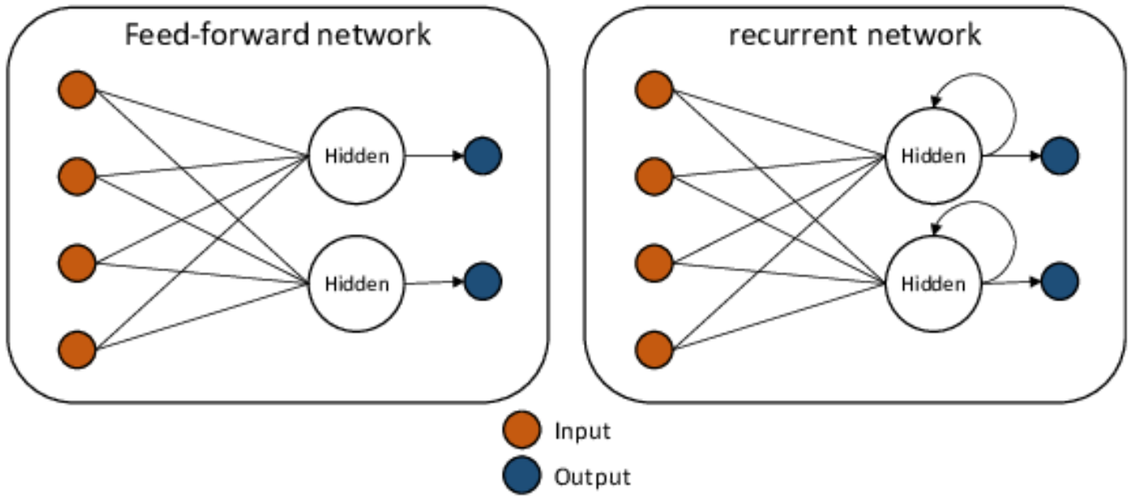
Recurrent neural network

1. Giới thiệu chung về Recurrent neural network (RNN)

Mạng nơ ron hồi quy (RNN) là mô hình rất phổ biến trong thời gian trước với những kết quả đầy hứa hẹn trên các bài toán xử lý ngôn ngữ tự nhiên (NLP). Cho dù ở thời điểm hiện tại, với sự phát triển của cơ chế Attention và các mô hình Transformer đạt kết quả cao trên các bài toán xử lý ngôn ngữ, ý tưởng về cơ chế hoạt động của các mô hình RNN vẫn đáng chú ý và được áp dụng trong một số trường hợp cụ thể.

1.1. Kiến trúc của mô hình RNN

Trong các mạng nơ ron truyền thống, các giá trị đầu vào và đầu ra là độc lập với nhau, tức là chúng không liên kết thành chuỗi với nhau, tuy nhiên, đối với bài toán xử lý ngôn ngữ, đây là một ý tưởng rất tồi. Trong việc xử lý ngôn ngữ tự nhiên, nếu muốn đoán từ tiếp theo có thể xuất hiện trong một câu thì ta cũng cần biết các từ trước đó xuất hiện lần lượt thế nào.



Từ đó, ý tưởng chính của RNN là sử dụng các chuỗi thông tin làm đầu vào cho mạng. RNN được gọi là hồi quy bởi lẽ chúng thực hiện cùng một tác vụ cho tất cả các phần tử của một chuỗi đầu vào. RNN tính toán giá trị đầu ra phụ thuộc vào cả các phép tính trước đó.

trong đó:

- x_i là giá trị tại vị trí thứ i của chuỗi đầu vào
- h_i là giá trị trạng thái của RNN tại vị trí thứ i
- W_X, W_H, W_Y là các trọng số của mô hình lần lượt tương ứng với x, h, y
- y_i là giá trị tại vị trí thứ i của chuỗi đầu ra

Một số thành phần mang lại sự khác biệt giữa ý tưởng của RNN và mạng nơ ron truyền thống:

- Đầu tiên là h_i, h_i được tính bằng $h_i = f(W_X x_i + W_H h_{i-1})$. Với $i = 0$, tức là ở vị trí đầu tiên trọng mạng, lúc này ta chưa có giá trị h_{-1}. Khi đó, ta thường khởi tạo ngẫu nhiên h_{-1} hoặc khởi tạo h_{-1} là vector 0.
- Tiếp theo là bộ trọng số W_X, W_H, W_Y. Khác với việc mỗi layer có bộ trọng số riêng trong mạng nơ ron truyền thông, trong RNN, W_X, W_H, W_Y được sử dụng chung cho tất cả các layer. Do đó, kết quả đầu ra tại mỗi thời điểm của RNN phụ thuộc hoàn toàn vào giá trị đầu vào và giá trị trạng thái ở thời điểm đó.
- Cuối cùng là giá trị đầu ra của layer cuối cùng. Với việc số lượng các layer của RNN không được xác định trước, điều này dẫn đến một câu hỏi về việc khi nào vòng lặp hồi quy sẽ kết thúc. Trong xử lý ngôn ngữ tự nhiên, có một từ đặc biệt được gọi là "từ kết thúc" và khi mô hình RNN tính toán đầu ra là "từ kết thúc" thì vòng lặp hồi quy của mô hình RNN sẽ dừng lại.

Với ý tưởng trên, về lý thuyết, RNN có khả năng nhớ các thông tin được tính toán ở các bước trước đó. Điều này có nghĩa là RNN có thể ghi nhớ và sử dụng được thông tin của một văn bản rất dài, tuy nhiên thực tế, nó chỉ có thể nhớ được thông tin của một vài bước trước đó.

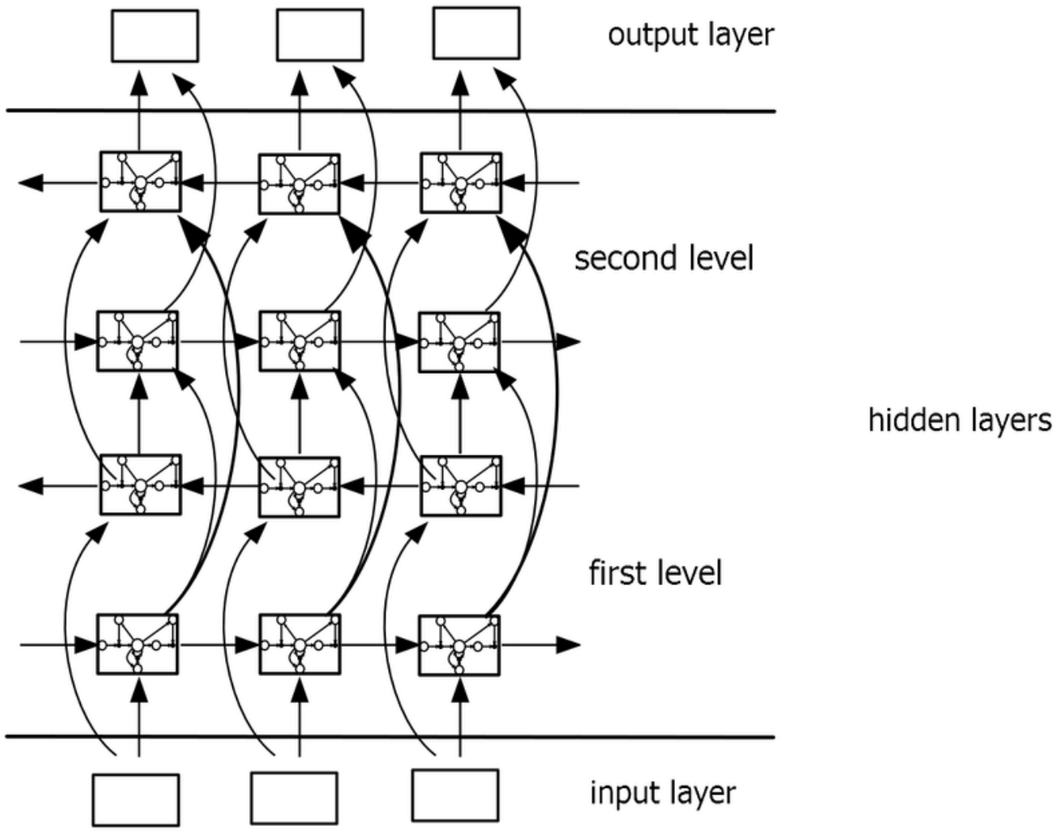
1.2. Một số biến thể của RNN cơ bản

1.2.1. RNN hai chiều (Bidirectional RNN)

Với mô hình RNN hai chiều, đầu ra tại mỗi bước không những phụ thuộc vào các giá trị đầu vào và giá trị trạng thái phía trước mà còn phụ thuộc cả vào các giá trị phía sau. Ví dụ, đối với bài toán điền từ còn thiếu trong câu, ta cần phải xem xét cả các giá trị phía trước trước và phía sau của từ cần điền.

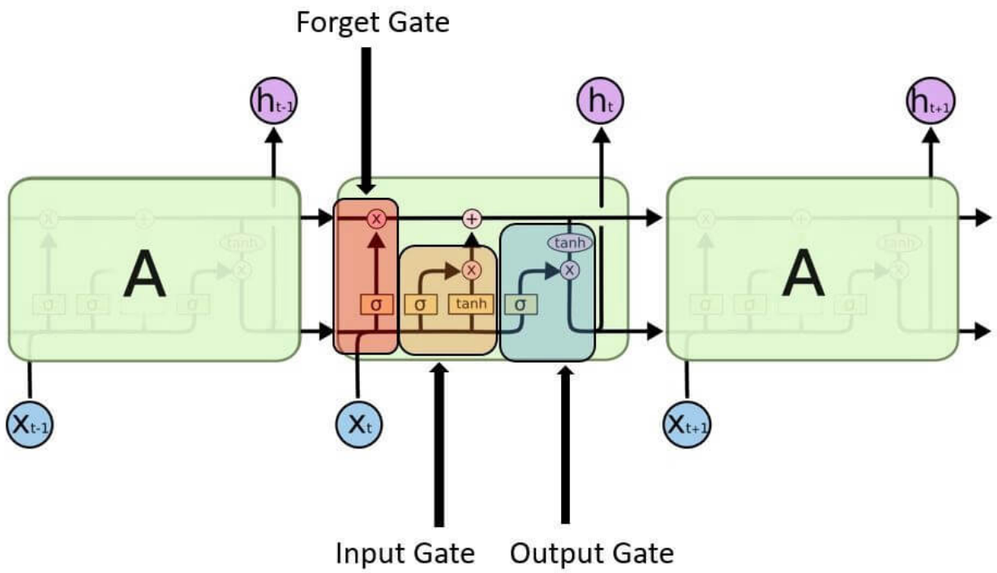
Từ đó, ta có thể coi mô hình Bidirectional RNN là việc chồng 2 mạng RNN ngược hướng lên nhau. Lúc này đầu ra được tính toán dựa vào cả 2 trạng thái ẩn của 2 mạng RNN ngược hướng này.

Một phiên bản khác của Bidirectional RNN là Deep bidirectional RNN - RNN hai chiều sâu.



1.2.2. Long short-term memory (LSTM)

LSTM không khác nhiều mô hình truyền thống của RNN, nhưng LSTM sử dụng hàm tính toán khác ở các trạng thái ẩn. LSTM là một dạng đặc biệt của RNN có khả năng học được các phụ thuộc xa, hoạt động hiệu quả hơn so với RNN cơ bản trên nhiều bài toán khác nhau và dần dần trở nên phổ biến hiện nay.

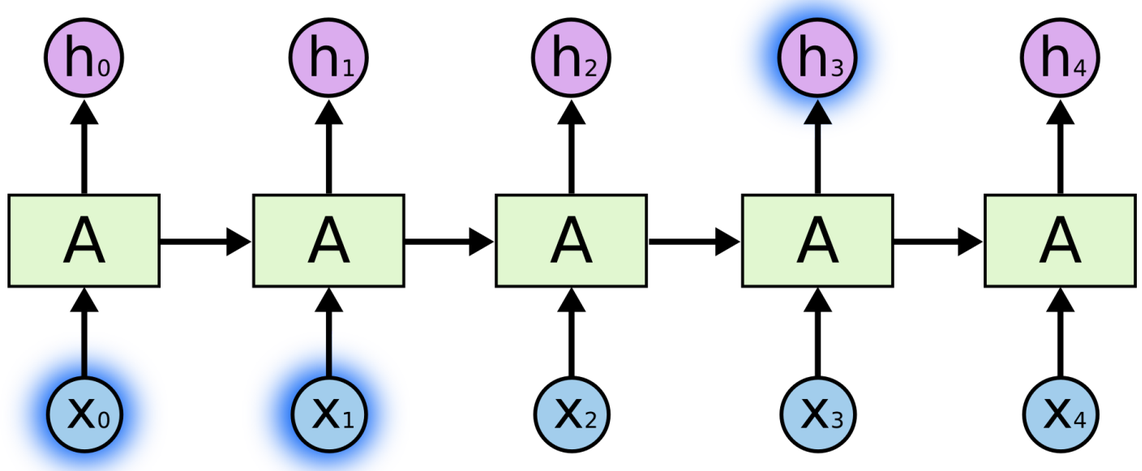


LSTM là phiên bản phổ biến và hiệu quả của RNN, do đó LSTM cũng có nhiều biến thể khác nhau.

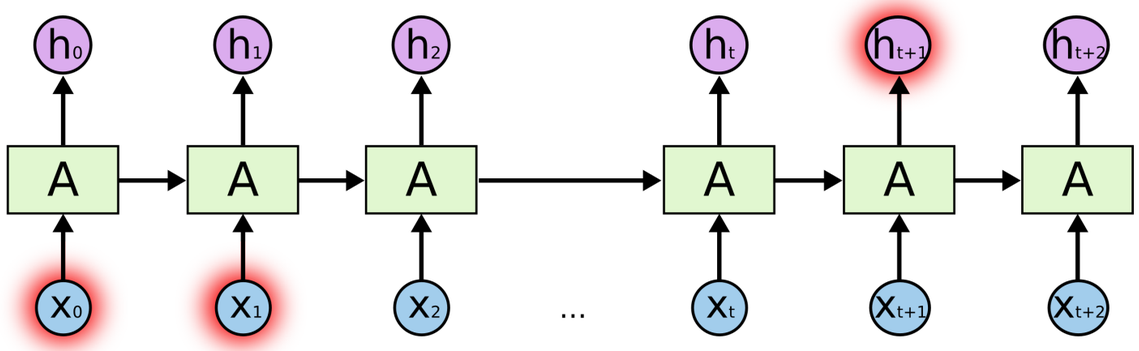
3. Vấn đề phụ thuộc xa của RNN

Một điểm nổi bật của RNN chính là ý tưởng kết nối các thông tin phía trước để dự đoán cho hiện tại. Nếu mà RNN có thể làm được việc đó thì chúng sẽ cực kì hữu dụng cho các bài toán xử lý ngôn ngữ, tuy nhiên, không phải lúc nào RNN cũng có thể làm được điều này.

Ví dụ, với câu "Các đám mây trên bầu trời", ta chỉ cần đọc tới "Các đám mây trên bầu ..." là đủ biết được chữ tiếp theo là "trời" rồi. Do đó, ta rút ra, với khoảng cách tới thông tin có được cần để dự đoán là nhỏ, nên RNN hoàn toàn có thể học được.



Ví dụ tiếp theo, với câu “Tôi lớn lên ở Pháp. ... Tôi nói rất lưu loát tiếng Pháp”. Rõ ràng là các thông tin gần “Tôi nói rất lưu loát” chỉ có phép ta biết được đằng sau nó sẽ là tên của một ngôn ngữ nào đó. Do đó, ta cần phải có thêm ngữ cảnh “Tôi lớn lên ở Pháp.” thì mới có thể suy luận được. Rõ ràng là khoảng cách thông tin lúc này có thể đã khá xa. Do đó, ta rút ra, với khoảng cách càng lớn dần thì RNN bắt đầu không thể nhớ và học được nữa.



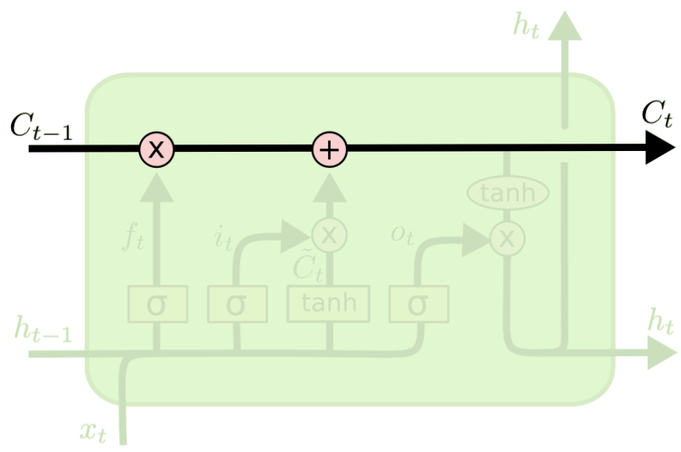
Về mặt lý thuyết, RNN có khả năng xử lý các phụ thuộc xa (long-term dependencies) và chúng ta có thể xem xét và cài đặt các tham số để giải quyết được vấn đề này. Tuy nhiên, Hochreiter (1991) và Bengio (1994) đã công bố các nghiên cứu nêu ra những lý do căn bản để giải thích tại sao RNN không thể học được những tham số đó trong thực tế.

Và LSTM là một mạng cải tiến của RNN nhằm giải quyết vấn đề nhớ các bước dài của RNN.

4. Kiến trúc mô hình Long short-term memory (LSTM)

LSTM là một dạng đặc biệt của RNN có khả năng học được các phụ thuộc xa tốt hơn so với RNN cơ bản, hoạt động cực kì hiệu quả trên nhiều bài toán xử lý ngôn ngữ khác nhau.

Ý tưởng cốt lõi của LSTM là trạng thái tế bào (cell state) - đường chạy thông ngang phía trên của sơ đồ hình vẽ. Cell state là một dạng giống như băng truyền. Nó chạy xuyên suốt tất cả các mắt xích (các nút mạng) và chỉ tương tác tuyến tính đôi chút. Vì vậy mà các thông tin có thể dễ dàng truyền đi thông suốt mà không sợ bị thay đổi.

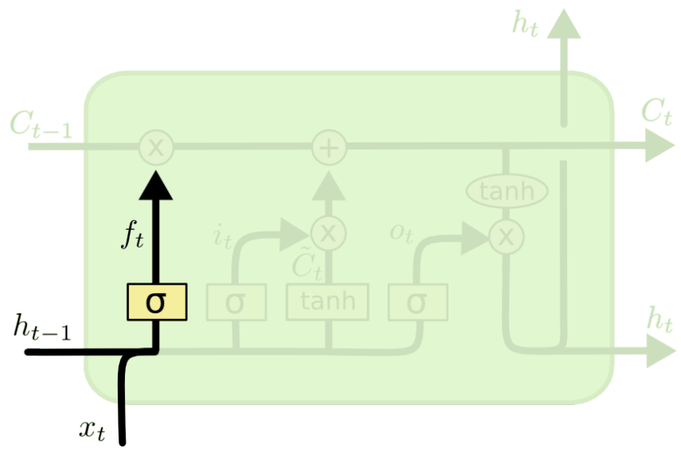


LSTM có khả năng bỏ đi hoặc thêm vào các thông tin cần thiết cho cell state, chúng được điều chỉnh cẩn thận bởi các cổng (gate). Các cổng là nơi sàng lọc thông tin đi qua nó, chúng được kết hợp bởi một tầng mạng sigmoid và một phép nhân.

Một LSTM gồm có 3 cổng để duy trì và điều hành cell state là Forget Layer Gate, Input Layer Gate và Output Layer Gate.

4.1. Forget Layer Gate

Đây là nơi quyết định xem thông tin nào cần phải bỏ đi khỏi cell state. Quyết định này được đưa ra bởi một lớp sigmoid.



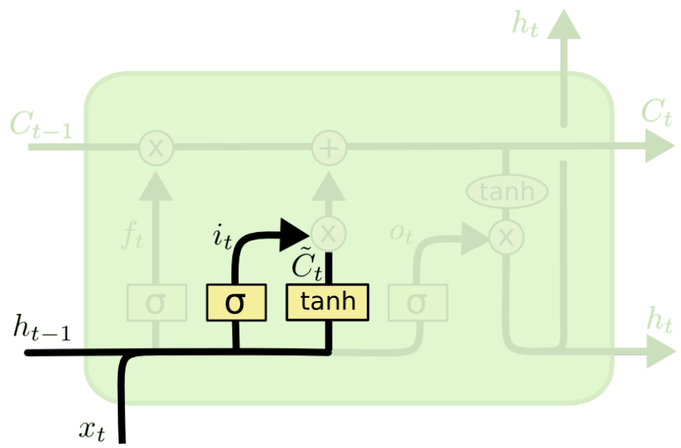
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Ví dụ, trong bài toán Dự đoán từ tiếp theo trong câu: Ta cần ghi nhớ bằng việc lưu lại trong cell state, giới tính của một đối tượng để biết cách xưng hô với đối tượng đó. Tuy nhiên, khi ta nhắc đến một đối tượng mới, thì chúng ta có thể quên đi bằng việc loại bỏ khỏi cell state giới tính của đối tượng cũ.

4.2. Input Layer Gate

Đây là nơi quyết định xem thông tin nào cần phải bổ sung thêm vào cell state và bổ sung thông tin đó vào cell state.

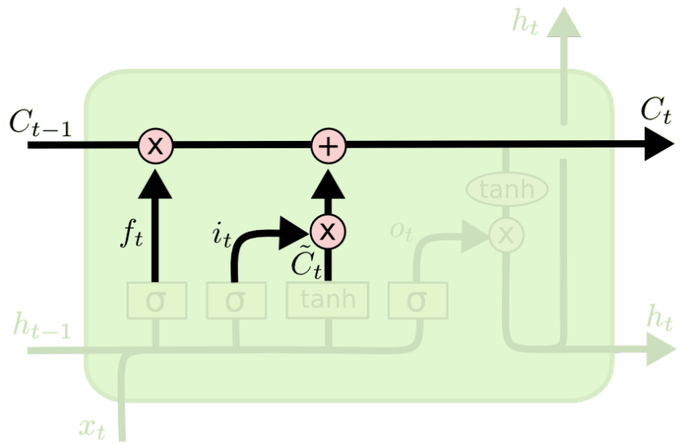
Đầu tiên là sử dụng một lớp sigmoid để quyết định giá trị nào ta sẽ cập nhật. Tiếp theo là một lớp tanh tạo ra một vector cho giá trị mới $\tilde{C}$ nhằm thêm vào cho cell state. Cuối cùng, ta sẽ kết hợp hai giá trị đó lại để tạo ra một cập nhật cho cell state.



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

Ví dụ, tiếp tục với bài toán Dự đoán từ tiếp theo trong câu như trên, ta sẽ muốn ghi nhớ giới tính của đối tượng mới bằng việc lưu vào cell state.

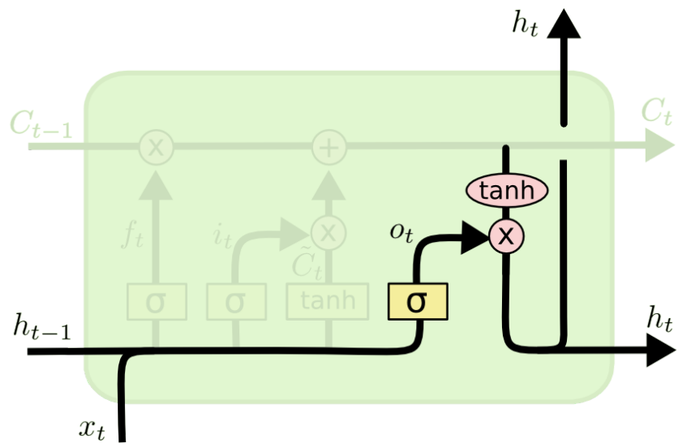
Sau khi thực hiện tính toán với Forget Layer Gate và Input Layer Gate, ta thực hiện cập nhật với cell state bằng việc xoá các thông tin không cần thiết và bổ sung thêm các thông tin mới cần thiết.



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

4.3. Output Layer Gate

Cuối cùng, ta cần quyết định xem ta muốn đầu ra là gì. Giá trị đầu ra sẽ dựa vào cell state, nhưng sẽ được tiếp tục sàng lọc. Cơ chế hoạt động ở Output Layer Gate cũng tương đối giống với Input Layer Gate, ta cũng sử dụng một layer sigmoid để quyết định phần nào của giá trị đầu vào và giá trị của bước trước mà ta muốn trả đầu ra ở bước này. Tiếp theo,ta cũng đưa cell state qua một hàm tanh trước khi nhân để tính toán giá trị đầu ra cuối cùng của layer này.



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

$$h_t = o_t * \tanh\left(C_t\right)$$

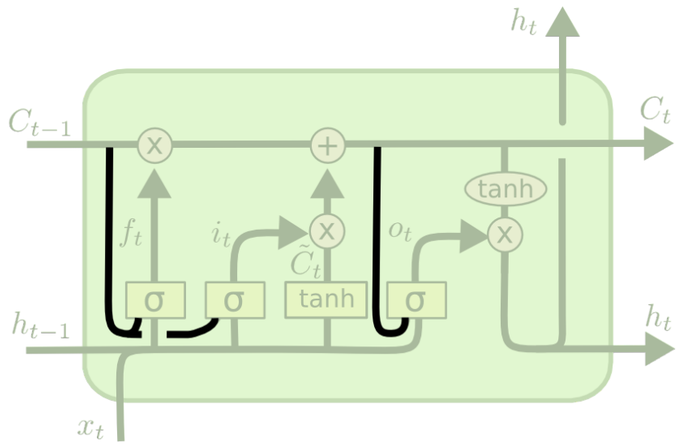
5. Các biến thể của LSTM

Mô hình mà ta vừa mô tả ở trên là một LSTM bình thường, nhưng không phải tất cả các LSTM đều giống như vậy. Thực tế, các nghiên cứu về LSTM đều sử dụng một phiên bản hơi khác so với mô hình LSTM chuẩn. Sự khác nhau không lớn, nhưng chúng giúp giải quyết phần nào đó riêng biệt đối với bài toán cụ thể.

Dưới đây là một vài biến thể được chú ý nhiều nhất, thực tế có rất nhiều các biến thể khác nhau của LSTM

5.1. LSTM với peephole connections

Đây là biến thể giúp cung cấp thêm thông tin về cell state trong các thời điểm cần đưa ra quyết định loại bỏ thông tin khỏi cell state (của Forget Layer Gate), bổ sung thông tin vào cell state (của Input Layer Gate) hay bổ sung thông tin vào kết quả đầu ra (của Output Layer Gate).

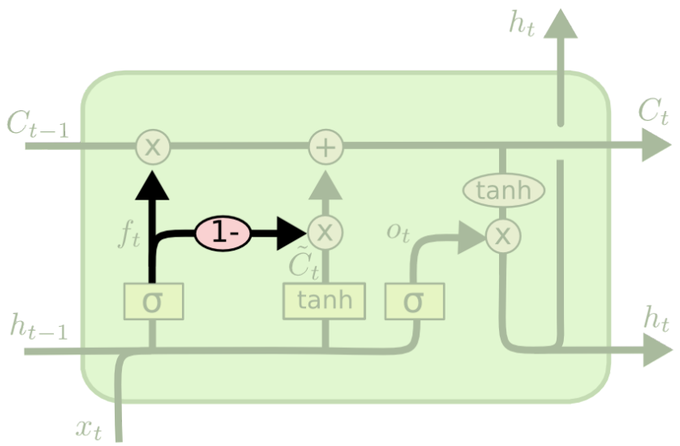


$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

Hình trên mô tả các đường được thêm vào mọi cổng, nhưng cũng có những nghiênc cứu chỉ thêm cho một vài cổng mà thôi.

5.2. LSTM với coupled forget - input gates

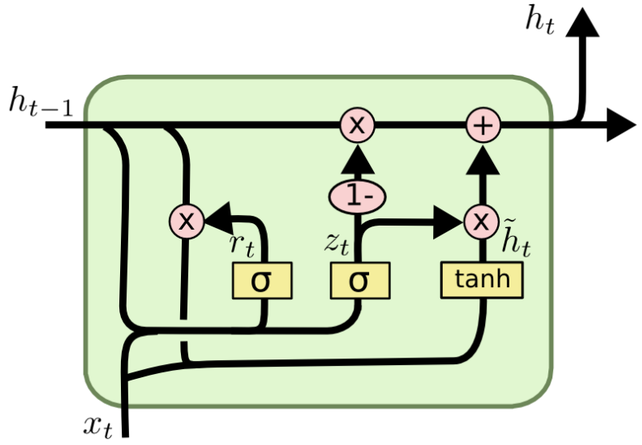
Đây là biến thể giúp LSTM cân bằng giữa phần "quên" và phần bổ sung thêm. Ta chỉ thêm thông tin mới vào cell state khi ta quên bớt thông tin gì đó hoặc ngược lại, ta chỉ quên bớt thông tin gì đó nếu ta bổ sung thêm thông tin mới vào cell state.



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

5.3. Gated Recurrent Unit (GRU)

GRU kết hợp Forget Layer Gate và Input Layer Gate thành Update Gate. GRU cũng kết hợp cell state và hidden state lại với nhau để tạo ra một kiến trúc đơn giản hơn so với LSTM tiêu chuẩn.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$