

RECURRENT NEURAL NETWORK AND THE EXPLODING/VANISHING GRADIENT PROBLEM

Thành viên:

| Mã số sinh viên | Họ tên |
|-----------------|-----------------------|
| 20520394 | Nguyễn Trần Minh Anh |
| 20520193 | Cao Văn Hùng |
| 20521444 | Dương Thành Bảo Khanh |
| 20520174 | Lê Nguyễn Bảo Hân |
| 20520736 | Trương Tấn Sang |

Giới thiệu

Recurrent Neural Networks (RNNs) - mạng nơ-ron hồi quy, được thiết kế để biến đổi một trình tự đầu vào thành một trình tự đầu ra trong một miền khác. Mô hình này phù hợp trong việc giải quyết các vấn đề về nhận dạng chữ viết tay, nhận dạng giọng nói và máy phiên dịch. Trước khi Transformer ra đời, hầu như các tác vụ xử lý ngôn ngữ tự nhiên, đặc biệt trong mảng Machine Translation (dịch máy) đều sử dụng kiến trúc Recurrent Neural Networks (RNNs).

Bài báo cáo này đưa ra một cái nhìn tổng quan về Recurrent Neural Networks. Bài báo cáo sẽ giới thiệu từ tổng quan về Recurrent Neural Network (RNN) cho đến phân tích cơ sở toán học và vấn đề vanishing và exploding mà mô hình RNN truyền thống gặp phải, sau đó sẽ đưa ra một số giải pháp trong đó nổi bật là Long Short Term Memory (LSTM).

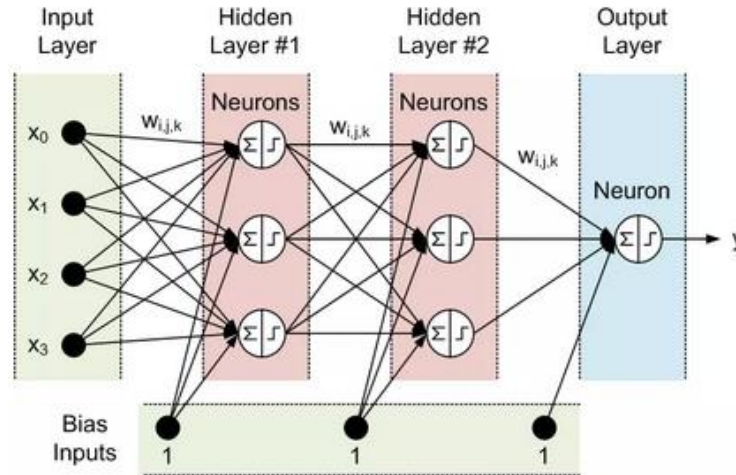
Mục lục

| | |
|--|----|
| 1. RNN | 3 |
| 1.1. Ý tưởng | 3 |
| 1.2. Phân loại bài toán RNN | 4 |
| 2. Feedforward Networks và Backpropagation Through Time | 5 |
| 2.1. Feedforward Networks | 5 |
| 2.2. Backpropagation Through Time | 5 |
| 3. Vanishing và Exploding Gradient | 6 |
| 3.1. Tại sao Gradient Explode và Vanish | 6 |
| 3.2. Giải pháp | 7 |
| 3.2.1. Giải pháp cho Exploding Gradients | 7 |
| 3.2.2. Giải pháp cho Vanishing Gradients | 8 |
| 4. Lời kết | 10 |
| 5. Tài liệu tham khảo | 10 |

1. RNN

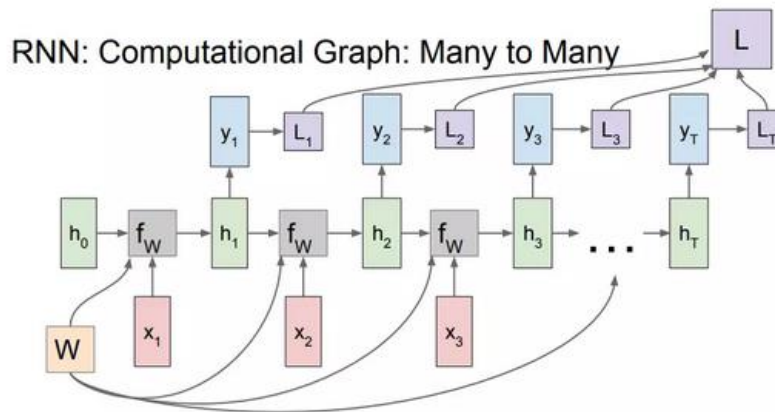
1.1. Ý tưởng

Trước khi bắt đầu tìm hiểu về RNN, ta cùng xét tới một mô hình Neural Network như sau:



Ta đã biết Neural Network có cấu tạo chính bao gồm: Input layer, Hidden layer và Output Layer, đầu vào và đầu ra của mạng neuron này tồn tại độc lập với nhau. Dạng mô hình này không phù hợp để xử lý dạng bài toán dạng chuỗi như mô tả, hoàn thành câu, ... vì trong những bài toán này yêu cầu những dự đoán tiếp theo phải được dựa trên những kết quả từ dự đoán trước đó.

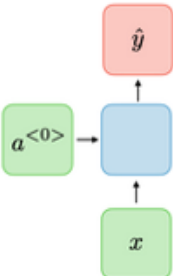
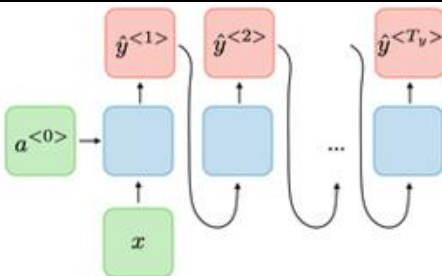
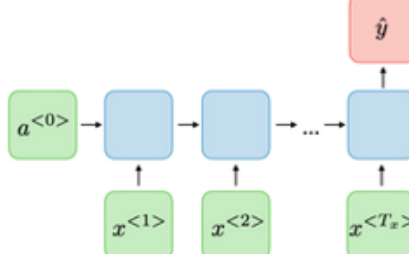
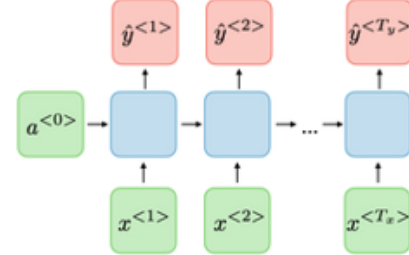
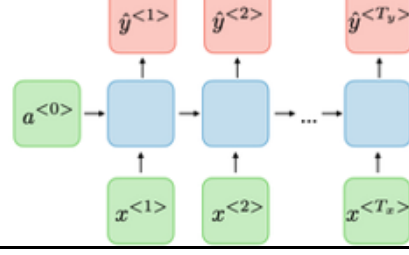
Và như vậy, RNN ra đời với ý tưởng chính là sử dụng một bộ nhớ để lưu lại thông tin từ những xử lý tính toán trước đó là cơ sở để đưa ra những dự đoán tiếp theo.



Mô hình neuron hồi quy

Nếu như mạng Neural Network chỉ là input layer x đi qua hidden layer a và cho ra output layer y với **full connected** giữa các layer thì trong RNN, các input x^t sẽ được kết hợp với hidden layer a^{t-1} bằng activation function để tính ra hidden layer a^t hiện tại và output y^t sẽ được tính ra từ a^t .

1.2. Phân loại bài toán RNN

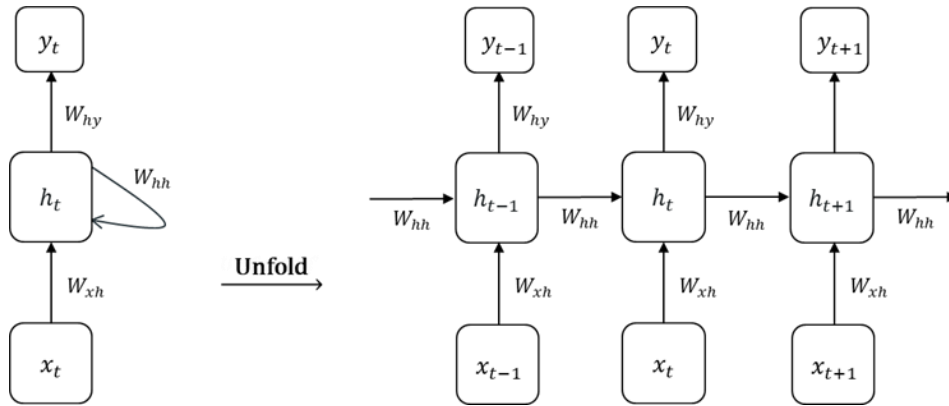
| Các dạng RNN | Minh hoạ | Ví dụ |
|-----------------------------------|---|----------------------------|
| One-to-one $T_x = T_y = 1$ |  | Image Classification |
| One-to-many $T_x = 1, T_y > 1$ |  | Image Captioning |
| Many-to-one $T_x = T_y$ |  | Sentiment Classification |
| Many-to-many $T_x = T_y$ |  | Video Captioning |
| Many-to-many $T_x \neq T_y$ |  | Neural Network Translation |

2. Feedforward Networks và Backpropagation Through Time

2.1. Feedforward Networks

Nếu như mạng Neural Network chỉ là input layer x đi qua hidden layer h và cho ra output layer y thì với RNN, các input x_t sẽ được kết hợp với hidden layer h_{t-1} thông qua một hàm kích hoạt để tính toán ra hidden layer h_t và output y_t . Giá trị h_t này sẽ được sử dụng làm input tiếp theo.

Ta thấy mạng RNN là một mạng neural chứa vòng lặp bên trong nó. Mỗi mạng sẽ truyền thông tin nó vừa xử lý cho mạng phía sau nó. Nếu ta tách từng vòng lặp xử lý ra thành từng mạng con thì ta sẽ có một mạng có kiến trúc như sau:



Cụ thể, ta sẽ có các bước tính toán:

$$h_t = f_W(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

$$y_t = g(W_{hy}h_t + b_y)$$

Mô hình RNN được tham số hóa bởi 3 ma trận trọng số:

- $W_{hx} \in \mathbb{R}^{h \times x}$ là ma trận trọng số giữa input và hidden layer
- $W_{hh} \in \mathbb{R}^{h \times h}$ là ma trận trọng số giữa hai hidden layers
- $W_{hy} \in \mathbb{R}^{h \times y}$ là ma trận trọng số giữa hidden và output layer

Với $b_h \in \mathbb{R}^h$, $b_y \in \mathbb{R}^{h \times y}$ là các bias vectors.

2.2. Backpropagation Through Time

Tương tự như Feedforward Network, về nguyên tắc, ta có thể sử dụng bất kỳ hàm loss nào cho mạng RNN để phù hợp với các nhiệm vụ khác nhau. Một trong số các hàm thường được sử dụng là cross-entropy. Độ lỗi trên toàn bộ mô hình sẽ bằng tổng độ lỗi các bước time step:

$$\frac{\partial loss}{\partial W} = \sum_{k=1}^t \frac{\partial loss_t}{\partial W}$$

Ta cần tìm 3 tham số là W_{xh} , W_{hh} , W_{hy} sao cho tổng độ lỗi là nhỏ nhất với tập dữ liệu huấn luyện. Hướng của việc cập nhật các tham số này sẽ được tính dựa vào đạo hàm của hàm lỗi

$$\frac{\partial loss}{\partial W_{xh}}, \frac{\partial loss}{\partial W_{hh}}, \frac{\partial loss}{\partial W_{hy}}$$

Áp dụng Chain Rule, ta thấy đạo hàm $\frac{\partial \text{loss}}{\partial W_{hy}}$ khá đơn giản vì nó chỉ phụ thuộc vào những giá trị ở bước hiện thời:

$$\frac{\partial \text{loss}}{\partial W_{hy}} = \frac{\partial \text{loss}}{\partial y} \frac{\partial y}{\partial W_{hy}}$$

Tuy nhiên, với W_{xh} và W_{hh} lại không đơn giản như vậy:

$$\frac{\partial \text{loss}}{\partial W_{xh}} = \frac{\partial \text{loss}}{\partial y} \frac{\partial y}{\partial h_t} \frac{\partial h_t}{\partial W_{xh}}, \quad \frac{\partial \text{loss}}{\partial W_{hh}} = \frac{\partial \text{loss}}{\partial y} \frac{\partial y}{\partial h_t} \frac{\partial h_t}{\partial W_{hh}}$$

h_t có h_{t-1} phụ thuộc vào W , h_{t-1} có h_{t-2} phụ thuộc vào W nên ta không thể xem h_{t-1} là hằng số để tính toán như với trường hợp W_{hy} được. Tiếp tục áp dụng Chain Rule, ta có:

$$\frac{\partial \text{loss}_t}{\partial W} = \sum_{k=1}^t \frac{\partial \text{loss}_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial W} \rightarrow \frac{\partial \text{loss}_t}{\partial W} = \sum_{k=1}^t \frac{\partial \text{loss}_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

Với $\frac{\partial h_t}{\partial h_k}$ là tích các ma trận Jacobian (vì các thành phần đều là vector) trên những mạng con liên kết một sự kiện tại thời điểm t và một sự kiện tại thời điểm k .

$$\frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \dots \frac{\partial h_{k+1}}{\partial h_k} = \prod_{j=k+1}^t \frac{\partial h_j}{\partial h_{j-1}}$$

3. Vanishing và Exploding Gradient

Ở phần trước đã giới thiệu kỹ thuật lan truyền ngược - một kỹ thuật thường được sử dụng trong quá trình tranining. Ý tưởng chung của thuật toán là sẽ đi từ output layer đến input layer và tính toán gradient của cost function tương ứng cho từng parameter (weight) của mạng. Gradient Descent sau đó được sử dụng để cập nhật các parameter. Tuy nhiên trong mô hình RNNs, Gradient rất dễ bị tiến dần về 0 (vanishing gradient) hay tiến dần tới vô cùng (exploding gradient) lúc này mô hình sẽ không học được nữa.

Phần này sẽ tìm hiểu lý do tại sao chúng lại thường bị vanishing và exploding gradient. Sau đó sẽ xem xét một số phương pháp giải quyết vấn đề trên.

3.1. Tại sao Gradient Explore và Vanish

Với kỹ thuật backpropagation through time (BPTT), nếu tính tổng độ lỗi tại mỗi thời điểm t ta có công thức độ lỗi sau:

$$\frac{\partial \text{loss}_t}{\partial W} = \sum_{k=1}^t \frac{\partial \text{loss}_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

Trong đó:

$$\frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \dots \frac{\partial h_{k+1}}{\partial h_k} = \prod_{j=k+1}^t \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^t W^T \text{diag}(f'(h_{j-1}))$$

Đặt:

- $\alpha \in \mathbb{R}, \|\text{diag}(f'(h_{j-1}))\| \leq \alpha$
- $\beta = \max(|\text{eigenvalues}(W^T)|)$

$$\Rightarrow \left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \|W^T\| \|diag(f'(h_{j-1}))\| \leq \alpha\beta$$

Đặt $\eta = \alpha\beta$:

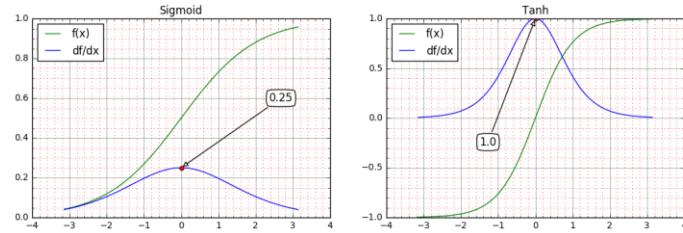
$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^t \frac{\partial h_j}{\partial h_{j-1}} \leq \eta^{t-k}$$

Kết luận, với $(t-k)$ lớn:

- $\alpha < \frac{1}{\beta}$ hoặc $\eta < 1 \rightarrow$ **hiện tượng vanishing gradient**
- $\alpha < \frac{1}{\beta}$ hoặc $\eta > 1 \rightarrow$ **hiện tượng exploding gradient**

Ví dụ: Gradient sẽ tiến dần về 0 khi

- $\alpha < 4$ nếu f là hàm sigmoid vì $\beta = 0.25$
- $\alpha < 1$ nếu f là hàm tanh vì $\beta = 1$



3.2. Giải pháp

3.2.1. Giải pháp cho Exploding Gradients

Truncated Backpropagation Through Time (TBPTT): Phương pháp này sẽ thiết lập n bước thời gian tối đa cùng với lỗi có thể được lan truyền. Có nghĩa là, chúng ta có $t-n$ khi $n \ll k$ do đó hạn chế được số lượng lỗi bước thời gian tổng thể của gradient trong quá trình backpropagation.

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| = \left\| \prod_{i=k+1}^t \frac{\partial h_i}{\partial h_{i-1}} \right\| \leq (\gamma_W \gamma_h)^{t-k}$$

Điều này giúp ngăn không cho gradient phát triển theo cấp số nhân vượt quá n bước. Một nhược điểm lớn của phương pháp này là nó hạn chế khả năng học các long-range dependencies ngoài phạm vi $t - n$ giới hạn.

L1 and L2 Penalty On The Recurrent Weights W_{hh} : Phương pháp này sử dụng sự điều hòa để đảm bảo rằng spectral radius của W_{hh} không vượt quá 1 (điều kiện đủ để các gradient không bị Exploding gradient.)

Nhược điểm: Model bị giới hạn ở một chế độ đơn giản, Không thể sử dụng cho generator model và learn long-range dependencies.

Teacher Forcing: Phương pháp này tìm cách khởi tạo model theo đúng chế độ và đúng vùng không gian. Nó có thể được sử dụng để đào tạo generator model hoặc các mô hình hoạt động với độ dài bộ nhớ không giới hạn.

Hạn chế là nó là ở mỗi bước thời gian yêu cầu phải xác định được hàm mục tiêu

Clipping Gradients: Gradient Clipping là một phương pháp nhằm giảm thiểu hiện tượng Exploding Gradients. Nó hoạt động bằng cách clip gradients trong quá trình backpropagation để ngăn chúng vượt qua một threshold nào đó.

Echo State Networks: Phương pháp này hoạt động bằng cách không học trọng số giữa input và hidden W_{hx} và trọng số giữa hidden và hidden W_{hh} . Thay vào đó ta sẽ thay thế bằng các mẫu phân phối được lựa chọn cẩn thận. Dữ liệu training được sử dụng để học trọng số giữa hidden và output W_{yh}

Hiệu quả của phương pháp là khi các trọng số trong các recurrent connections W_{hh} được lấy mẫu sao cho spectral radius của chúng nhỏ hơn 1 một chút, thông tin đưa vào mô hình được lưu giữ trong một số bước thời gian giới hạn trong quá trình huấn luyện.

Nhược điểm: Không có khả năng learn long-range dependencies. Và có thể dẫn đến vanishing gradient problem.

3.2.2. Giải pháp cho Vanishing Gradients

Hessian Free Optimizer with Structural Dumping: Phương pháp này sử dụng Hessian có khả năng rescale lại tỷ lệ các thành phần ở các high dimensions một cách độc lập. Vì có lẽ, có nhiều khả năng các thành phần dài hạn trực giao với các thành phần ngắn hạn trong thực tế. Tuy nhiên, người ta không thể đảm bảo giữ được những property.

Structural dumping cải thiện điều này bằng cách cho phép mô hình được lựa chọn nhiều hơn trong cách nó xử lý các hướng thay đổi trong không gian tham số, tập trung vào những hướng có nhiều khả năng dẫn đến những thay đổi lớn trong hidden state sequence. Điều này buộc sự thay đổi trạng thái phải nhỏ, khi tham số ΔW thay đổi một giá trị nhỏ.

Leaky Integration Units: This method forces a subset of the units to change slowly using the following leaky integration state to state map:

Phương pháp này buộc một tập hợp con của các đơn vị thay đổi từ từ bằng cách sử dụng leaky integration state thành state map. Hạn chế ở đây là vì các giá trị được chọn cho $\alpha < 1$ nên các gradient vẫn có thể vanish trong khi vẫn explode thông qua fW.

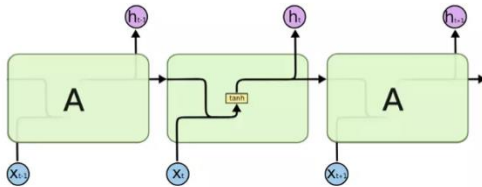
Vanishing Gradient Regularization: Phương pháp này thực hiện một bộ điều chỉnh đảm bảo trong quá trình backpropagation, các gradient không tăng hoặc giảm nhiều về độ lớn.

Long Short-Term Memory:

Ta có bài toán là dự đoán từ tiếp theo trong đoạn văn. Đoạn đầu tiên “Mặt trời mọc ở hướng ...”, ta có thể chỉ sử dụng các từ trước trong câu để đoán là đông. Tuy nhiên, với đoạn, “Tôi là người Việt Nam. Tôi đang sống ở nước ngoài. Tôi có thể nói trôi chảy tiếng ...” thì rõ ràng là chỉ sử dụng từ trong câu đầy hoặc câu trước là không thể dự đoán được từ cần điền là Việt. Ta cần các thông tin từ state ở trước đó rất xa à cần long term memory điều mà RNN không làm được => Cần một mô hình mới để giải quyết vấn đề này à Long short-term memory (LSTM) ra đời.

RNN

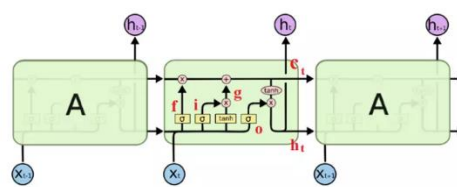
$$h_t = \tanh \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$

*RNN***LSTM**

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

*LSTM*

Đầu tiên, chúng ta có **i**, **f**, **g** có công thức gần giống hệt nhau và chỉ khác mỗi ma trận tham số. Chính ma trận này sẽ quyết định chức năng khác nhau của từng cổng. σ là ký hiệu của hàm sigmoid. Quan sát hình để thấy rõ hơn vị trí các cổng:

- Input gate: Cổng vào giúp quyết định bao nhiêu lượng thông tin đầu vào sẽ ảnh hưởng đến trạng thái mới.
- Forget gate: Cổng quyết định sẽ bỏ đi bao nhiêu lượng thông tin đến từ trạng thái trước đó.
- Output gate: Cổng điều chỉnh lượng thông tin có thể ra ngoài y_t và lượng thông tin truyền tới trạng thái tiếp theo.
- Tiếp theo, **g** thực chất cũng chỉ là một trạng thái ẩn được tính dựa trên đầu vào hiện tại x_t và trạng thái trước h_{t-1} . Tính hoàn toàn tương tự như input gate, chỉ thay vì dùng sigmoid, ta dùng tanh. Kết hợp hai điều này lại để cập nhật trạng thái mới.
- Cuối cùng, ta có c_t là bộ nhớ trong của LSTM. Nhìn vào công thức, có thể thấy nó là tổng hợp của bộ nhớ trước c_{t-1} đã được lọc qua cổng quên **f**, cộng với trạng thái ẩn **g** đã được lọc bởi cổng vào **i**. Cell state sẽ mang thông tin nào quan trọng truyền đi xa hơn và sẽ được dùng khi cần. Đây chính là long term memory.
- Sau khi có c_t , ta sẽ đưa nó qua cổng ra để lọc thông tin một lần nữa, thu được trạng thái mới h_t .

Nhìn một lượt qua kiến trúc LSTM, ta có thể tóm tắt:

Thứ nhất, LSTM có long-term memory. Tuy nhiên, h_t , g_t khá giống với RNN truyền thống, tức có short-term memory. Nhìn chung, LSTM giải quyết phần nào vanishing gradient so với RNN, nhưng chỉ một phần.

Với lượng tính toán như trên, RNN đã chậm, LSTM nay còn chậm hơn. Tuy vậy, với những cải tiến so với RNN thuần, LSTM đã và đang được sử dụng phổ biến. Trên thực tế, cách cài đặt LSTM cũng rất đa dạng và linh hoạt theo bài toán, tuy nhiên vẫn dựa trên LSTM chuẩn như trên.

4. Lời kết

Có thể thấy rằng với dữ liệu dạng sequence (time series) như đoạn văn, video mọi người hay nghe thấy mô hình Recurrent Neural Network (RNN). Tuy nhiên mô hình RNN truyền thống bị vanishing gradient nên học không được tốt. Về sau có mô hình Long Short Term Memory (LSTM) đỡ vanishing gradient hơn RNN truyền thống và thường xuyên được sử dụng. Từ đó khi nói về bài toán sử dụng model RNN thì ngầm hiểu là dùng LSTM/GRU chứ ít ai dùng RNN thuần nữa. Tuy nhiên, với sự phức tạp của LSTM, thì tốc độ train đã trở nên chậm hơn nhiều lần so với RNNs. Vấn đề này ảnh hưởng đáng kể tới tốc độ xử lý để phát triển sản phẩm của NLP nên LSTM cũng chưa được đánh giá cao.

5. Tài liệu tham khảo

1. Vanishing And Exploding Gradient Problems (Jefkine)
2. Pascanu, Razvan; Mikolov, Tomas; Bengio, Yoshua (2012) On the difficulty of training Recurrent Neural Networks [\[PDF\]](#)
3. Doya, K. (1993). Bifurcations of recurrent neural networks in gradient descent learning. IEEE Transactions on Neural Networks, 1, 75–80. [\[PDF\]](#)
4. Martens, J. and Sutskever, I. (2011). Learning recurrent neural networks with Hessian-free optimization. In Proc. ICML'2011 . ACM. [\[PDF\]](#)
5. Jaeger, H., Lukosevicius, M., Popovici, D., and Siewert, U. (2007). Optimization and applications of echo state networks with leaky- integrator neurons. Neural Networks, 20(3), 335–352. [\[PDF\]](#)
6. Yoshua Bengio, Nicolas Boulanger-Lewandowski, Razvan Pascanu, Advances in Optimizing Recurrent Networks arXiv report 1212.0901, 2012. [\[PDF\]](#)
7. Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8):1735–1780. [\[PDF\]](#)