Project in Mathematics and Applications 2019

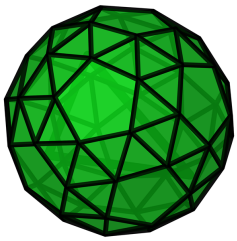
Deep Learning

nhóm 3

Sentiment Analysis

Nguyễn Tài Tuệ[1]
Nguyễn Hoàng Thuận[2]
Nguyễn Phong Phú[3]
Nguyễn Kim Phương Trang[4]

29th August 2019



---

1. Trường Phổ Thông Năng Khiếu - ĐHQG TPHCM
2. Trường THPT Trị An, Đồng Nai
3. Trường THPT chuyên Nguyễn Đình Chiểu, Sa Đéc
4. Trường THPT chuyên Lương Văn Chánh, Phú Yên

Contents

Lời cảm ơn

Đầu tiên, chúng tôi xin chân thành gửi lời cảm ơn đến Ban tổ chức PiMA vì đã hết lòng giúp đỡ, phát triển chương trình trong xuyên suốt bốn mùa để chúng tôi có được cơ hội trải nghiệm một trại hè Toán học bổ ích và thú vị. Đến với PiMA, chúng tôi không chỉ tiếp xúc với những kiến thức mới lạ, mà còn được nhìn thấy rõ ràng hơn sự liên kết giữa khoa học với đời sống thường nhật thông qua những chia sẻ từ mentor, cũng như những diễn giả đầy nhiệt huyết và kinh nghiệm. Họ đã tận tình chỉ dạy và mong muốn lan toả niềm đam mê, niềm yêu thích toán học của mình đến với các học sinh THPT như chúng tôi. Không chỉ có thế, chương trình đã mang những học sinh có cùng niềm yêu thích toán học từ mọi hoàn cảnh, mọi vùng miền đến gần nhau hơn, tạo môi trường để chúng tôi chia sẻ và học tập lẫn nhau. Những câu chuyện ở PiMA là điều khiến mùa hè của chúng tôi có ý nghĩa hơn bao giờ hết.

Tiếp đến, chúng tôi muốn cảm ơn các vị mentor đã giúp đỡ nhóm là anh Nguyễn Hồ Thăng Long, anh Vũ Lê Thế Anh và chị Phạm Thanh Ngọc đã theo dõi và nhẫn nại giúp đỡ nhóm trong suốt quá trình nghiên cứu hoàn thành được dự án. Các anh chị chính là niềm cảm hứng của chúng tôi.

Cuối cùng, nhóm xin cảm ơn trường Đại học Khoa học Tự nhiên TP.HCM cùng với các đơn vị tài trợ khác đã tạo điều kiện, cung cấp cơ sở vật chất để trại hè PiMA có thể diễn ra thành công tốt đẹp.

Chúng tôi tin rằng, dù trại hè năm nay có khép lại, hành trình PiMA cũng như tình cảm của mỗi trại viên cũng sẽ không bao giờ là kết thúc. Rằng mỗi người chúng tôi sẽ sử dụng những gì có được từ PiMA làm động lực thúc đẩy bản thân vươn lên trong tương lai.

Với ảnh hưởng tích cực mà PiMA 2019 đã mang lại, chúng tôi xin chúc chương trình có thể tiếp tục duy trì và tiến xa hơn nữa trong tương lai để những thế hệ học sinh tiếp theo cũng có thể trải nghiệm một trại hè tuyệt vời như vậy.

Xin cảm ơn.

Nhóm 3

Tóm tắt nội dung

Việc nhận định cảm xúc của con người thông qua ngôn ngữ chữ viết là một vấn đề khá mới mẻ. Nhờ vào Internet cùng với khả năng truy cập một lượng lớn thông tin đã tạo điều kiện để các nhà nghiên cứu từ nhiều ngành khác nhau: xử lí ngôn ngữ tự nhiên (NLP), học máy (ML), và thậm chí là tâm lí học,... có thể thực hiện việc nghiên cứu xác định cảm xúc dựa vào nguồn dữ liệu văn bản công khai.

Trong khuôn khổ bài báo cáo lần này, nhóm 3 chúng tôi xin trình bày về mô hình LSTM và cách áp dụng nó để giải quyết bài toán **Sentiment Analysis**. Cụ thể là về những vấn đề mà **Sentiment Analysis** đặt ra (*vanishing gradient, exploding gradient*), cũng như đi sâu vào phân tích những điểm vượt trội của LSTM so với mô hình RNN truyền thống, những cải tiến giúp nó có thể giải quyết vấn đề trong bài toán **Sentiment Analysis** mà RNN truyền thống không làm được. Song, chúng tôi sẽ đề xuất những ứng dụng, hướng phát triển trong tương lai cũng như mã nguồn demo cho mô hình LSTM phục vụ mục đích tham khảo.

Keywords: Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), Sentiment Analysis, Vanishing Gradient, Exploding Gradient, Backpropagation Through Time (BPTT)

1 Giới thiệu bài toán Sentiment Analysis

NLP (*Natural Language Processing*) hay còn gọi là xử lí ngôn ngữ tự nhiên là một lĩnh vực quan trọng trong Machine Learning. Một số bài toán quan trọng trong NLP: *Text Classification, Machine Translation, Text Generation*,...

Bài toán **Sentiment Analysis** là một bài toán con trong *Text Classification*. Đối với **Sentiment Analysis** nếu dùng những mô hình *Machine Learning* truyền thống thì chúng ta sẽ biểu diễn văn bản thông qua các từ ngữ quan trọng có ảnh hưởng đến việc phân loại câu. Nhưng phương pháp này có những hạn chế nhất định.

Ví dụ, xét một bài toán **Sentiment Analysis** đơn giản là phân loại bình luận phim, chúng ta có câu như sau: "Phim rất hay nhưng tôi không thích"

Đối với chúng ta, khi đọc sẽ biết ngay đây là một câu bình luận tiêu cực *(Negative)* nhưng nếu áp dụng một số mô hình ML truyền thống thì máy sẽ phân loại đây là bình luận tích cực *(Positive)*. Lí giải khá đơn giản bởi vì ta thấy có sự xuất hiện của từ thường hay xuất hiện trong bình luận tích cực đó là "hay" . Trong trường hợp này mô hình truyền thống lại tỏ ra không hiệu quả.

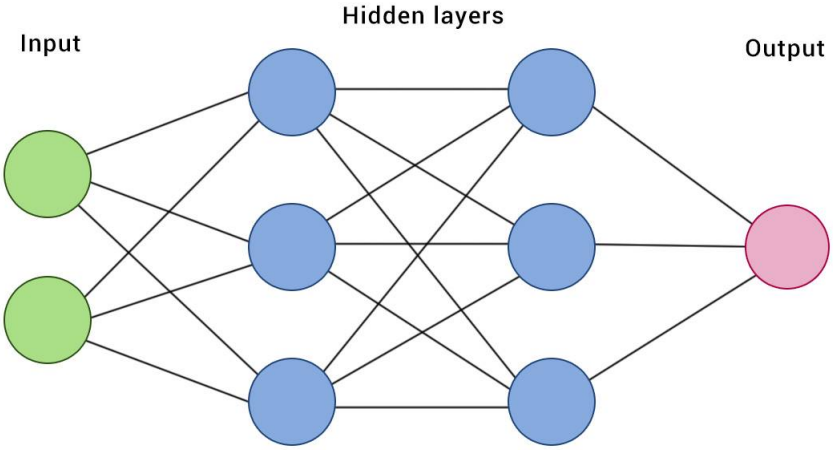
Như vậy ta mong muốn tạo được một mô hình có thể "hiểu" được ngữ nghĩa của câu hay ít nhất cũng phải "nhớ" được thứ tự các từ ngữ trong câu.

2 Recurrent Neural Network(RNN)

2.1 Giới thiệu

Trong mạng *Neural Network* (NN) thông thường, chúng ta mặc định tất cả dữ liệu đầu vào cũng như các dữ liệu đầu ra là hoàn toàn độc lập với nhau, và mặc dù ý tưởng này có thể giúp ta giải quyết được nhiều bài toán nhưng trong một số vấn đề nhất định, khi mà giả thuyết của chúng ta sai - tức các dữ liệu đầu vào (đầu ra) phụ thuộc vào nhau, mô hình NN lại trở nên tồi tệ. Ví dụ như trong trường hợp muốn dự đoán tình tiết tiếp theo của bộ phim thì phải tận dụng những thông tin từ các phần trước của bộ phim; lúc này, các dữ liệu phụ thuộc vào nhau và tuân theo một trình tự thì không thể sử dụng NN được nữa.

Được khám phá vào năm 1982 bởi John Hopfield, *Recurrent Neural Network* (RNN) ra đời. Khác với mạng NN, *Recurrent Neural Network* (RNN) tận dụng thêm những thông tin về thứ tự. RNN ghi nhớ những gì nó đã thực hiện ở quá khứ và những kí ức đó sẽ ảnh hưởng đến những tính toán trong tương lai. Nhờ vào khả năng này cùng với tính linh động trong dữ liệu đầu vào và đầu ra (không cố định kích thước) mà RNN đã là một phát minh đột phá trong việc phát triển nhận dạng giọng nói *(speech recognition)* và xử lí ngôn ngữ tự nhiên *(natural language processing - NLP)*...

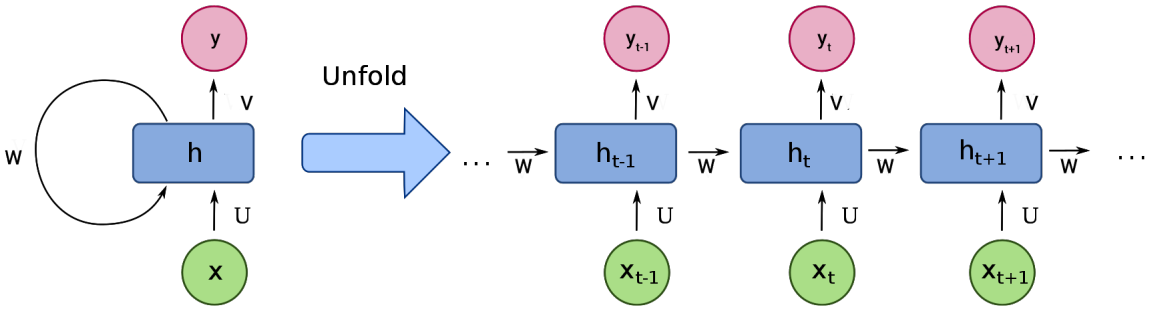


Hình 1: Cấu trúc của Neural Network (NN)

2.2 Cấu trúc

Đầu tiên, hãy cùng nhìn lại mạng NN truyền thống: Trong NN truyền thống, trọng số (*weight*) giữa 2 nơ ron khác nhau thì khác nhau. Còn RNN sử dụng cùng một bộ trọng số tại mọi thời điểm t, việc này được gọi là chia sẻ trọng số theo thời gian (*weight shared across time*). Cách này giúp giảm độ phức tạp đi rất nhiều khi ta cần cập nhật trọng số.

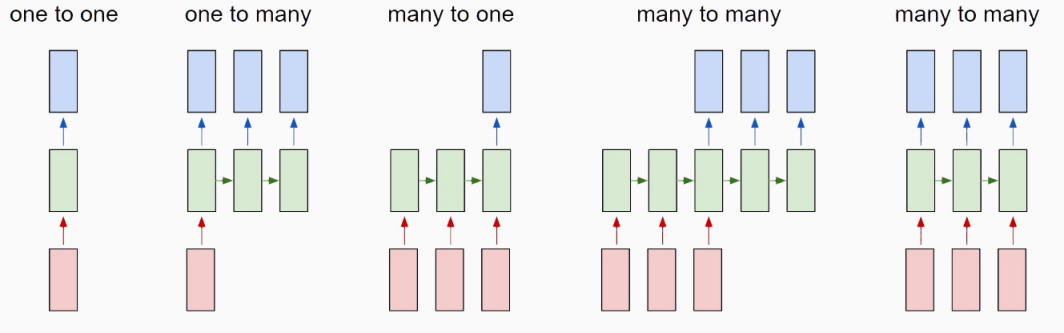
Theo lí thuyết, việc sử dụng trạng thái ẩn tại thời điểm $t-1$ làm input để tính toán trạng thái ẩn tại thời điểm t (tính h_t dựa vào h_{t-1}) giúp cho RNN có thể ghi nhớ được tất cả các thông tin từ thời điểm bắt đầu đến thời điểm t. Nhưng trên thực tế, RNN không thể ghi nhớ những thông tin từ những thời điểm cách xa nó, việc này sẽ được đề cập ở *Vanishing/Exploding Gradient*.
Mạng RNN:



Hình 2: Sơ đồ cấu trúc của RNN

Trong đó:
x_t là đầu vào tại thời điểm t,
h_t là trạng thái ẩn tại thời điểm t. Đây được xem như là trí nhớ của RNN. Được tính dựa vào input và trạng thái ẩn tại thời điểm $t-1$, tức x_t và h_{t-1}; y_t là đầu ra tại thời điểm t.

Biểu đồ trên thấy mạng RNN cho ra output tại từng thời điểm t, nhưng điều này còn phụ thuộc vào từng bài toán. Ví dụ khi ta cần giải bài toán **Sentiment Analysis** thì chỉ cần quan tâm đến output cuối cùng thôi. Do đó mà RNN linh động hơn rất nhiều so với mạng NN. Trong khi mạng NN giới hạn đầu vào và đầu ra bởi kích thước,... thì RNN không như vậy.
Một số ví dụ như: bài toán nhận diện cảm xúc văn bản nhận vào một chuỗi kí tự



Hình 3: Tùy theo bài toán mà ta có các cách cho ra output

và cho ra kết quả là một số nguyên phân loại sắc thái của chuỗi kí tự đó, bài toán dịch thuật từ ngôn ngữ này sang ngôn ngữ khác nhận vào một chuỗi kí tự và cho ra một chuỗi kí tự có độ dài khác với chuỗi ban đầu, ngoài ra còn có nhiều bài toán khác như nhận diện giọng nói, phân tích chuỗi ADN,.. Từ biểu đồ trên ta cũng có thể dễ dàng suy ra công thức tính trạng thái ẩn h_t (dựa vào x_t và h_{t-1}) tổng quát như sau:

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

Ở phiên bản đầu tiên của RNN thì các phép tính toán được thực hiện như sau

$$\mathbf{h_1} = \tanh(\mathbf{U x_1})$$

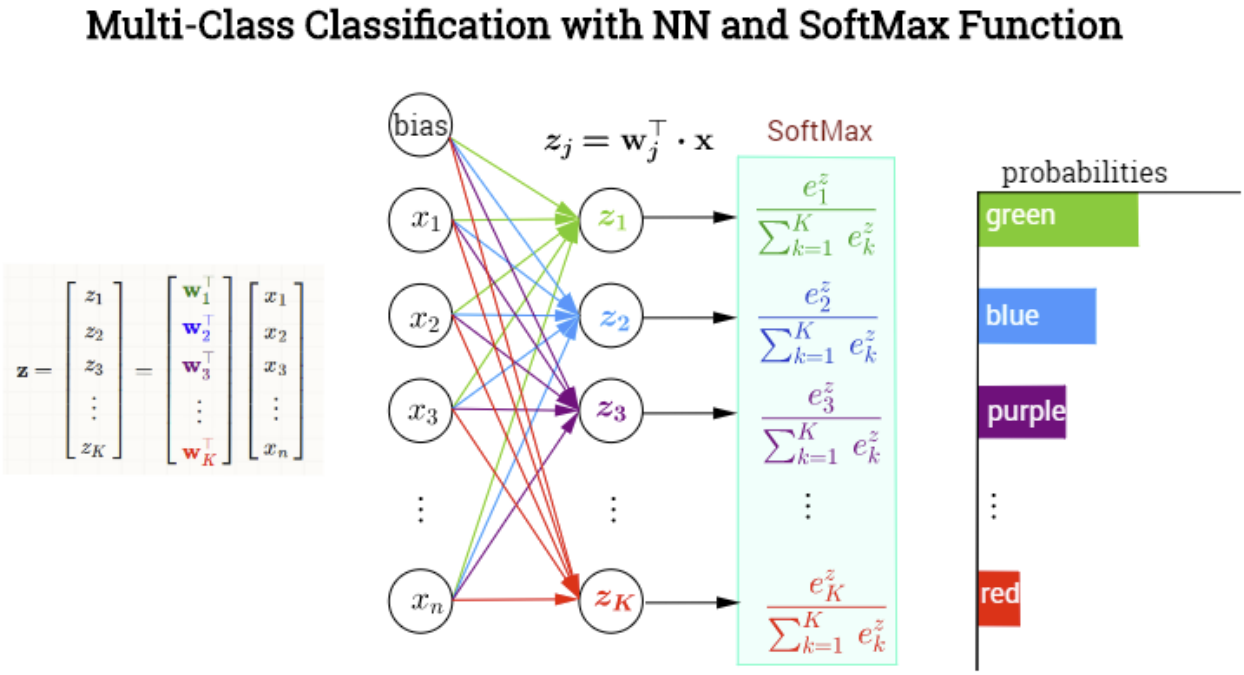
$$\mathbf{h_t} = \tanh(\mathbf{U x_t} + \mathbf{W h_{t-1}})$$

$$\mathbf{\hat{y}_t} = softmax(\mathbf{V h_t}) = softmax(\mathbf{o_t})$$

2.3 Softmax

Softmax là một hàm phi tuyến thường được dùng ở các neuron output để nó phân loại dữ liệu đầu vào thành vào các nhóm khác nhau. Hay nói cách khác, khi áp dụng cho trường hợp của RNN, hàm *softmax* nhận vào o_t, và trả về các giá trị z_{t_i} thể hiện xác xuất giá trị o_t rơi vào lớp i (tích cực, tiêu cực, trung tính).

$$\mathbf{o} = \begin{bmatrix} o_1 \\ o_2 \\ \dots \\ o_n \end{bmatrix}, \; y = softmax(\mathbf{o}) = \begin{bmatrix} z_1 \\ z_2 \\ \dots \\ z_n \end{bmatrix}, \text{ trong đó } z_i = \frac{e^{o_i}}{\sum_{k=1}^{n} e^{o_k}}$$



Hình 4: Hàm softmax dùng trong bài toán phân loại ở mạng NN[5]

5.Nguồn: http://rinterested.github.io/statistics/softmax.html

2.4 Cross Entropy Loss

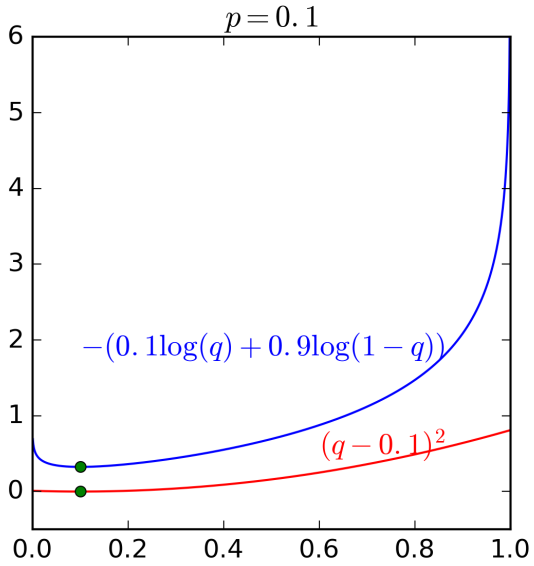
Cross-entropy loss hay còn gọi là log loss, dùng để đo error value trong các bài toán phân loại (classification). *Cross-entropy* trả về kết quả là một giá trị thực trong đoạn [0;1].

$$L = \sum_{t=1}^{N} \mathbf{H}(\mathbf{y_t}, \hat{\mathbf{y}}_\mathbf{t})$$

$$L = \sum_{t=1}^{N} \mathbf{y_t} \cdot \log(\hat{\mathbf{y}}_\mathbf{t})$$

Có 2 lý do để hàm này được dùng cho một bài toán phân loại là:

- Hàm này đạt cực tiểu khi $\hat{\mathbf{y}}_\mathbf{t} = \mathbf{y_t}$
- Khi $\hat{y}_{ti}$ càng xa y_{ti} thì giá trị của hàm loss sẽ khá lớn và tăng khá nhanh. Do đó, khi dùng hàm này thì việc tối ưu hóa sẽ giúp cho ta cho ra các output gần với giá trị thực hơn.



Hình 5: Đồ thị hàm cross entropy $L = p\log(q) + (1-p)\log(1-q)$ (đường xanh) và đồ thị hàm bình phương khoảng cách $L = (q-p)^2$ (đường đỏ) với giá trị p=0.1(machine learning cơ bản[1]). Có thể thấy càng ra xa giá trị 0.1 thì hàm cross entropy tăng nhanh hơn hàm bình phương khoảng cách.

Khi chỉ cần phân loại vào 2 lớp, ta có thể dùng binary cross-entropy:

$$L = -(y\log(\hat{y}_t) + (1-y)\log(1-\hat{y}_t))$$

Bài toán **Sentiment Analysis** mà chúng tôi trình bày ở đây chỉ có một output $\hat{\mathbf{y}}_\mathbf{N}$ sau khi đã tính toán với các input $\mathbf{x_1}$, $\mathbf{x_2}$,..., $\mathbf{x_N}$. Do đó, hàm cross entropy mà chúng tôi sẽ dùng trong lúc tính toán là

$$L = \mathbf{y} \cdot \log(\hat{\mathbf{y}}_\mathbf{N})$$

2.5 Backpropagation ở RNN

Trong khi train, để cho mạng có thể "học" từ những dữ liệu train thì nó cần phải tối ưu hóa hàm Loss bằng cách điều chỉnh các tham số bằng phương pháp *gradient descent*. Để sử dụng được phương pháp đó thì ta cần phải tính đạo hàm của hàm Loss theo các tham số trong mỗi lần chạy. Vì vậy, bên cạnh quá trình lan truyền tiến (*feedforward*) thì ta cần một quá trình lan truyền ngược (*backpropagation*) để tính đạo hàm từ output ngược về input. Đối với RNN, quá trình tính toán lan truyền ngược được thực hiện như sau.

- $\frac{\partial L}{\partial \mathbf{V}}$

$$\mathbf{V} = \begin{bmatrix} \mathbf{V_1} & \mathbf{V_2} & ... & \mathbf{V_d} \end{bmatrix} = \begin{bmatrix} V_{11} & V_{12} & ... & V_{1d} \\ V_{21} & V_{22} & ... & V_{2d} \\ & ... & & \\ V_{c1} & V_{c2} & ... & V_{cd} \end{bmatrix}$$

Xét một phần tử bất kì V_{ij} của $\mathbf{V}$ ở dòng i và cột j

$$\frac{\partial L}{\partial V_{ij}} = \left(\frac{\partial \hat{\mathbf{y}}_\mathbf{N}}{\partial V_{ij}}\right)^T \frac{\partial L}{\partial \hat{\mathbf{y}}_\mathbf{N}} = \left(\frac{\partial o_{Ni}}{\partial V_{ij}} \frac{\partial \hat{\mathbf{y}}_\mathbf{N}}{\partial o_{Ni}}\right)^T \frac{\partial L}{\partial \hat{\mathbf{y}}_\mathbf{N}} = \frac{\partial o_{Ni}}{\partial V_{ij}} \left(\frac{\partial \hat{\mathbf{y}}_\mathbf{N}}{\partial o_{Ni}}\right)^T \frac{\partial L}{\partial \hat{\mathbf{y}}_\mathbf{N}}$$

Trong đó, với đạo hàm thứ ba

$$L = -\mathbf{y} \cdot \log(\hat{\mathbf{y}}_\mathbf{N}) = -[y_1 \log(\hat{y}_{N1}) + y_2 \log(\hat{y}_{N2}) + ... + y_c \log(\hat{y}_{Nc})]$$

$$\Rightarrow \frac{\partial L}{\partial \hat{\mathbf{y}}_\mathbf{N}} = \begin{bmatrix} \frac{\partial L}{\partial \hat{y}_{N1}} \\ \frac{\partial L}{\partial \hat{y}_{N2}} \\ ... \\ \frac{\partial L}{\partial \hat{y}_{Nc}} \end{bmatrix} = \begin{bmatrix} -\frac{y_1}{\hat{y}_{N1}} \\ -\frac{y_2}{\hat{y}_{N2}} \\ ... \\ -\frac{y_c}{\hat{y}_{Nc}} \end{bmatrix}$$

Với đạo hàm thứ hai

$$\hat{\mathbf{y}}_\mathbf{N} = \begin{bmatrix} \hat{y}_{N1} \\ \hat{y}_{N2} \\ ... \\ \hat{y}_{Nc} \end{bmatrix} = \begin{bmatrix} \frac{e^{o_{N1}}}{S} \\ \frac{e^{o_{N2}}}{S} \\ ... \\ \frac{e^{o_{Nc}}}{S} \end{bmatrix}, \text{ với } S = \sum_{k=1}^{c} e^{o_{Nk}}$$

$$\Rightarrow \frac{\partial \hat{\mathbf{y}}_\mathbf{N}}{\partial o_{Ni}} = \begin{bmatrix} \frac{\partial \hat{y}_{N1}}{\partial o_{Ni}} \\ \frac{\partial \hat{y}_{N2}}{\partial o_{Ni}} \\ \dots \\ \frac{\partial \hat{y}_{Ni}}{\partial o_{Ni}} \\ \dots \\ \frac{\partial \hat{y}_{Nc}}{\partial o_{Ni}} \end{bmatrix} = \begin{bmatrix} -\hat{y}_{N1}\hat{y}_{Ni} \\ -\hat{y}_{N2}\hat{y}_{Ni} \\ \dots \\ \hat{y}_{Ni} - \hat{y}_{Ni}^2 \\ \dots \\ -\hat{y}_{Nc}\hat{y}_{Ni} \end{bmatrix}$$

Suy ra

$$\left(\frac{\partial \hat{\mathbf{y}}_\mathbf{N}}{\partial o_{Ni}}\right)^T \frac{\partial L}{\partial \hat{\mathbf{y}}_\mathbf{N}} = \begin{bmatrix} -\hat{y}_{N1}\hat{y}_{Ni} & -\hat{y}_{N2}\hat{y}_{Ni} & \dots & \hat{y}_{Ni} - \hat{y}_{Ni}^2 & \dots & -\hat{y}_{Nc}\hat{y}_{Ni} \end{bmatrix} \begin{bmatrix} -\frac{y_1}{\hat{y}_{N1}} \\ -\frac{y_2}{\hat{y}_{N2}} \\ \dots \\ -\frac{y_c}{\hat{y}_{Nc}} \end{bmatrix}$$

$$\left(\frac{\partial \hat{\mathbf{y}}_\mathbf{N}}{\partial o_{Ni}}\right)^T \frac{\partial L}{\partial \hat{\mathbf{y}}_\mathbf{N}} = y_1\hat{y}_{Ni} + y_2\hat{y}_{Ni} + \dots + y_i\hat{y}_{Ni} - y_i + \dots + y_c\hat{y}_{Ni} = \left(\sum_{k=1}^{c} y_k\right)\hat{y}_{Ni} - y_i$$

Vì các phần tử y_1, y_2,..., y_c là các giá trị xác suất để output được xếp vào các lớp phân loại nên do đó $\sum_{k=1}^{c} y_k = 1$. Vì vậy,

$$\left(\frac{\partial \hat{\mathbf{y}}_\mathbf{N}}{\partial o_{Ni}}\right)^T \frac{\partial L}{\partial \hat{\mathbf{y}}_\mathbf{N}} = \hat{y}_{Ni} - y_i$$

Với đạo hàm thứ nhất

$$o_{Ni} = \sum_{k=1}^{d} V_{ik} h_{Nk}$$

$$\Rightarrow \frac{\partial o_{Ni}}{\partial V_{ij}} = h_{Nj}$$

Từ đó ta suy ra

$$\frac{\partial L}{\partial V_{ij}} = \frac{\partial o_{Ni}}{\partial V_{ij}} \left(\frac{\partial \hat{\mathbf{y}}_\mathbf{N}}{\partial o_{Ni}}\right)^T \frac{\partial L}{\partial \hat{\mathbf{y}}_\mathbf{N}} = (\hat{y}_{Ni} - y_i) h_{Nj}$$

Tính toán tương tự như trên với các phần tử thuộc cột j của $\mathbf{V}$, ta được

$$\frac{\partial L}{\partial \mathbf{V_j}} = \begin{bmatrix} \frac{\partial L}{\partial V_{1j}} \\ \frac{\partial L}{\partial V_{2j}} \\ \dots \\ \frac{\partial L}{\partial V_{cj}} \end{bmatrix} = \begin{bmatrix} (\hat{y}_{N1} - y_1) h_{Nj} \\ (\hat{y}_{N2} - y_2) h_{Nj} \\ \dots \\ (\hat{y}_{Nc} - y_c) h_{Nj} \end{bmatrix} = (\hat{\mathbf{y}}_\mathbf{N} - \mathbf{y}) h_{Nj}$$

Và ta tiếp tục tính đạo hàm cho các thành phần $\mathbf{V_1}$, $\mathbf{V_2}$,..., $\mathbf{V_d}$, ta được

$$\frac{\partial L}{\partial \mathbf{V}} = \begin{bmatrix} \frac{\partial L}{\partial \mathbf{V_1}} & \frac{\partial L}{\partial \mathbf{V_2}} & ... & \frac{\partial L}{\partial \mathbf{V_c}} \end{bmatrix} = \begin{bmatrix} h_{N1}(\hat{\mathbf{y}}_\mathbf{N} - \mathbf{y}) & h_{N2}(\hat{\mathbf{y}}_\mathbf{N} - \mathbf{y}) & ... & h_{Nc}(\hat{\mathbf{y}}_\mathbf{N} - \mathbf{y}) \end{bmatrix}$$

$$\frac{\partial L}{\partial \mathbf{V}} = (\hat{\mathbf{y}}_\mathbf{N} - \mathbf{y}){\mathbf{h}_\mathbf{N}}^T$$

hay

$$\frac{\partial L}{\partial \mathbf{V}} = (\hat{\mathbf{y}}_\mathbf{N} - \mathbf{y}) \otimes \mathbf{h}_\mathbf{N}$$

- $\frac{\partial L}{\partial \mathbf{U}}$, $\frac{\partial L}{\partial \mathbf{W}}$
  Sử dụng quy tắc chuỗi (*chain rule*), ta có:

$$\frac{\partial L}{\partial \mathbf{U}} = \frac{\partial \mathbf{h_N}}{\partial \mathbf{U}} \frac{\partial \hat{\mathbf{y}}_\mathbf{N}}{\partial \mathbf{h_N}} \frac{\partial L}{\partial \hat{\mathbf{y}}_\mathbf{N}}$$

  Hai nhân tử đạo hàm $\frac{\partial \hat{\mathbf{y}}_\mathbf{N}}{\partial \mathbf{h_N}}$, $\frac{\partial L}{\partial \hat{\mathbf{y}}_\mathbf{N}}$ có thể được tính toán tương tự như phần trên, còn nhân tử $\frac{\partial \mathbf{h_N}}{\partial \mathbf{U}}$ có thể được tính bằng *chain rule* khi biết $\mathbf{h_N}$ là hàm của $\mathbf{h_1}$, $\mathbf{h_2}$,..., $\mathbf{h_{N-1}}$

$$\frac{\partial L}{\partial \mathbf{U}} = \sum_{k=1}^{N-1} \frac{\partial \mathbf{h_k}}{\partial \mathbf{U}} \frac{\partial \mathbf{h_N}}{\partial \mathbf{h_k}} \frac{\partial \hat{\mathbf{y}}_\mathbf{N}}{\partial \mathbf{h_N}} \frac{\partial L}{\partial \hat{\mathbf{y}}_\mathbf{N}}$$

  Tới lượt $\frac{\partial \mathbf{h_N}}{\partial \mathbf{h_k}}$ thì cũng phải tính bằng *chain rule* nếu như $k \neq N-1$. Khi dùng *chain rule* thì $\frac{\partial \mathbf{h_N}}{\partial \mathbf{h_k}}$ là tích của các thành phần $\frac{\partial \mathbf{h_N}}{\partial \mathbf{h_{N-1}}}$, $\frac{\partial \mathbf{h_{N-1}}}{\partial \mathbf{h_{N-2}}}$,..., $\frac{\partial \mathbf{h_{k+1}}}{\partial \mathbf{h_k}}$.

$$\frac{\partial L}{\partial \mathbf{U}} = \sum_{k=1}^{N-1} \frac{\partial \mathbf{h_k}}{\partial \mathbf{U}} \left(\prod_{j=k}^{N-1} \frac{\partial \mathbf{h_{j+1}}}{\partial \mathbf{h_j}} \right) \frac{\partial \hat{\mathbf{y}}_\mathbf{N}}{\partial \mathbf{h_N}} \frac{\partial L}{\partial \hat{\mathbf{y}}_\mathbf{N}}$$

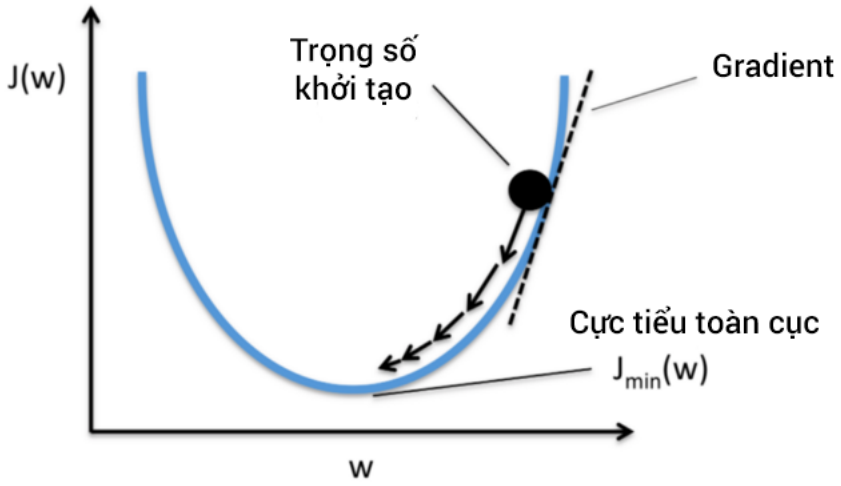
  Tiếp theo $\frac{\partial L}{\partial \mathbf{W}}$ cũng được tính tương tự như trên.

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{k=1}^{N-1} \frac{\partial \mathbf{h_k}}{\partial \mathbf{W}} \left(\prod_{j=k}^{N-1} \frac{\partial \mathbf{h_{j+1}}}{\partial \mathbf{h_j}} \right) \frac{\partial \hat{\mathbf{y}}_\mathbf{N}}{\partial \mathbf{h_N}} \frac{\partial L}{\partial \hat{\mathbf{y}}_\mathbf{N}}$$

2.6 Gradient Descent

Gradient descent là một trong những thuật toán phổ biến nhất sử dụng trong việc train các mô hình học máy. Như ta đã biết, một mô hình học máy chủ yếu là bao gồm các tham số và một hàm mất mát để tính toán độ hiệu quả của tham số đó. Mạng của chúng ta học bằng cách điều chỉnh các tham số để tối thiểu hàm mất mát.

Gradient descent hoạt động bằng cách tối ưu hoá dần dần một bộ tham số ban đầu. Để tối ưu một bộ trọng số, ta phải tìm cách biểu thị mối quan hệ của hàm mất mát đối với các trọng số bằng cách tìm gradient của hàm mất mát như ta đã thực hiện ở phần trên. Gradient sẽ cho ta biết chiều tăng của hàm số, vì vậy nên nếu muốn tối thiểu hàm số cần phải đi theo chiều ngược lại. Đó cũng chính là lí do mà thuật toán này có tên gọi là *Gradient descent* (đi ngược Gradient). Thuật toán *Gradient*



Hình 6: Mô tả thuật toán Gradient Descent. Nguồn: Medium

descent được dùng như sau

$$W = W - \alpha \frac{\partial L}{\partial W}$$

Với W là tham số, L là hàm loss và α là tốc độ học (*learning rate*) với giá trị do ta chọn.

2.7 Vanishing/Exploding Gradient

Trong việc xử lí ngôn ngữ, dữ liệu văn bản nói chung, thường sẽ gặp trường hợp đòi mạng phải ghi nhớ những thông tin rất lâu về trước- hay nói cách khác là xử lí các phụ thuộc xa *(long-term dependency)*. Mặc dù RNN được thiết kế để có thể nhớ được những thông tin trước đó, nhưng trên thực tế, điều này lại không đúng. Vấn đề này gọi là *vanishing gradient*. Từ những năm đầu của 1990, *vanishing gradient* trở thành một bước cản lớn đối với sự phát triển của RNN.
Như ta đã biết, gradient của hàm mất mát biểu diễn cách mà sự thay đổi của mọi trọng số dẫn đến sự thay đổi của hàm mất mát, từ đó cho ta biết hướng đi để có thể tối thiểu nó. Nếu ta không biết được hướng đi, cũng có nghĩa là ta không biết cách để cập nhật trọng số sao cho hàm mất mát giảm dần và mạng của ta sẽ dừng học.
Vanishing gradient là hiện tượng khi mà gradient trở nên rất nhỏ khiến việc cập nhật trọng số chỉ khiến trọng số thay đổi một cách không đáng kể, do đó cũng sẽ không làm cho hàm mất mát giảm đi. Ta cùng xem lại đạo hàm của hàm mất mát tại thời điểm t theo trọng số U của mạng RNN:

$$\frac{\partial L}{\partial \mathbf{U}} = \sum_{k=1}^{N-1} \frac{\partial \mathbf{h_k}}{\partial \mathbf{U}} \left(\prod_{j=k}^{N-1} \frac{\partial \mathbf{h_{j+1}}}{\partial \mathbf{h_j}} \right) \frac{\partial \hat{\mathbf{y}}_\mathbf{N}}{\partial \mathbf{h_N}} \frac{\partial L}{\partial \hat{\mathbf{y}}_\mathbf{N}}$$

Giả sử ta sử dụng hàm kích hoạt là hàm *tanh*, $\mathbf{h_t} = \tanh(\mathbf{U x_t} + \mathbf{W h_{t-1}})$

$$\frac{\partial \mathbf{h_{t+1}}}{\partial \mathbf{h_t}} = \mathbf{W}(1 - \mathbf{h_t} \odot \mathbf{h_t})$$

Hàm *tanh* giới hạn giá trị của các phần tử trong $\mathbf{h_t}$ vào khoảng [-1;1]. Khi mà các đạo hàm như trên bé hơn 1, nếu N càng xa k thì dẫn tới khi tính đạo hàm của $\mathbf{h_N}$ theo $\mathbf{h_k}$ sẽ phải tính một cái tích rất dài của các số bé hơn 1 kéo theo đạo hàm của L theo $\mathbf{U}$ sẽ càng tiến tới $\mathbf{0}$

$$\frac{\partial \mathbf{h_N}}{\partial \mathbf{h_k}} \to \mathbf{0}$$
$$\Leftrightarrow \frac{\partial L}{\partial \mathbf{U}} \to \mathbf{0}$$

Đây được gọi là hiện tượng *vanishing gradient* khi mà một thành phần trong gradient của hàm mất mát mà ta tính ra đối với biến $\mathbf{h_k}$ cách xa $\mathbf{h_N}$ rất bé. Vì vậy trong các trường hợp input là một chuỗi rất dài thì khi chạy tới một phần tử càng xa thì mạng sẽ có xu hướng "quên " đi những phần tử trước đó. *Exploding gradient* ngược lại với *vanishing gradient*, thay vì giảm dần thì *exploding gradient* là hiện tượng gradient tăng dần và trở nên rất lớn. *Vanishing gradient* và *exploding gradient* là những vấn đề chung của mạng NN, hầu hết RNN thường phải đối mặt với vấn đề *vanishing gradient*. Để khắc phục nhược điểm đó người ta đã phát triển một mô hình khác tương tự RNN là *Long Short-Term Memory* (LSTM)

3 Long Short-Term Memory(LSTM)

3.1 Giới thiệu

Những vấn đề *vanishing gradient, exploding gradient, short-term memory* khiến mạng RNN gặp khó khăn trong việc mô hình các câu dài. Do vậy mà người ta đã tạo ra một cấu trúc mạng mới gọi là Kí ức ngắn hạn-dài hạn (*Long Short-term Memory* – LSTM). LSTM được chỉnh sửa từ RNN, nhưng thay vì chỉ đơn giản sử dụng trạng thái ẩn ở thời điểm $t-1$ là h_{t-1} để tính trạng thái ẩn h_t thì LSTM thực hiện thêm một loạt các thao tác tính toán khác để nó có khả năng ghi nhớ tốt hơn.

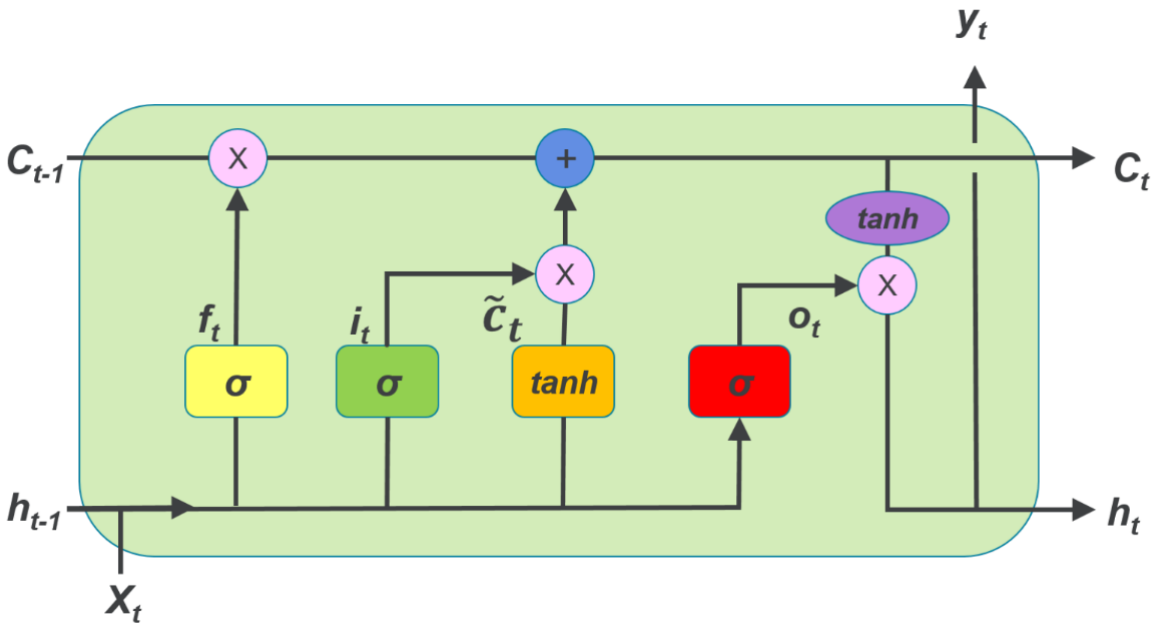
3.2 Cấu trúc

Trong mô hình này ta thay đổi cách tính *hidden state* bằng cách thêm 3 cổng $\mathbf{i_t}$, $\mathbf{f_t}$, $\mathbf{o_t}$, 1 biến $\widetilde{\mathbf{C}}_\mathbf{t}$ và một biến mới $\mathbf{C_t}$.

Cổng input kiểm tra xem có thông tin nào đáng để có thể tác động lên bộ nhớ cũ và cập nhật bộ nhớ với các thông tin mới đó.

$$\mathbf{i_t} = \sigma(\mathbf{U}^i \mathbf{x_t} + \mathbf{W^i h_{t-1}})$$

Cổng forget xem xét xem thông tin nào trong bộ nhớ cần được giữ lại cũng như thông tin nào cần được quên đi. Nó là một hàm *sigmoid* đóng vai trò là một bộ lọc



Hình 7: Sơ đồ cấu trúc của LSTM

thông tin. Nếu đóng cổng này (trả về 0) sẽ không có thông tin nào được giữ lại, nếu mở cổng này (trả về 1) thì mọi thông tin đều được giữ lại.

$$\mathbf{f_t} = \sigma(\mathbf{U}^f\mathbf{x_t} + \mathbf{W^f h_{t-1}})$$

Cổng output quyết định xem thông tin nào sẽ được output.

$$\mathbf{o_t} = \sigma(\mathbf{U}^o\mathbf{x_t} + \mathbf{W^o h_{t-1}})$$

Các công thức tính đều giống nhau chỉ khác nhau ở các tham số. Các cổng này sử dụng hàm *sigmoid* để co giá trị của vector xuống trong đoạn [0,1]. Và ta nhân *element-wise* các cổng với một vector khác sẽ cho ta biết nên cho bao nhiêu phần của vector đó được đi qua.

$\widetilde{\mathbf{C}}_\mathbf{t}$ là "ứng cử viên" cho *hidden state* như ta thấy là cách tính của nó là cách tính *hidden state* trong mạng RNN truyền thống.

$$\widetilde{\mathbf{C_t}} = \tanh(\mathbf{U}^g\mathbf{x_t} + \mathbf{W^g h_{t-1}})$$

$\mathbf{C_t}$ đóng vai trò là trí nhớ RNN. Tính dựa vào trí nhớ trước đó $\mathbf{C_{t-1}}$, cổng forget, hidden state tiềm năng và cổng input.

$$\mathbf{C_t} = \mathbf{C_{t-1}} \odot \mathbf{f_t} + \widetilde{\mathbf{C}}_\mathbf{t} \odot \mathbf{i_t}$$

$\mathbf{h_t}$ là hidden state ở thời điểm t.

$$\mathbf{h_t} = \tanh(\mathbf{C_t}) \odot \mathbf{o_t}$$

Bên cạnh đó để cho thuận lợi trong tính toán ta quy ước các biến sau.

$$\mathbf{z_t} = \begin{bmatrix} \hat{\mathbf{i}}_\mathbf{t} \\ \hat{\mathbf{f}}_\mathbf{t} \\ \hat{\mathbf{o}}_\mathbf{t} \\ \widehat{\widetilde{\mathbf{C}}}_\mathbf{t} \end{bmatrix} = \begin{bmatrix} \mathbf{U}^i\mathbf{x_t} + \mathbf{W^i h_{t-1}} \\ \mathbf{U}^f\mathbf{x_t} + \mathbf{W^f h_{t-1}} \\ \mathbf{U}^o\mathbf{x_t} + \mathbf{W^o h_{t-1}} \\ \mathbf{U}^c\mathbf{x_t} + \mathbf{W^c h_{t-1}} \end{bmatrix}, \mathbf{U} = \begin{bmatrix} \mathbf{U^i} \\ \mathbf{U^f} \\ \mathbf{U^o} \\ \mathbf{U^c} \end{bmatrix}, \mathbf{W} = \begin{bmatrix} \mathbf{W^i} \\ \mathbf{W^f} \\ \mathbf{W^o} \\ \mathbf{W^c} \end{bmatrix}$$

$$\Leftrightarrow \mathbf{z_t} = \mathbf{U x_t} + \mathbf{W h_{t-1}}$$

Trong đó $\mathbf{z_t}$ là một biến đại diện cho các cổng khi ta chỉ lấy những thành phần trong các hàm *activation*. $\mathbf{W}$, $\mathbf{U}$ là ma trận chứa các tham số mà mạng dùng trong suốt một lần chạy.

3.3 Backpropagation ở LSTM

Ở phần này ta trình bày quá trình *backpropagation* để tính các đạo hàm riêng của hàm loss với các ma trận tham số. Để cho thuận tiện, ta đặt chung kí hiệu cho đạo hàm riêng của hàm L theo các ma trận là

$$\delta\mathbf{a_t} = \frac{\partial L}{\partial \mathbf{a_t}}$$

Với $\mathbf{a_t}$ là một ma trận bất kì tại thứ tự t

- $\delta\mathbf{o_t}$

$$\frac{\partial L}{\partial o_{tk}} = \frac{\partial L}{\partial h_{tk}}\frac{\partial h_{tk}}{\partial o_{tk}} = \delta h_{tk}\tanh(C_{tk})$$

Suy ra

$$\delta\mathbf{o_t} = \delta\mathbf{h_t} \odot \tanh(\mathbf{C_t})$$

- $\delta\mathbf{C_t}$

$$\frac{\partial L}{\partial C_{tk}} = \frac{\partial L}{\partial h_{tk}}\frac{\partial h_{tk}}{\partial C_{tk}} + \frac{\partial L}{\partial C_{(t+1)k}}\frac{\partial C_{(t+1)k}}{\partial C_{tk}}$$

$$\frac{\partial L}{\partial C_{tk}} = \delta h_{tk}(1 - \tanh^2(C_{tk}))o_{tk} + \delta C_{(t+1)k} f_{(t+1)k}$$

Suy ra

$$\delta\mathbf{C_t} = \delta\mathbf{h_t} \odot [1 - \tanh^2(\mathbf{C_t})] \odot \mathbf{o_t} + \delta\mathbf{C_{t+1}} \odot \mathbf{f_{t+1}}$$

- $\delta\mathbf{i_t}$

$$\frac{\partial L}{\partial i_{tk}} = \frac{\partial L}{\partial C_{tk}}\frac{\partial C_{tk}}{\partial i_{tk}} = \delta C_{tk}\widetilde{C}_{tk}$$

Suy ra

$$\delta\mathbf{i_t} = \delta\mathbf{C_t} \odot \widetilde{\mathbf{C}}_\mathbf{t}$$

- $\delta\mathbf{f_t}$

$$\frac{\partial L}{\partial f_{tk}} = \frac{\partial L}{\partial C_{tk}}\frac{\partial C_{tk}}{\partial f_{tk}} = \delta C_{tk} C_{(t-1)k}$$

Suy ra

$$\delta\mathbf{f_t} = \delta\mathbf{C_t} \odot \mathbf{C_{t-1}}$$

- $\delta\widetilde{\mathbf{C}}_\mathbf{t}$

$$\frac{\partial L}{\partial \widetilde{C}_{tk}} = \frac{\partial L}{\partial C_{tk}}\frac{\partial C_{tk}}{\partial \widetilde{C}_{tk}} = \delta C_{tk} i_{tk}$$

Suy ra

$$\delta\widetilde{\mathbf{C}}_\mathbf{t} = \delta\mathbf{C_t} \odot \mathbf{i_t}$$

- $\delta\hat{\mathbf{i}}_\mathbf{t}$, $\delta\hat{\mathbf{f}}_\mathbf{t}$, $\delta\hat{\mathbf{o}}_\mathbf{t}$

$$\frac{\partial L}{\partial \hat{i}_{tk}} = \frac{\partial L}{\partial i_{tk}}\frac{\partial i_{tk}}{\partial \hat{i}_{tk}} = \delta i_{tk}\sigma'(\hat{i}_{tk})$$

$$\frac{\partial L}{\partial \hat{i}_{tk}} = \delta i_{tk}\sigma(\hat{i}_{tk})[1-\sigma(\hat{i}_{tk})] = \delta i_{tk} i_{tk}(1-i_{tk})$$

Suy ra

$$\delta\hat{\mathbf{i}}_\mathbf{t} = \delta\mathbf{i_t} \odot \mathbf{i_t} \odot (1-\mathbf{i_t})$$

Tương tự

$$\delta\hat{\mathbf{f}}_\mathbf{t} = \delta\mathbf{f_t} \odot \mathbf{f_t} \odot (1-\mathbf{f_t})$$

$$\delta\hat{\mathbf{o}}_\mathbf{t} = \delta\mathbf{o_t} \odot \mathbf{o_t} \odot (1-\mathbf{o_t})$$

- $\delta\widehat{\widetilde{\mathbf{C}}}_\mathbf{t}$

$$\frac{\partial L}{\partial \widehat{\widetilde{C}}_{tk}} = \frac{\partial L}{\partial \widetilde{C}_{tk}}\frac{\partial \widetilde{C}_{tk}}{\partial \widehat{\widetilde{C}}_{tk}} = \delta\widetilde{C}_{tk}\tanh'(\widehat{\widetilde{C}}_{tk})$$

$$\frac{\partial L}{\partial \widehat{\widetilde{C}}_{tk}} = \delta\widetilde{C}_{tk}[1-\tanh^2(\widehat{\widetilde{C}}_{tk})] = \delta\widetilde{C}_{tk}(1-\widetilde{C}_{tk}^2)$$

Suy ra

$$\delta\widehat{\widetilde{\mathbf{C}}}_\mathbf{t} = \delta\widetilde{\mathbf{C}}_\mathbf{t} \odot (1-\widetilde{\mathbf{C}}_\mathbf{t} \odot \widetilde{\mathbf{C}}_\mathbf{t})$$

- $\delta\mathbf{z_t}$

$$\delta\mathbf{z_t} = \begin{bmatrix} \delta\hat{\mathbf{i}}_\mathbf{t} \\ \delta\hat{\mathbf{f}}_\mathbf{t} \\ \delta\hat{\mathbf{o}}_\mathbf{t} \\ \delta\widehat{\widetilde{\mathbf{C}}}_\mathbf{t} \end{bmatrix} = \begin{bmatrix} \delta\mathbf{i_t} \odot \mathbf{i_t} \odot (1-\mathbf{i_t}) \\ \delta\mathbf{f_t} \odot \mathbf{f_t} \odot (1-\mathbf{f_t}) \\ \delta\mathbf{o_t} \odot \mathbf{o_t} \odot (1-\mathbf{o_t}) \\ \delta\widetilde{\mathbf{C}}_\mathbf{t} \odot (1-\widetilde{\mathbf{C}}_\mathbf{t} \odot \widetilde{\mathbf{C}}_\mathbf{t}) \end{bmatrix}$$

- $\delta\mathbf{h_{t-1}}$

$$\delta\mathbf{h_{t-1}} = \left(\frac{\partial \mathbf{z_t}}{\partial \mathbf{h_{t-1}}}\right)^T \frac{\partial L}{\partial \mathbf{z_t}} = (\mathbf{W}^T)^T \delta\mathbf{z_t} = \mathbf{W}\delta\mathbf{z_t}$$

$\delta\mathbf{h_{t-1}}$ được tính để phục vụ cho việc tính tiếp $\delta\mathbf{z_{t-1}}$. Để bắt đầu quá trình *backpropagation* thì đầu tiên ta cần tính $\delta\mathbf{h_N}$ (có thể dễ dàng tính được từ output). Cuối cùng sau khi đã tính xong các vector $\delta\mathbf{z_1}$, $\delta\mathbf{z_2}$,..., $\delta\mathbf{z_N}$ thì ta có thể bắt tay vào tính $\delta\mathbf{W}$, $\delta\mathbf{U}$

- $\delta\mathbf{U}$, $\delta\mathbf{W}$

$$\delta\mathbf{U} = \sum_{t=1}^{N} \frac{\partial L}{\partial \mathbf{z_t}} \left(\frac{\partial \mathbf{z_t}}{\partial \mathbf{U}}\right)^T = \sum_{t=1}^{N} \delta\mathbf{z_t}\mathbf{x_t}^T = \sum_{t=1}^{N} \delta\mathbf{z_t} \otimes \mathbf{x_t}$$

$$\delta\mathbf{W} = \sum_{t=2}^{N} \frac{\partial L}{\partial \mathbf{z_t}} \left(\frac{\partial \mathbf{z_t}}{\partial \mathbf{W}}\right)^T = \sum_{t=2}^{N} \delta\mathbf{z_t}\mathbf{h_{t-1}}^T = \sum_{t=2}^{N} \delta\mathbf{z_t} \otimes \mathbf{h_{t-1}}$$

Thành phần chính gây ra *vanishing gradient* trong RNN là $\frac{\partial h_{t+1}}{\partial h_t} = (1 - h_t^2)W$. Tương tự trong LSTM, ta quan tâm đến $\frac{c_t}{c_{t-1}} = f_t$. Do hàm f_t là một hàm sigmoid nên giá trị $0 < f_t < 1$ nên về cơ bản thì LSTM vẫn bị *vanishing gradient* nhưng bị ít hơn so với RNN. Hơn thế nữa, khi mang thông tin lên bộ nhớ thì ít khi cần phải quên giá trị bộ nhớ cũ nên $f_t \approx 1$, giúp ta chống được *vanishing gradient*.

4 Áp dụng

Mô tả bài toán

Sau khi tìm hiểu lý thuyết về hai mô hình RNN, LSTM ở các phần trên thì đến phần này chúng ta sẽ áp dụng mô hình vào một bài toán cụ thể và cũng là bài toán chính của dự án này (**Sentiment Analysis**). Như đã đề cập trong phần tổng quát **Sentiment Analysis** là một bài toán con của *Text Classification*. Nhiệm vụ của chúng ta là phải xây dựng một mô hình để có thể phân loại được một bình luận là tích cực(*positive*) hay tiêu cực (*negative*) dựa vào đánh giá của khách hàng.

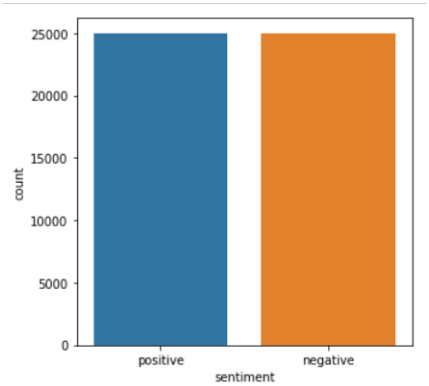
Tản mạn một xíu về ứng dụng của bài toán này trong cuộc sống: Giả sử bạn là chủ của một cửa hàng ăn uống và cửa hàng của bạn có một trang web riêng để khách hàng có thể review cho cửa hàng của bạn. Sau khi có được dữ liệu rồi thì dĩ nhiên bạn sẽ cần thống kê lại xem có bao nhiêu người bình luận tích cực, tiêu cực và bình luận tiêu cực thường đến từ vấn đề gì để từ đó bạn sẽ cố gắng khắc phục để đáp ứng nhu cầu của khách hàng... Vậy nên ta có thể thấy tính ứng dụng của bài toán này rất là cao.

Khi huấn luyện bất cứ mô hình nào, ta luôn cần phải có bộ dữ liệu. Bộ dữ liệu mà nhóm mình sử dụng là *"IMDB Dataset of 50K Movie Reviews"*, có thể được tìm thấy ở [15].

4.1 Xử lý dữ liệu

4.1.1 Phân tích dữ liệu

Đầu tiên chúng ta sẽ quan sát về số dữ liệu cũng như sự phân bố của dữ liệu vào các lớp là như thế nào:



Hình 8: Sự phân bố của dữ liệu vào các lớp

Biểu đồ trên cho ta thấy chúng ta có 50000 dữ liệu và được phân bố đều vào 2 lớp là *positive* và *negative* với 25000 dữ liệu mỗi lớp (Trong trường hợp số lượng dữ liệu bị mất cân bằng thì cần phải tiến hành cân bằng lại dữ liệu).

Thử quan sát 15 dòng dữ liệu đầu tiên xem như thế nào:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
5	Probably my all-time favorite movie, a story o...	positive
6	I sure would like to see a resurrection of a u...	positive
7	This show was an amazing, fresh & innovative i...	negative
8	Encouraged by the positive comments about this...	negative
9	If you like original gut wrenching laughter yo...	positive
10	Phil the Alien is one of those quirky films wh...	negative
11	I saw this movie when I was about 12 when it c...	negative
12	So im not a big fan of Boll's work but then ag...	negative
13	The cast played Shakespeare. Shakes...	negative
14	This a fantastic movie of three prisoners who ...	positive

Quan sát thấy do bộ dataset đơn giản nên các câu cũng khá dễ phân biệt. Còn một số câu có một số kí tự đặc biệt không mong muốn như *"

"* vì thế ta cần phải lọc lại các câu. Vấn đề này sẽ được nói trong phần tiền xử lý dữ liệu.

4.1.2 Tiền xử lý dữ liệu

Clean data
Theo như quan sát dữ liệu ở trên thì dữ liệu có một số câu chứa những kí tự đặc biệt dẫn đến việc chúng ta phải *clean data*, mục đích là để lọc những câu chứa kí tự đặc biệt, chứa biểu cảm không cần thiết cho việc phân loại.

Tokenizer
Tách câu ra thành các từ, sau đó ứng với mỗi từ được gán cho một số nguyên và trả mảng của các từ sau khi đã tách và chuyển thành số.

```
'the': 1,          'with': 16,
'and': 2,          'movie': 17,
'a': 3,            'but': 18,
'of': 4,           'film': 19,
'to': 5,           'on': 20,
'is': 6,           'not': 21,
'br': 7,           'you': 22,
'in': 8,           'are': 23,
'it': 9,           'his': 24,
'i': 10,           'have': 25,
'this': 11,        'be': 26,
'that': 12,        'one': 27,
'was': 13,         'he': 28,
'as': 14,          'all': 29,
'for': 15,         'at': 30,
```

Hình 9: 30 từ đầu tiên được tách và được biểu diễn bởi 1 số nguyên từ 1 tới 30

Pad Sequences
Việc thao tác với các chuỗi có độ dài khác nhau là vô cùng khó khăn. Ví dụ một câu có 1 từ và một câu có 100 từ, chính vì thế cần phải chuyển dữ liệu về dạng mảng *numpy 2D*, để *padding* giúp các chuỗi có độ dài bằng nhau và bằng một giá trị nào đó.

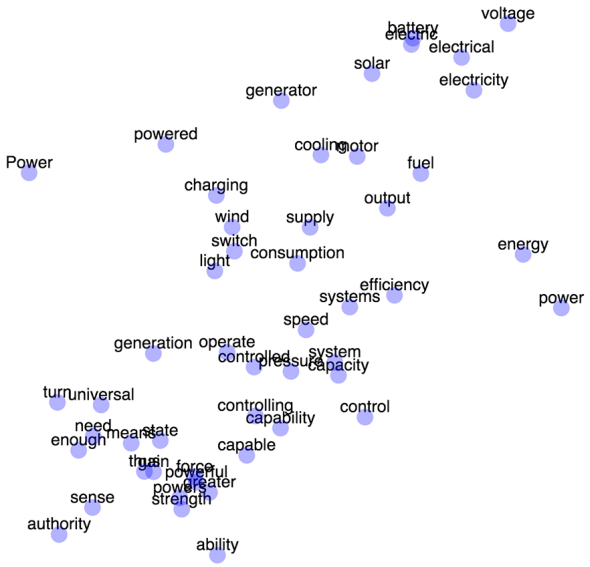
```
array([  538,    32,  4636,  2468,     4,     1,  1208,   117,    29,
           1,  7017,    25,  2970, 11964,     2,   391,    34, 16740,
           6,    21,   299,    20,     1,  4910,  7364,   538,     6,
         344,     5,   106, 25421,  8161, 42636, 14813,  5050,  7889,
        2453,     2,    51,    34, 48042,   327,  9106,  7365, 12486,
           2,  8697, 33191,    23,   110,   225,   243,     7,     7,
          10,    58,   131,     1,   280,  1324,     4,     1,   119,
           6,   693,     5,     1,   192,    12,     9,   269,   117,
          79,   276,   589,  3024,   834,   180,  1320,  4161,    15,
        2523,  1243,   834,  1443,   834,   887,  3184,   149,   954,
         183,     1,    86,   398,    10,   123,   210,  3241,    68,
          14,    34,  1637,     9,    13,  2239,    10,   413,   131,
          10,    13,  1592,    15,     9,    18,    14,    10,   287,
          51,    10,  1417,     3,  1280,    15,  3184,     2,   189,
       10168,     5,     1,   299,  2046,     4,  2150,   570,    21,
          39,   570,    18,  7658,  7154,  5010, 13497,    26,  2983,
          41,    15,     3, 25422,  6904, 13497,   504,    20,   642,
           2,    76,   243,    16,     9,    69,  7598,   651,   710,
        6904,   109,   662,    82,  1208, 19395,   693,     5,    65,
         574,     4,   920,  2021,    38,  1208,   559,   147,  3184,
          22,   200,   426,  3819,    16,    48,     6,  3314,   805,
        1603,    43,    22,    67,    76,     8,  1228,    16,   125,
        4103,   486], dtype=int32)
```

Hình 10: 1 câu sau khi đã được Pad Sequences, với độ dài là 200

Word Embedding

Chuyển từ hoặc cụm từ thành một vector mà giữa chúng có sự tương quan về mặt ý nghĩa. Hơn thế chúng ta còn có thể thực hiện trên những vector này các phép tính cộng, trừ.

Ví dụ: vector Hoàng Hậu + vector Nhà Vua - vector Phụ Nữ = vector Đàn Ông



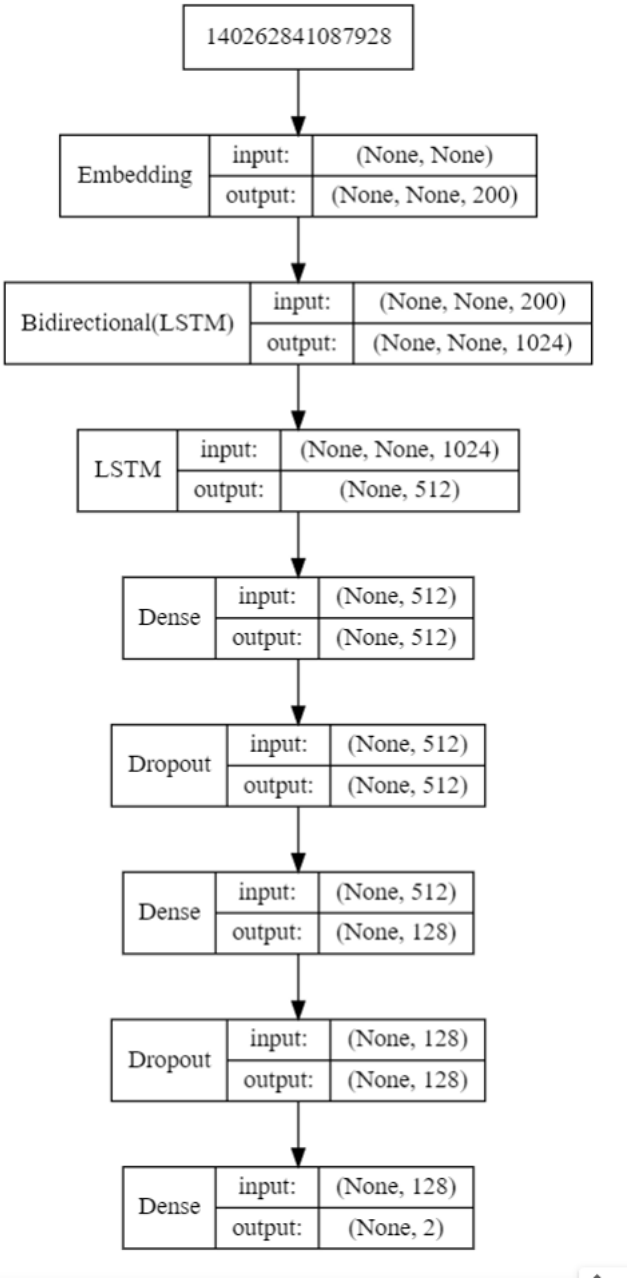
Hình 11: Ứng dụng Word Embedding vào dữ liệu của nhóm em

```
array([[[ 0.02026546,  0.04279691, -0.01832438, ...,  0.04720614,
         -0.0445037 , -0.02729278]],

       [[-0.03285909,  0.02380784,  0.03868261, ...,  0.00027636,
         -0.05689319,  0.01239457]],

       [[-0.01133746,  0.01544211, -0.04971611, ..., -0.05778742,
         -0.01324257, -0.0064427 ]],

       ...,

       [[ 0.02562615, -0.06312152,  0.02800502, ...,  0.04525236,
         -0.00221671,  0.01917973]],

       [[ 0.00522899, -0.07664026, -0.00823366, ..., -0.01624249,
          0.03075233,  0.07587465]],

       [[ 0.01351674, -0.01768329, -0.0253251 , ...,  0.02557267,
          0.02574865,  0.07379822]]], dtype=float32)
```

Hình 12: Một câu sau khi đã được Word Embedding thành dạng vector

4.2 Thiết kế mô hình

Mô hình mà nhóm mình thiết kế có dạng như sơ đồ sau:



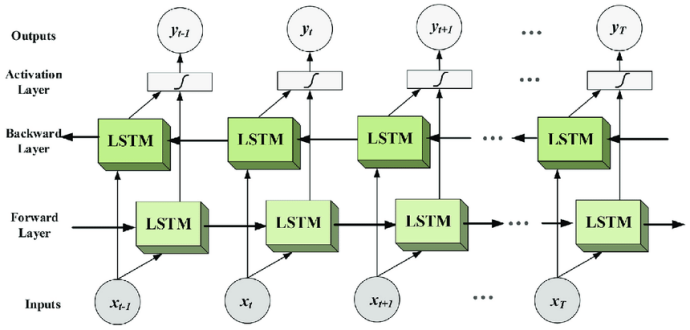
Hình 13: Mô hình

Thiết kế mô hình bằng Python

Ở đây nhóm mình thử 2 kiến trúc mạng RNN và LSTM vừa học được ở phần 2 để thiết kế mô hình để so sánh sự khác biệt. Nhìn chung để so sánh thì nhóm mình quyết định dùng chung mọi tham số, chỉ thay đổi kiến trúc của mô hình.

Giải thích mô hình:

- Lớp *Embedding* đầu tiên có nhiệm vụ embedding các câu thành các vector sau đó đưa vào lớp Bidirectional lồng LSTM.



Hình 14: Sơ đồ hoạt động của BLSTM

- Lớp Bidirectional lồng LSTM là một kiến trúc mở rộng của LSTM, nếu như bài toán phân loại văn bản trong LSTM, lớp phía sau phụ thuộc vào các lớp phía trước như vậy sẽ không có sự "công bằng" giữa lớp phía sau và lớp phía trước trong việc quyết định ý nghĩa của câu.
  Lấy ví dụ ta có 1 câu: "Em là trại sinh", như vậy thì từ quan trọng trong việc biểu diễn ý nghĩa của câu là "trại sinh". Nhìn chung trong một số trường hợp các từ ngữ phía trước lại không mang ý nghĩa quan trọng trong câu, trong khi những từ ngữ quan trọng lại ở cuối, nên chúng ta cần phải sử dụng một kiến trúc mạng khác để giải quyết vấn đề này. Bidirectional lồng LSTM có nghĩa là sẽ có 2 luồng LSTM chạy song song, 1 luồng chạy từ đầu tới cuối, luồng kia thì ngược lại. Như vậy việc quyết định ý nghĩa của câu sẽ trở nên "công bằng" hơn.

- Lớp *Fully Connected Layer* dùng để phân loại với activation cuối là hàm sigmoid.

- Dùng dropout để giảm overfitting do mô hình quá phức tạp để diễn giải dữ liệu.

- Loss Function là hàm Binary entropy và optimizer là Adam

Mô hình đầu tiên là mô hình RNN cơ bản, tức đầu ra của lớp phía trước sẽ được đưa ngược trở lại thành đầu vào của lớp tiếp theo:

```
def build_model():
    model = Sequential()
    model.add(Embedding(50000, 200))
    model.add(Bidirectional(SimpleRNN(512, return_sequences=True)))
    model.add(SimpleRNN(512))
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(2, activation='sigmoid'))
    model.compile(loss="binary_crossentropy",
                  optimizer=keras.optimizers.Adam(),
                  metrics=["acc"])
    return model
```

Độ chính xác của mô hình vào khoảng 0.6, loss khoảng 0.6.

Mô hình thứ 2 là mô hình LSTM:

```
def build_model():

    # create model
    model = Sequential()
    model.add(Embedding(50000, 200))
    model.add(Bidirectional(LSTM(512, return_sequences=True)))
    model.add(LSTM(512))
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(2, activation='sigmoid'))

    # compile model
    model.compile(loss="binary_crossentropy",
                  optimizer=keras.optimizers.Adam(),
                  metrics=["acc"])
    return model
```

Độ chính xác của mô hình gần đạt 0.9, loss gần tiến tới 0.3 trên tập validation.

```
15000/15000 [==============================] - 177s 12ms/step
[0.3065265076160431, 0.8753000000317891]
```

Như vậy có thể thấy, ở bài toán **Sentiment Analysis** này LSTM đã phát huy khả năng của mình và đạt hiệu quả cao. Ngược lại RNN đạt được kết quả khá tệ.

Ta có thể lý giải cho việc này như sau:
Chính vấn đề *Vanishing Gradient* của RNN mà nhóm đã trình bày ở phần 2 đã ảnh hưởng đến hiệu năng của RNN. Nếu phân tích dữ liệu kĩ hơn ta có thể thấy 1 câu bình luận khá dài, điều này lại là nhược điểm của RNN nên kết quả ta nhận được không có gì bất ngờ.

5 Kết luận đánh giá

Mặc dù mô hình *Neural Network* (NN) rất mạnh mẽ trong việc xử lí hình ảnh và nhiều ứng dụng khác, nhưng bản thân NN không có bộ nhớ nên sẽ gặp nhiều vấn đề trong việc giải các bài toán về dữ liệu mang tính thứ tự như văn bản,...Điều này giới hạn NN trong khuôn khổ giải quyết những bài toán có các input độc lập. *Recurrent Neural Network* (RNN) giải quyết những vấn đề trên bằng cách lưu giữ những thông tin ngắn hạn về những tính toán nó đã thực hiện trước đó, do vậy mà một vài input trước đó sẽ ảnh hưởng đến việc sinh ra output lúc sau, tuy nhiên do gặp phải vấn đề về *vanishing gradient/exploding gradient* nên khả năng nhớ của RNN cũng hạn chế. *Long Short-term Memory* (LSTM) mở rộng ý tưởng của RNN bằng cách tạo ra bộ nhớ dài hạn song song với bộ nhớ ngắn hạn, khiến nó trở thành một công cụ tuyệt vời để xử lí dữ liệu không độc lập.

Hơn thế nữa, mô hình RNN/LSTM còn rất linh động trong việc nhận dữ liệu đầu vào và trả về giá trị đầu ra có kích thước tuỳ ý. Việc này có ý nghĩa và ứng dụng rất lớn trong thực tiễn. Ví dụ như soạn nhạc (music composing), soạn thảo văn bản (text generating),...

Tuy mô hình LSTM mạnh mẽ là vậy, có cải tiến hơn mạng RNN nhưng cũng không hẳn là không có điểm yếu. LSTM có thể nhớ một chuỗi độ dài 100, 1000, nhưng không thể là 10000 hay 100000. RNN/LSTM nói chung rất 'cồng kềnh', chúng đòi hỏi yêu cầu phần cứng nhất định, tài nguyên lớn để có thể train được nhanh chóng.

Về bài toán, **Sentiment Analysis** là một bước tiến quan trọng, giúp ta có thể phân tích nguồn dữ liệu công khai trên mạng xã hội, internet một cách nhanh chóng. Ngoài việc sử dụng hệ thống các thang điểm để cho người dùng đánh giá, bây giờ ta đã có thể tận dụng nhiều lợi thế hơn trong thông tin văn bản như bình luận... bằng những bước tiến công nghệ vượt bậc trong AI: RNN/LSTM.

Tổng kết lại, tuy LSTM là một mô hình tốt để giải các bài toán về dữ liệu có trình tự nhưng vẫn còn một vài hạn chế cần khắc phục, giải quyết và có thể mở rộng ra các mô hình có kết quả tốt hơn.

6 Hướng phát triển trong tương lai

Hiện tại mô hình nhóm chúng mình gặp vấn đề *overfitting* khi lựa chọn epoch không phù hợp, độ chính xác trên tập test vẫn chưa thật sự cao, dữ liệu chưa thật sự nhiều. Trong tương lai nhóm mình muốn khắc phục tình trạng này bằng một số phương pháp như: Gia tăng dữ liệu (tăng dữ liệu từ bộ dữ liệu có sẵn hoặc thu thập thêm dữ liệu review từ bên ngoài), xây dựng một mô hình phù hợp hơn bằng cách điều chỉnh các tham số của mô hình như số lớp, số chiều vector,... Sử dụng một số phương pháp giảm *overfitting* như: Early stopping, Dropout, Batch normalization, Regularization,...

Bài toán **Sentiment Analysis** lần này nhóm chúng mình làm trên bộ dữ liệu tiếng Anh, trong tương lai, nhóm muốn áp dụng cho bài toán này với ngôn ngữ là tiếng Việt. Nhìn chung thì các bước tiền xử lý dữ liệu tiếng Anh khá là đơn giản, đơn cử như là việc tokenizer với tiếng Anh dễ dàng hơn nhiều so với tiếng Việt. Vì phần lớn các từ tiếng Anh chỉ cần tách ra 1 từ là có nghĩa còn đối với hệ thống tiếng Việt thì ta có hệ thống từ ghép vô cùng phong phú, khi 2 từ đứng một mình thì chúng không có nghĩa nhưng khi ghép lại thì tạo thành 1 từ có nghĩa như: bắt bẻ, dạy dỗ, cà khịa,... Chưa kể đại từ nhân xưng của tiếng Việt lại phong phú vô cùng phụ thuộc vào cảm xúc người nói, người viết cộng thêm dấu câu, dấu chữ,... Tổng kết lại ta thấy bài toán **Sentiment Analysis** cho tiếng Việt khó khăn hơn rất nhiều so với tiếng Anh đặc biệt là khi dữ liệu ngôn ngữ ở Việt Nam lại vô cùng "quý hiếm" - đây chính là phần khó khăn nhất trong bài toán này vì với mô hình DL thì dữ liệu là thứ không thể thiếu. Khó khăn là thế nhưng nhóm chúng mình vẫn mong muốn có thể nghiên cứu về bài toán này bởi vì điều đó thật sự có ý nghĩa cho cộng đồng Machine Learning Việt Nam.

7 Các kí hiệu đã dùng

- $\cdot$ là tích vô hướng giữa 2 vector.
  Ví dụ: cho 2 vector $\mathbf{a}$, $\mathbf{b} \in \mathbb{R}^n$

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T\mathbf{b} = a_1b_1 + a_2b_2 + ... + a_nb_n$$

- $\odot$ là element-wise product nghĩa là nhân 2 phần tử tương ứng của 2 vector để cho ra một vector mới.
  Ví dụ:

$$\mathbf{a} \odot \mathbf{b} = \begin{bmatrix} a_1b_1 \\ a_2b_2 \\ ... \\ a_nb_n \end{bmatrix}$$

- $\otimes$ là tích ngoài giữa 2 vector cho ra kết quả là một ma trận. Ví dụ:

$$\mathbf{a} \otimes \mathbf{b} = \mathbf{a}\mathbf{b}^T = \begin{bmatrix} a_1b_1 & a_1b_2 & ... & a_1b_n \\ a_2b_1 & a_2b_2 & ... & a_2b_n \\ ... & & & \\ a_nb_1 & a_nb_2 & ... & a_nb_n \end{bmatrix}$$

Tham khảo

[1] Vũ Hữu Tiệp. Machine Learning Cơ Bản. 2017

[2] Amey, L 2013, *Basic Understanding of LSTM*, Good Audience

[3] Bui, H 2018, *Hướng dẫn Chi tiết về Cơ chế của LSTM và GRU trong NLP*, Chappiebot

[4] Chi-Feng, W 2019, *The Vanishing Gradient Problem: the Problem, its Causes, its Significance, and its Solution*, Towards Data Science

[5] Denny, B 2015, *Recurrent Neural Network Tutorial*, Wildml

[6] Eremenko, K 2018, *The Vanishing Gradient Problem*, Super Data Science

[7] Gomez, A 2016, *Backpropagating an LSTM: A Numerical Example*, Medium

[8] Grandic, I 2019, *How LSTM's Work*, Medium

[9] Kapur, R 2016, *Rohan #4: The Vanishing Gradient Problem*, A Year of AI

[10] Lan, H 2017, *The Softmax Function, Neural Net Outputs as Possibilities, and Ensemble Classifiers*, Towards Data Science

[11] Mahmood, H 2018, *The Softmax Function, Simplified*, Towards Data Science

[12] Nicholson, C 2018, *A Beginner's Guide to LSTMs and Recurrent Neural Networks*, Skymind

[13] Pranjal, S 2017, *Essentials of Deep Learning: Introduction to Long Short-term Memory*, Analytics Vidhya

[14] Shibuya, N 2018, *Demystifying Cross-Entropy*, Medium

[15] Yun-nung, C 2016, *Gating Mechanism*, National Taiwan University

[16] bit.ly/IMDBDataset