi đó ma trận Jacobi của f được định nghĩa là ma trận sau:

$$J_f(x_1, x_2, ..., x_n) = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

Định lý. Quy tắc mắt xích cho đạo hàm riêng

Cho hàm số khả vi $f : \mathbb{R}^n \to \mathbb{R}$ và các hàm số khả vi $x_1, x_2, \cdots, x_n : \mathbb{R}^m \to \mathbb{R}$ đều là hàm số theo các biến $t_1, t_2, \cdots, t_m$, khi đó:

$$\frac{\partial f}{\partial t_i} = \sum_{j=1}^{n} \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial t_i}, \ i \in \{1, 2, \cdots, m\}$$

Định nghĩa. Matrix norm

p-norm ($p \in \mathbb{R}$) của một ma trận A được định nghĩa bởi

$$\|A\|_p = \max_{x : \|x\|_p = 1} \|Ax\|_p$$

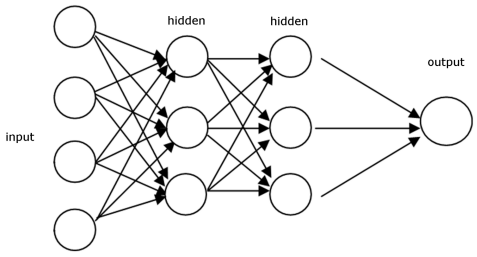
với $\|x\|_p$ là vector norm.

Trong bài viết này do chỉ sử dụng 2-norm nên để thuận tiện ta sẽ kí hiệu 2-norm của ma trận A bởi $\|A\|$.

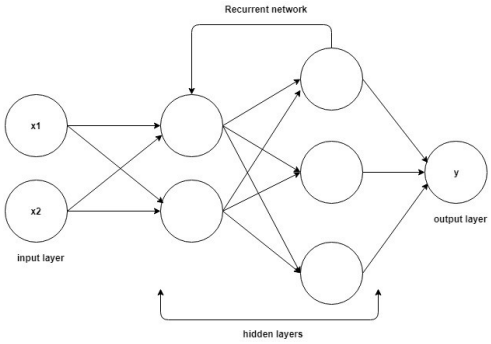
3 RECURRENT NEURAL NETWORK - RNN

3.1 Giới thiệu RNN

Mạng nơ ron nhân tạo (Artificial Neural Network - ANN) là một mô hình toán học hay mô hình tính toán được xây dựng dựa trên các mạng nơ-ron sinh học. Nó gồm có một nhóm các nơ-ron nhân tạo (nút) nối với nhau, và xử lý thông tin bằng cách truyền theo các kết nối và tính giá trị mới tại các nút. Kiến trúc chung của một mạng nơron nhân tạo (ANN) gồm 3 thành phần đó là: Input Layer, Hidden Layer và Output Layer.



Hình 1: MLP



Hình 2: RNN

Mạng nơ ron hồi quy (Recurrent Neural Network - RNN) là một lớp của ANN mà các liên kết giữa các nút tạo thành một đường truyền có hướng theo trình tự thời gian. Khác với mạng nơ ron nhiều lớp(Multi-layer Perceptron -MLP: thông tin được truyền thẳng từ lớp đầu vào qua các lớp ẩn đến lớp đầu ra), RNN có khả năng ghi nhớ thông tin ở các bước tính toán trước đó để dựa vào đó đưa ra dự đoán chính xác nhất cho bước dự đoán hiện tại.

Ứng dụng của bài toán RNN: RNN được ứng dụng và thành công ở nhiều bài toán:

- Speech to text: Chuyển giọng nói sang văn bản.
- Sentiment analysis : Nhận diện cảm xúc văn bản..
- Machine translation: Bài toán dịch tự động giữa các ngôn ngữ.
- Video recognition: Nhận diện hành động trong video.
- Heart attack: Dự đoán đột quỵ tim.

3.2 Mô hình chung của mang RNN và thuật toán lan truyền ngược theo thời gian (Backpropagation through time)

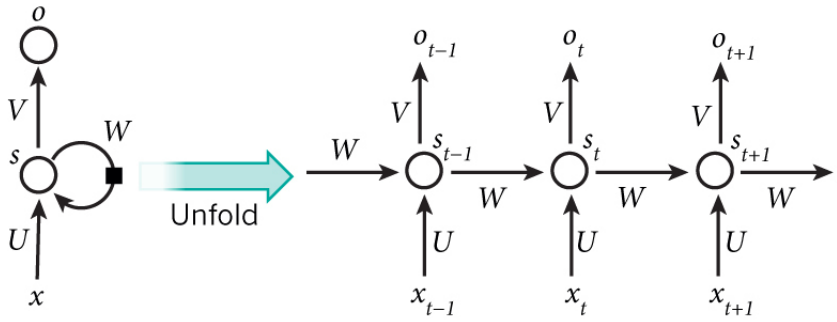
Mô hình chung của một mạng RNN thường được cho bởi công thức:

$$x_t = \sigma(\mathbf{W}_{rec}x_{t-1} + \mathbf{W}_{in}u_t + \mathbf{b}) \quad (*)$$

Trong bài viết này, để thuận tiện, ta có thể viết công thức này dưới dạng:

$$x_t = \mathbf{W}_{rec}\sigma(x_{t-1}) + \mathbf{W}_{in}u_t + \mathbf{b} \quad (**)$$

Trong đó u_t là đầu vào và x_t là trạng thái tại bước thứ t. Các tham số của mô hình được cho bởi ma trận trọng số hồi quy $\mathbf{W}_{rec}$, ma trận trọng số của đầu vào $\mathbf{W}_{in}$ và bias $\mathbf{b}$. σ là một hàm số element-wise (thường là tanh và sigmoid). Hàm mất mát của mô hình là $\mathcal{E} = \sum_{t=1}^{T} \mathcal{E}_t$ với $\mathcal{E}_t = \mathcal{L}(x_t)$.



Hình 3: RNN dưới dạng chuỗi

Lưu ý: Mô hình có công thức $(**)$ tương đương với mô hình có công thức $(*)$ nếu ta lấy $\mathcal{E}_t = \mathcal{L}(\sigma(x_t))$ thay vì $\mathcal{E}_t = \mathcal{L}(x_t)$.

Để áp dụng thuật toán backpropagation, ta phải biểu diễn RNN dưới dạng một một chuỗi các layer giống nhau trong đó mỗi layer nhận đầu vào từ cả layer trước và từ data (xem **Hình 3**). Vì tính chất đặc biệt này của các layer nên các ma trận trọng số tại mỗi bước là giống nhau.

Thuật toán lan truyền ngược theo thời gian về bản chất là thuật toán lan truyền ngược (vẫn sử dụng quy tắc mắc xích để tính gradient của hàm mất mát đối với các trọng số). Cụm từ

"theo thời gian" được thêm vào nhằm nhấn mạnh rằng nó được dùng trong mô hình có yếu tố thời gian (dữ liệu được truyền thêm vào tại mỗi bước).

$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{t=1}^{T} \frac{\partial \mathcal{E}_t}{\partial \theta} \tag{1}$$

$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{k=1}^{t} \left(\frac{\partial \mathcal{E}_t}{\partial x_t} \frac{\partial x_t}{\partial x_k} \frac{\partial x_k}{\partial \theta} \right) \tag{2}$$

$$\frac{\partial x_t}{\partial x_k} = \prod_{i=k+1}^{t} \frac{\partial x_i}{\partial x_{i-1}} = \prod_{i=k+1}^{t} \mathbf{W}_{rec}^{\top} \, diag(\sigma'(x_{t-1})) \tag{3}$$

với θ là trọng số của mô hình.

$\frac{\partial x_t}{\partial x_k}$ ở (3) thể hiện sự ảnh hưởng của kết quả ở bước thứ k đối với bước thứ t. Ta gọi sự ảnh hưởng đó là dài hạn nếu $k \ll t$ và ngắn hạn trong các trường hợp còn lại.

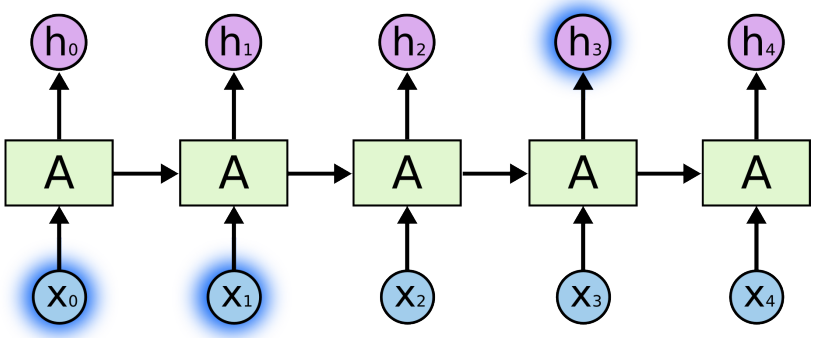
Kết hợp các đẳng thức $(1), (2), (3)$ ta tính được đạo hàm của hàm mất mát $\mathcal{E}$ đối với trọng số θ. Từ (1) để ý rằng điểm đặc biệt của thuật toán backpropagation trong RNN so với thuật toán backpropagation truyền thống là ta cộng tổng đạo hàm của hàm mất mát đối với trọng số tại mỗi bước.

4 VẤN ĐỀ VANISHING/EXPLODING GRADIENT TRONG RNN TRUYỀN THỐNG

4.1 Vấn đề phụ thuộc xa (Long-term dependency)

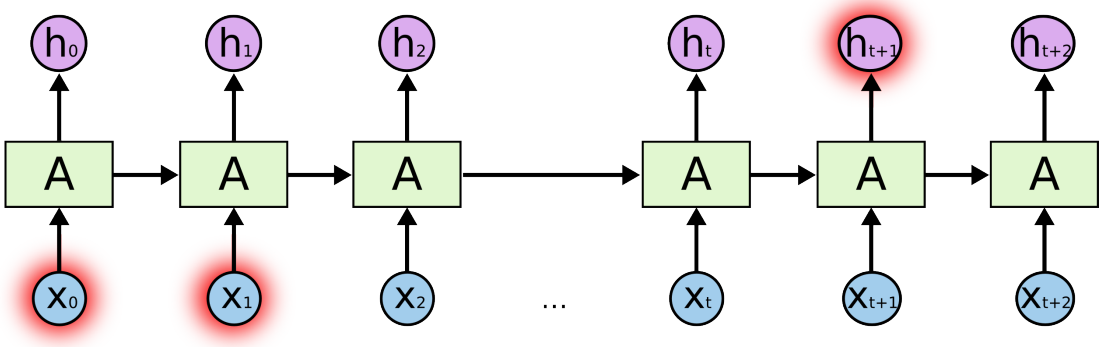
Sức mạnh của RNN nằm ở ý tưởng sử dụng thông tin của các bước trước để thực hiện tác vụ hiện tại. Điều này đặc biệt hữu dụng trong các bài toán có yếu tố thời gian (VD: dịch tự động, phân tích sắc thái bình luận, gợi ý từ ngữ,...). Nếu RNN có thể thực hiện hiệu quả ý tưởng của nó thì ứng dụng của nó là rất lớn.

Trong một số bài toán, ta chỉ cần xem xét thông tin của một số ít bước trước để thực hiện tác vụ tại bước hiện tại. Ví dụ như cần dự đoán từ tiếp theo trong câu: "Anh Phúc đẹp ..." thì mô hình chỉ cần xem xét các từ trong câu để cho ra kết quả là "trai". Trong trường hợp này khoảng cách của các thông tin liên quan là không lớn nên RNN có thể thực hiện không quá khó khăn. Các thông tin trong trường hợp này được gọi là phụ thuộc gần (short-term dependency).



Hình 4: Short-term dependency

Câu hỏi đặt ra ở đây là liệu khi khoảng cách giữa các thông tin liên quan lớn thì liệu RNN có thể xử lý tốt hay không? Ví dụ một đoạn văn "Lâm Hữu Phúc là một du học sinh Mỹ ... Hiện tại anh ấy đã thông thạo tiếng Việt và tiếng ...". Giả sử giữa hai câu văn cách nhau một đoạn dài. Khi đó để dự đoán được từ tiếp theo ta phải biết được rằng "Lâm Hữu Phúc" và "anh ấy" cùng trỏ về một người dù khoảng cách giữa chúng là xa nhau. Thông tin trong trường hợp trên được gọi là phụ thuộc xa (long-term dependency).



Hình 5: Long-term dependency

Trên lý thuyết thì RNN vẫn có thể xử lý được trong trường hợp trên. Tuy nhiên trong thực tế thì lại không như vậy. Đây là vấn đề chính mà mạng RNN gặp phải. Nguyên nhân của vấn đề này là hiện tượng Vanishing Gradient được giải thích kĩ trong phần tiếp theo.

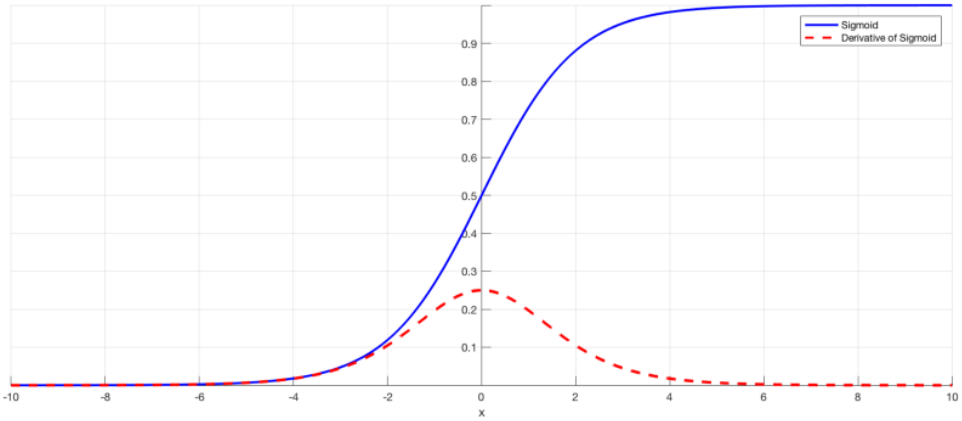
4.2 Vấn đề Vanishing / Exploding Gradient

Trong một Neural Network (NN), thông tin được truyền từ lớp đầu vào đến lớp đầu ra còn gradient của hàm mất mát được tính bằng cách lan truyền ngược (backpropagation) từ lớp đầu ra về lại lớp đầu vào sau đó sử dụng thuật toán gradient descent để cập nhật các trọng số. Tuy nhiên đối với NN nhiều lớp thì khi lan truyền ngược đến các lớp thấp hơn thì gradient sẽ nhỏ dần và tiến dần về 0. Hơn nữa, đối với giá trị quá gần với 0 thì máy tính sẽ làm tròn về 0 sẽ dẫn đến việc trọng số không được cập nhật. Hiện tượng này được gọi là vanishing gradient. Ngược lại, exploding gradient là hiện tượng mà gradient sẽ lớn dần khi tới các lớp thấp khiến nó vượt quá khả năng tính toán của máy tính. Tiếp theo chúng ta sẽ xem xét kĩ lưỡng nguyên nhân xảy ra hai vấn đề này.

4.3 Nguyên nhân

Trong RNN, các hàm sigmoid, tanh,... thường được sử dụng làm activation function ở các layer. Đặc điểm chung của các hàm này là khi tiến dần tới cận trên/dưới của nó thì sự thay đổi lớn ở đầu vào chỉ khiến cho giá trị của hàm thay đổi rất ít. Vì vậy đạo hàm của hàm số tại các điểm này là rất nhỏ ($\ll 1$). Điều này dẫn đến hiện tượng vanishing gradient do các trọng số của ta gần như không được cập nhật và coi như là không "học" được gì.

Điều này có thể xảy ra khi các trọng số khởi tạo của chúng ta không tốt. Ví dụ khi giá trị tuyệt đối của các trọng số quá lớn thì khi đi qua hàm sigmoid đạo hàm của nó sẽ tiến dần tới 0. Tuy nhiên, thậm chí khi trọng số của chúng ta đã được khởi tạo tốt thì vấn đề này vẫn xảy ra. Ví dụ các trọng số được khởi tạo sao cho sau khi đi qua hàm sigmoid thì đạo hàm của hàm số bằng khoảng 0.2 (tốt bởi $\sigma'(x) \in [0, 0.25]$) thì sau khi qua n lớp thì đạo hàm chỉ có giá trị 0.2^n. Đây là con số rất nhỏ (≈ 0) bởi số lớp trong một mô hình RNN thực tế là khá lớn. Chúng ta sẽ chứng minh chặt chẽ ý tưởng này trong phần tiếp theo.



Hình 6: Đồ thị của hàm sigmoid và đạo hàm của nó

Ở **Hình 6** ta đưa ra ví dụ là hàm sigmoid. Có thể thấy rõ từ đồ thị rằng khi x tiến tới $+\infty$ hay $-\infty$ thì đạo hàm của hàm số tiến tới 0.

4.4 Biểu diễn và chứng minh toán học cho vấn đề Vanishing / Exploding Gradient trong RNN

Để hiểu về vấn đề này, xét biểu thức $\frac{\partial x_t}{\partial x_k}$ có dạng là tích của $t-k$ ma trận Jacobian (xem (3)). Ở đây ta sẽ chứng minh norm của tích này tiến về 0 hay tăng lên tới $+\infty$ tương tự như tích các số thực. Ta chứng minh kết quả sau:

Cho mạng RNN như định nghĩa ở phần trước với σ là hàm số thỏa $\sigma'(x)$ bị chặn bởi $\gamma \in \mathbb{R}$. Chứng minh rằng điều kiện đủ để xảy ra vanishing gradient là $\lambda < \frac{1}{\gamma}$ với $\lambda = \max\{\lvert\lambda_1\rvert, \lvert\lambda_2\rvert, \cdots, \lvert\lambda_n\rvert\}$ và $\lambda_1, \lambda_2, \cdots, \lambda_n$ là các eigenvalue của $\mathbf{W}_{rec}$

Chứng minh

Từ điều kiện của σ ta có:

$$\left\| diag\left(\sigma'(x_k)\right)\right\| \leq \gamma,\ \forall k \tag{4}$$

Áp dụng bất đẳng thức Cauchy-Schwarz và (4):

$$\left\|\frac{\partial x_{k+1}}{\partial x_k}\right\| = \left\|\mathbf{W}_{rec}^{\top}\, diag(\sigma'(x_k)\right\| \leq \left\|\mathbf{W}_{rec}^{\top}\right\| \left\|diag(\sigma'(x_k)\right\| < \gamma \cdot \frac{1}{\gamma} < 1,\ \forall k \tag{5}$$

$$\Rightarrow \exists \eta \in \mathbb{R} : \left\|\frac{\partial x_{k+1}}{\partial x_k}\right\| \leq \eta < 1,\ \forall k$$

Từ đây bằng quy nạp ta chứng minh được rằng:

$$\frac{\partial \mathcal{E}_t}{\partial x_t}\left(\prod_{i=k}^{t-1}\frac{\partial x_{i+1}}{\partial x_i}\right) \leq \eta^{t-k}\frac{\partial \mathcal{E}_t}{\partial x_t} \tag{6}$$

Do $\eta < 1$ nên đẳng thức này chứng minh được rằng sự ảnh hưởng của kết quả ở bước thứ k đối với bước thứ t sẽ giảm theo hàm số mũ nếu t càng lớn. Đây chính là hiện tượng vanishing gradient.

Thay đổi điều kiện của λ trong kết quả trên thành $\lambda > \frac{1}{y}$ ta có được điều kiện cần để xảy ra vấn đề exploding gradient.

Kết quả này có ý nghĩa bởi các hàm activation trong RNN như sigmoid, tanh đều thỏa tính chất của hàm σ trong các kết quả trên. Cụ thể thì γ ứng với hàm sigmoid là $\frac{1}{4}$ và với hàm tanh là 1.

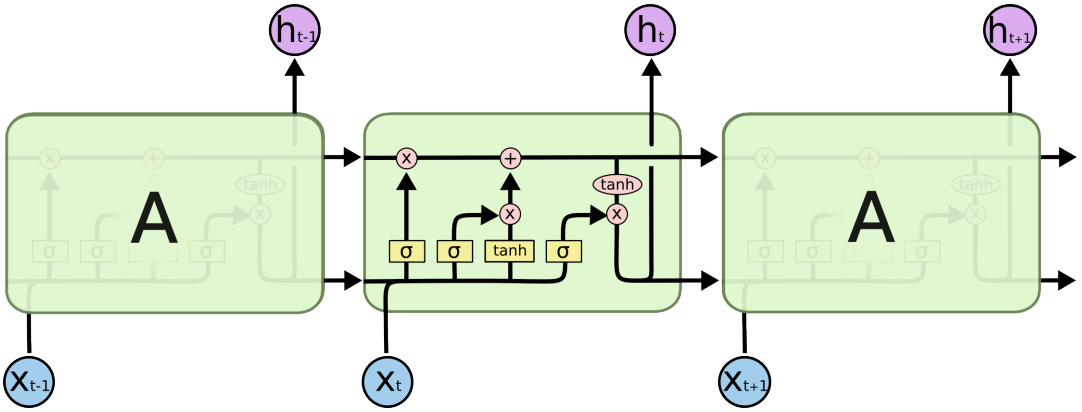
5 LSTMs (Long Short Term Memory networks) - Sự cải tiến của RNN

5.1 Lịch sử ra đời

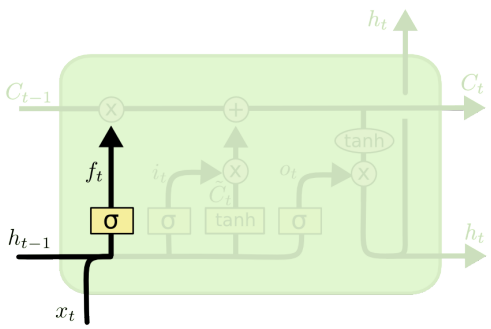
Để giải quyết các khó khăn của mạng RNN truyền thống là Gradient Vanishing và Gradient Exploding, vào năm 1997, Sepp Hochreiter và Jürgen Schmidhuber lần đầu tiên giới thiệu về mạng LSTM.

Dựa trên phiên bản gốc, về sau, LSTM được nhiều nhà khoa học máy tính khác cải tiến qua nhiều phiên bản và được sử dụng rất rộng rãi cho đến ngày nay bởi tính hiệu quả đến bất ngờ của nó trên nhiều loại bài toán khác nhau.

5.2 Cấu trúc



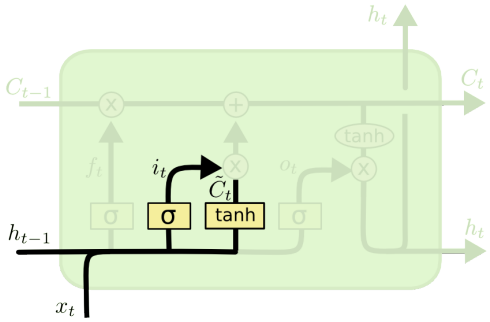
Mạng LSTM có cấu trúc "mắt xích" (chain structure). Mỗi đơn vị bao gồm 4 lớp Neural Network tương tác chặt chẽ với nhau; các "khối nhớ" khác nhau còn được gọi là các cell của mỗi đơn vị.



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

- Đầu tiên, những thông tin không còn "có ích" trong cell state sẽ bị xóa bằng Forget Gate.

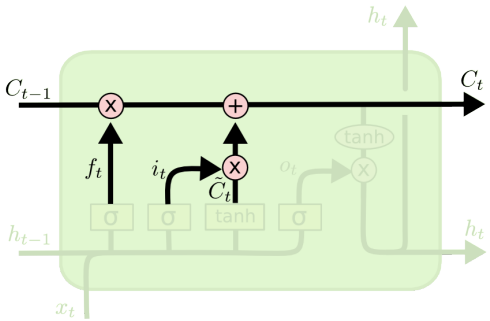
Input x_t (input tại một thời điểm cụ thể) và input h_{t-1} (output của cell kề trước) được đưa vào Forget Gate, lần lượt được nhân với các ma trận trọng số và chuẩn hóa bằng cách cộng với độ lệch (bias). Kết quả khi qua hàm kích hoạt sigmoid sẽ cho output có các giá trị nằm trong khoảng $[0, 1]$. Trong một cell state cụ thể, nếu output là 0 thì toàn bộ thông tin sẽ bị quên và và nếu output là 1 thì toàn bộ thông tin sẽ được giữ lại để sử dụng về sau.



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

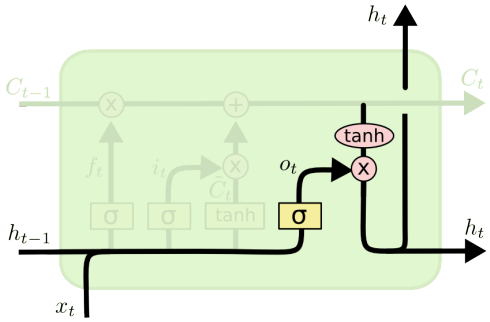
- Tiếp theo, Input Gate có nhiệm vụ thêm những thông tin "có ích" vào cell state.

Ban đầu, thông tin được điều chỉnh bằng việc sử dụng hàm sigmoid để chọn lọc giá trị nào sẽ được giữ lại, qua việc sử dụng các input x_t và h_{t-1} như Forget Gate. Tiếp theo, một vector được tạo ra bằng cách áp dụng hàm tanh (cho output trong đoạn $[-1, 1]$).



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

Sau đó, các giá trị của vector nói trên và các giá trị điều chỉnh được nhân với nhau để tạo được một thông tin "có ích" để cập nhật cho cell state.



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

- Cuối cùng, việc trích xuất thông tin từ cell state tại một thời điểm cụ thể sẽ được thực hiện bằng Output Gate.

Đầu tiên, một vector sẽ được tạo ra bằng cách áp dụng hàm tanh cho cell state để cho output vào khoảng $[-1, 1]$ rồi nhân với output của hàm sigmoid để được output mà ta mong muốn. Và output đó sẽ được làm input cho cell trong đơn vị tiếp theo.
Một cách tương tự, các quá trình trên được lặp lại ở các đơn vị tiếp theo.

5.3 Giải thích tính hiệu quả của LSTM

Như đã đề cập ở đề mục lớn của phần này, ta có thể xem LSTM là một sự cải tiến, một sự điều chỉnh khôn khéo về mặt toán học của RNN truyền thống để giải quyết hai khó khăn là Vanishing Gradient và Exploding Gradient.
Đặt E_t là hàm lỗi tại đơn vị thứ t của mạng LSTM.
Theo Quy tắc "Mắt xích" (Chain Rule)

$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial H_t} \cdot \frac{\partial H_t}{\partial C_t} \cdot \frac{\partial C_t}{\partial C_{t'}} \cdot \frac{\partial C_{t'}}{\partial W} \tag{7}$$

Ta có

$$\frac{\partial C_k}{\partial C_{k-1}} = \frac{\partial (f_k * C_{k-1})}{\partial C_{k-1}} + \frac{\partial (i_k * \tilde{C}_{k-1})}{\partial C_{k-1}} \approx f_k \tag{8}$$

Do đó với $t > t'$

$$\frac{\partial C_t}{\partial C_{t'}} = \prod_{k=t'+1}^{t} \frac{\partial C_k}{\partial C_{k-1}} \approx \prod_{k=t'+1}^{t} f_k$$

Thay vào (7) ta có

$$\frac{\partial E_t}{\partial W} \approx \frac{\partial E_t}{\partial H_t} \cdot \frac{\partial H_t}{\partial C_t} \cdot \prod_{k=t'+1}^{t} f_k \cdot \frac{\partial C_{t'}}{\partial W}$$

Để ý đẳng thức (8), Gradient $\frac{\partial C_k}{\partial C_{k-1}}$ xấp xỉ với hàm Forget Gate f_k, và nếu Forget Gate quyết định thông tin nào đó cần nhớ, nó sẽ "mở" và các giá trị sẽ gần với 1 để cho thông tin đi vào. Để đơn giản, ta có thể xem

$$f_k \approx \vec{1}$$

Khi đó, $\frac{\partial C_t}{\partial C_{t'}}$ sẽ không tiến về 0. Và cuối cùng, $\frac{\partial E_t}{\partial W}$ cũng sẽ không tiến về 0.
Như vậy, trong mạng LSTM, vấn đề Vanishing Gradient sẽ khó xảy ra hơn so với RNN truyền thống.

Hơn nữa, với giải thích như trên, ta cũng có thể khẳng định mạng LSTM cũng tránh được vấn đề Exploding Gradient vì các giá trị luôn bị chặn bởi 1 bởi sự kích hoạt của hàm sigmoid f_k.

6 BÀI TOÁN SENTIMENT ANALYSIS

6.1 Giới thiệu bài toán

Sentiment Analysis - hay phân tích sắc thái - là một bài toán Học máy đề cập đến việc phân tích cảm xúc của con người thông qua văn bản. Đầu vào của bài toán thường là các câu văn, đoạn văn hoặc tài liệu ngôn ngữ tự nhiên (Tiếng Việt, Tiếng Anh) và đầu ra của bài toán là các loại cảm xúc (tích cực, trung lập, tiêu cực). Sentiment Analysis giúp các doanh nghiệp quan tâm đến cảm xúc, đánh giá của người tiêu dùng trên các dịch vụ, sản phẩm.

Để giải quyết Sentiment Analysis, có thể sử dụng nhiều phương pháp. Một trong số đó chính là áp dụng RNN kết hợp với cấu trúc LSTM

6.2 Ví dụ về cài đặt mạng RNN kết hợp LSTM trong Python để giải Sentiment Analysis

Mô hình mạng RNN kết hợp LSTM dưới đây được cài đặt trong Python với thư viện Keras

6.2.1 *Nạp tệp dữ liệu*

Dataset được sử dụng trong ví dụ cài đặt này là Sentiment140 - tập dữ liệu gồm 1,600,000 Tweet trên Tweeter, đã được làm sạch emoji (biểu tượng cảm xúc) Ta import các thư viện cần thiết

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import keras.preprocessing.text as kpt


from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, Dropout
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
import re

from collections import Counter
```

Quy định một số tham số cố định

```
# dataset
data_col = ["target", "ids", "date", "flag", "user", "text"]
data_encode = "ISO-8859-1"
train_size = 0.8

# cleaning
text_cleaning_re = "@\S+|https?:\S+|http?:\S|[^A-Za-z0-9]+"

# WORD2VEC
w2v_size = 300
w2v_window = 7
w2v_epoch = 32
w2v_min_count = 10

# KERAS
seq_len = 300
epochs = 8
batch_size = 1024

# SENTIMENT
posi = "POSITIVE"
nega = "NEGATIVE"
neutral = "NEUTRAL"
sentiment_thresholds = (0.4, 0.7)

# EXPORT
lstm_model = "model.h5"
word2vec_model = "model.w2v"
tokenizer_model = "tokenizer.pkl"
encoder_model = "encoder.pkl"
```

Sử dụng thư viện panda để load dataset vào biến df, sau đó kiểm tra lại kích thước.

```
dataset_path = os.path.join("/content/drive/My Drive/Colab Notebooks/[PIMA 2019][Sentiment Analysis]/fulldata.csv")
print("Open file:", dataset_path)
df = pd.read_csv(dataset_path, encoding =data_encode , names=data_col)
```

Vì bên trong datasets có các chỉ số quy định cảm xúc của câu (0 là Negative - tiêu cực, 1 là Neutral - trung lập và 2 là Positive - tích cực) nên ta cần tạo một Map để chỉ dữ liệu từ số sang trạng thái thích hợp

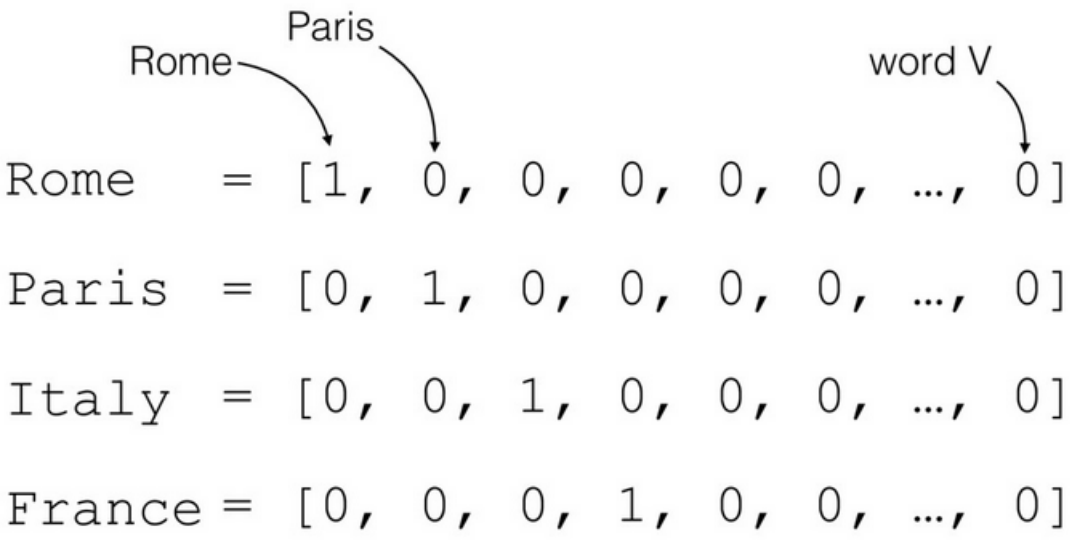
```
decode_map = {0: "NEGATIVE", 2: "NEUTRAL", 4: "POSITIVE"}
def decode_sentiment(label):
    return decode_map[int(label)]
df.target = df.target.apply(lambda x: decode_sentiment(x))
```

6.2.2 *Tiền xử lý*

1. **Word2Vec**

   Với một bài xử lý ngôn ngữ tự nhiên NLP, tiền xử lý là giai đoạn quan trọng, vì máy tính chỉ hiểu được các vector đại số, không thể hiểu được văn bản. Do đó, cần phải có một phương pháp chuyển đổi các văn bản sang các vector đại số.

   Cách truyền thống thường xuyên được sử dụng được gọi là one-hot encoding

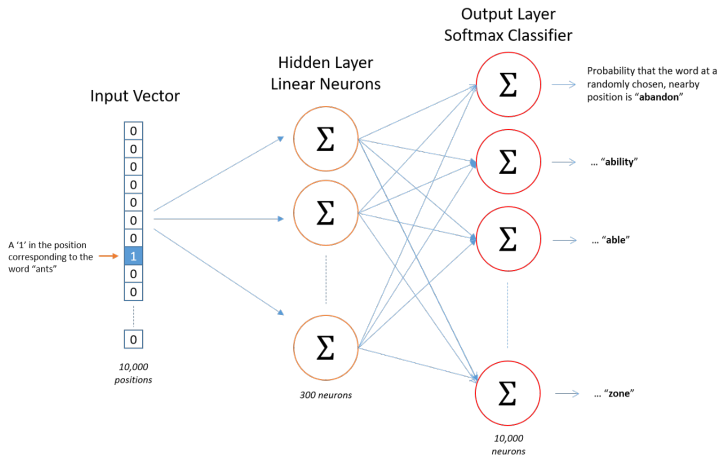


. Các từ sẽ được thể hiện dưới dạng các one-hot vector. Đặc điểm của one-hot vector:

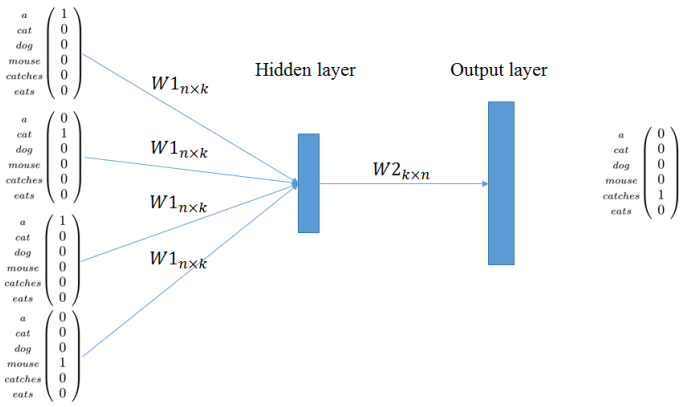
- Độ lớn vector bằng số lượng từ vựng
- Vector chỉ bao gồm các số 0 và một số 1 duy nhất tại vị trí của chính từ đó trong bộ từ điển
- Không thể hiện được sự tương quan hay liên hệ giữa các từ

Đặc điểm cuối cùng của one-hot vector chính là động lực để tìm ra những phương pháp vectorize tốt hơn mà vẫn thể hiện được những mối liên hệ về ngữ nghĩa, ngữ pháp của các từ. Một trong những phương pháp đó chính là Word2Vec. Có 2 dạng Word2Vec chính được khái quát như sau:

- Skip-gram: dùng để tìm các từ có mối quan hệ với từ đã cho.



- Continuous Bag of Words(CBOW): dùng tìm từ phù hợp với ngữ cảnh đã cho



Ở trong mô hình mạng LSTM này, dựa vào công dụng của hai loại Word2Vec, Skip-gram Word2Vec là phương pháp phù hợp để tiền xử lý bộ từ vựng.

Đầu tiên, tìm bộ stopwords phù hợp nhờ thư viện nltk nhằm loại bỏ các từ không có ý nghĩa hoặc các kí tự đặc biệt trong thứ tiếng đang sử dụng (ở đây là tiếng Anh)

```
!pip install nltk
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from  nltk.stem import SnowballStemmer

stop_words = stopwords.words("english")
stemmer = SnowballStemmer("english")
```

Thực hiện phương pháp Train-test split: Tách tập data ban đầu thành 2 tập train và test nhằm sử dụng để huấn luyện và kiểm chứng mô hình, chống hiện tượng Overfitting

```
df_train, df_test = train_test_split(df, test_size=1-train_size, random_state=42)
print("TRAIN size:", len(df_train))
print("TEST size:", len(df_test))
```

```
TRAIN size: 1280000
TEST size: 320000
```

Chia nhỏ văn bản thành các từ và gói vào biến document nhằm tạo bộ data cho việc huấn luyện mô hình Word2Vec.

```
documents = [_text.split() for _text in df_train.text]
```

Huấn luyện Word2Vec

```
import gensim

w2v_model = gensim.models.word2vec.Word2Vec(size=w2v_size,
                                            window=w2v_window,
                                            min_count=w2v_min_count,
                                            workers=8)
```

```
w2v_model.build_vocab(documents)
```

```
words = w2v_model.wv.vocab.keys()
vocab_size = len(words)
print("Vocab size", vocab_size)
```

```
Vocab size 30369
```

```
w2v_model.train(documents, total_examples=len(documents), epochs=w2v_epoch)
```

```
(263128520, 295270528)
```

```
w2v_model.wv.most_similar("love")
```

```
/usr/local/lib/python3.6/dist-packages/g
  if np.issubdtype(vec.dtype, np.int):
[('luv', 0.5712414979934692),
 ('loves', 0.5481855869293213),
 ('loved', 0.5420869588851929),
 ('adore', 0.5171909928321838),
 ('amazing', 0.5106487274169922),
 ('looove', 0.48079293966293335),
 ('awesome', 0.47822606563568115),
 ('loooove', 0.45320290327072144),
 ('miss', 0.44124847650527954),
 ('lovee', 0.4344443678855896)]
```

2. **Tokenizing và Padding**

   Tokenizing và padding là quá trình cắt nhỏ văn bản thành các từ và vector hóa thành các vector có chiều bằng nhau

```
[ ]  tokenizer = Tokenizer()
     tokenizer.fit_on_texts(df_train.text)

     vocab_size = len(tokenizer.word_index) + 1
     print("Total words", vocab_size)
```

```
Total words 290419
```

```
x_train = pad_sequences(tokenizer.texts_to_sequences(df_train.text), maxlen=seq_len)
x_test = pad_sequences(tokenizer.texts_to_sequences(df_test.text), maxlen=seq_len)
```

Chỉnh sửa nhãn và gán nhãn vào tệp nhãn kết quả

```
[ ]  labels = df_train.target.unique().tolist()
     labels.append(neutral)
     labels
```

```
['POSITIVE', 'NEGATIVE', 'NEUTRAL']
```

```
[ ]  from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import LabelEncoder

     encoder = LabelEncoder()
     encoder.fit(df_train.target.tolist())

     y_train = encoder.transform(df_train.target.tolist())
     y_test = encoder.transform(df_test.target.tolist())

     y_train = y_train.reshape(-1,1)
     y_test = y_test.reshape(-1,1)

     print("y_train",y_train.shape)
     print("y_test",y_test.shape)
```

```
y_train (1280000, 1)
y_test (320000, 1)
```

Khi thực hiện xong hai kỹ thuật trên, tạo Embedding Layer cho mạng RNN từ các vector nhận được.

Embedding

```
[ ]  embedding_matrix = np.zeros((vocab_size, w2v_size))
     for word, i in tokenizer.word_index.items():
       if word in w2v_model.wv:
         embedding_matrix[i] = w2v_model.wv[word]
     print(embedding_matrix.shape)
```

```
(290419, 300)
```

```
[ ]  embedding_layer = Embedding(vocab_size, w2v_size, weights=[embedding_matrix], input_length=seq_len, train
```

6.2.3 *Xây dựng mô hình mạng LSTM*

Tạo mô hình mạng LSTM đơn giản (1 lớp LSTM)

Building Model

```
[ ]  model = Sequential()
     model.add(embedding_layer)
     model.add(Dropout(0.5))
     model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
     model.add(Dense(1, activation='sigmoid'))

     model.summary()
```

```
W0807 19:16:40.403254 139876604532608 deprecation.py:506] From /usr/local/lib/python3.6/dist-packages/keras/backend/te
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 300, 300)          87125700
_________________________________________________________________
dropout_1 (Dropout)          (None, 300, 300)          0
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               160400
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 87,286,201
Trainable params: 160,501
Non-trainable params: 87,125,700
_________________________________________________________________
```

6.2.4 *Xây dựng mô hình mạng RNN*

Tạo mô hình mạng RNN đơn giản (1 lớp RNN) để so sánh hiệu năng với mạng RNN có LSTM

```
[75]  from keras.layers import SimpleRNN
      rnn_model = Sequential()
      rnn_model.add(embedding_layer)
      rnn_model.add(Dropout(0.5))
      rnn_model.add(SimpleRNN(100, dropout=0.2, recurrent_dropout=0.2))
      rnn_model.add(Dense(1, activation='sigmoid'))

      rnn_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 300, 300)          87125700
_________________________________________________________________
dropout_3 (Dropout)          (None, 300, 300)          0
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 100)               40100
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
=================================================================
Total params: 87,165,901
Trainable params: 40,201
Non-trainable params: 87,125,700
_________________________________________________________________
```

6.2.5 *Huấn luyện mô hình*

Bắt đầu huấn luyện 2 mô hình với cùng một hàm mất mát *BinaryCrossentropy* và hàm tối ưu hóa *Adam* Lưu lại quá trình huấn luyện thông qua *ReduceLROnPlateau* và *EarlyStopping*

```
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
callbacks = [ReduceLROnPlateau(monitor='val_loss', patience=5, cooldown=0), EarlyStopping(monitor='val_a
```

```
model.compile(loss='binary_crossentropy',
              optimizer="adam",
              metrics=['accuracy'])
```

Hình 7: LSTM Compile

```
rnn_model.compile(loss='binary_crossentropy',
                  optimizer="adam",
                  metrics=['accuracy'])
```

Hình 8: RNN Compile

Bắt đầu quá trình huấn luyện

```
%%time
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    validation_split=0.1,
                    verbose=1,
                    callbacks=callbacks)
```

```
Train on 1152000 samples, validate on 128000 samples
Epoch 1/8
1152000/1152000 [==============================] - 1023s 888us/step - loss: 0.5079 - acc: 0.7479 - val_loss: 0.4653 - val_acc: 0.7784
Epoch 2/8
1152000/1152000 [==============================] - 1019s 884us/step - loss: 0.4829 - acc: 0.7651 - val_loss: 0.4573 - val_acc: 0.7835
Epoch 3/8
1152000/1152000 [==============================] - 1023s 888us/step - loss: 0.4758 - acc: 0.7701 - val_loss: 0.4532 - val_acc: 0.7868
Epoch 4/8
1152000/1152000 [==============================] - 1023s 888us/step - loss: 0.4713 - acc: 0.7726 - val_loss: 0.4524 - val_acc: 0.7869
Epoch 5/8
1152000/1152000 [==============================] - 1019s 885us/step - loss: 0.4691 - acc: 0.7736 - val_loss: 0.4520 - val_acc: 0.7876
Epoch 6/8
1152000/1152000 [==============================] - 1020s 886us/step - loss: 0.4675 - acc: 0.7746 - val_loss: 0.4487 - val_acc: 0.7890
Epoch 7/8
1152000/1152000 [==============================] - 1020s 885us/step - loss: 0.4654 - acc: 0.7759 - val_loss: 0.4481 - val_acc: 0.7889
Epoch 8/8
1152000/1152000 [==============================] - 1019s 885us/step - loss: 0.4638 - acc: 0.7771 - val_loss: 0.4469 - val_acc: 0.7904
CPU times: user 2h 18min 52s, sys: 21min 52s, total: 2h 40min 44s
Wall time: 2h 16min 7s
```

```
%%time
rnn_history = rnn_model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    validation_split=0.1,
                    verbose=1,
                    callbacks=callbacks)
```

```
Train on 1152000 samples, validate on 128000 samples
Epoch 1/8
1152000/1152000 [==============================] - 370s 321us/step - loss: 7.5608 - acc: 0.5059 - val_loss: 7.9692 - val_acc: 0.5001
Epoch 2/8
1152000/1152000 [==============================] - 370s 321us/step - loss: 7.9784 - acc: 0.4995 - val_loss: 7.9692 - val_acc: 0.5001
Epoch 3/8
1152000/1152000 [==============================] - 372s 323us/step - loss: 7.9784 - acc: 0.4995 - val_loss: 7.9692 - val_acc: 0.5001
Epoch 4/8
1152000/1152000 [==============================] - 369s 320us/step - loss: 7.9784 - acc: 0.4995 - val_loss: 7.9692 - val_acc: 0.5001
Epoch 5/8
1152000/1152000 [==============================] - 369s 320us/step - loss: 7.9784 - acc: 0.4995 - val_loss: 7.9692 - val_acc: 0.5001
Epoch 6/8
1152000/1152000 [==============================] - 370s 321us/step - loss: 7.9784 - acc: 0.4995 - val_loss: 7.9692 - val_acc: 0.5001
CPU times: user 41min 9s, sys: 5min 36s, total: 46min 45s
Wall time: 36min 59s
```

6.2.6 *Dự đoán*

Sử dụng mô hình LSTM đã được huấn luyện để thử dự đoán một vài văn bản

Predicting

```
[65]  def decode_sentiment(score, include_neutral=True):
          if include_neutral:
              label = neutral
              if score <= sentiment_thresholds[0]:
                  label = nega
              elif score >= sentiment_thresholds[1]:
                  label = posi

              return label
          else:
              return nega if score < 0.5 else posi
```

```
[66]  import time
      def predict(text, include_neutral=True):
          start_at = time.time()
          # Tokenize text
          x_test = pad_sequences(tokenizer.texts_to_sequences([text]), maxlen=seq_len)
          # Predict
          score = model.predict([x_test])[0]
          # Decode sentiment
          label = decode_sentiment(score, include_neutral=include_neutral)

          return {"label": label, "score": float(score),
             "elapsed_time": time.time()-start_at}
```

```
predict("I love it")
```

```
{'elapsed_time': 0.20113492012023926,
 'label': 'POSITIVE',
 'score': 0.8805591464042664}
```

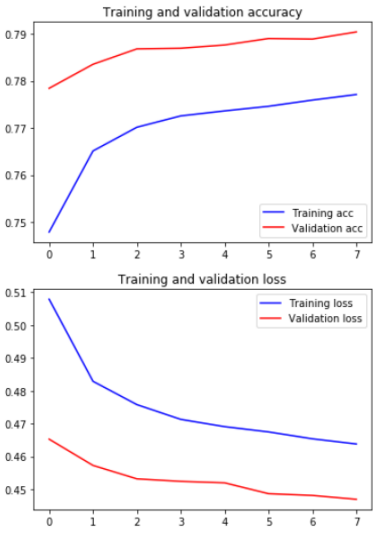
```
[70]  predict("I don't want to talk about it")
```

```
{'elapsed_time': 0.19706273078918457,
 'label': 'NEGATIVE',
 'score': 0.21893535554409027}
```

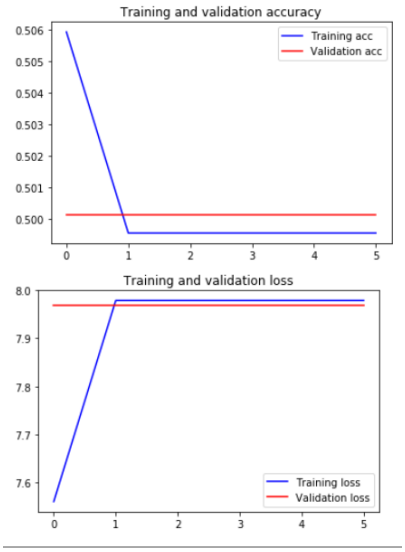
s

6.2.7 *Nhận xét*

- Về thời gian chạy: LSTM mất khá nhiều thời gian để huấn luyện xong (hơn tận 1 tiếng 40 phút so với mạng RNN)
- Về loss và accuracy:



Hình 10: LSTM Plotting



Hình 11: RNN Plotting

```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'b', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

Hình 9: Code Plotting

Dựa vào hai đồ thị trên có thể đưa ra nhận xét sau: Mô hình LSTM hoạt động hiệu quả hơn rất nhiều so với mô hình RNN thông thường, với loss lẫn validation loss đều thấp và accuracy đều cao (80%)

TÀI LIỆU

[1] Shashank Gupta. Sentiment Analysis: Concept, Analysis and Applications.

[2] Christopher Olah. Understanding LSTM Networks.

[3] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training Recurrent Neural Networks. Nov 2012.

[4] Trieu H. Trinh, Andrew M. Dai, Minh-Thang Luong, and Quoc V. Le. Learning Longer-term Dependencies in RNNs with Auxiliary Losses. Feb 2018.

[5] Chi-Feng Wang. The Vanishing Gradient Problem.