VII Recurrent Neural Network

16. Recurrent neural network

Deep learning có 2 mô hình lớn là Convolutional Neural Network (CNN) cho bài toán có input là ảnh và Recurrent neural network (RNN) cho bài toán dữ liệu dạng chuỗi (sequence). Tôi đã giới thiệu về Convolutional Neural Network (CNN) và các ứng dụng của deep learning trong computer vision bao gồm: classification, object detection, segmentation. Có thể nói là tương đối đầy đủ các dạng bài toán liên quan đến CNN. Bài này tôi sẽ giới thiệu về RNN.

16.1 Recurrent Neural Network là gì?

Bài toán: Cần phân loại hành động của người trong video, input là video 30s, output là phân loại hành động, ví dụ: đứng, ngồi, chạy, đánh nhau, bắn súng,...

Khi xử lý video ta hay gặp khái niệm FPS (frame per second) tức là bao nhiêu frame (ảnh) mỗi giây. Ví dụ 1 FPS với video 30s tức là lấy ra từ video 30 ảnh, mỗi giây một ảnh để xử lý.

Ta dùng 1 FPS cho video input ở bài toán trên, tức là lấy ra 30 ảnh từ video, ảnh 1 ở giây 1, ảnh 2 ở giây 2,... ảnh 30 ở giây 30. Bây giờ input là 30 ảnh: ảnh 1, ảnh 2,... ảnh 30 và output là phân loại hành động. Nhận xét:

- Các ảnh có thứ tự: ảnh 1 xảy ra trước ảnh 2, ảnh 2 xảy ra trước ảnh 3,... Nếu ta đảo lộn các ảnh thì có thể thay đổi nội dung của video. Ví dụ: nội dung video là cảnh bắn nhau, thứ tự đúng là A bắn trúng người B và B chết, nếu ta đảo thứ tự ảnh thành người B chết xong A mới bắn thì rõ ràng bây giờ A không phải là kẻ giết người => nội dung video bị thay đổi.
- Ta có thể dùng CNN để phân loại 1 ảnh trong 30 ảnh trên, nhưng rõ ràng là 1 ảnh không thể mô tả được nội dung của cả video. Ví dụ: Cảnh người cướp điện thoại, nếu ta chỉ dùng 1 ảnh là người đấy cầm điện thoại lúc cướp xong thì ta không thể biết được cả hành động cướp.

=> Cần một mô hình mới có thể giải quyết được bài toán với input là sequence (chuỗi ảnh 1->30)
=> Recurrent Neural Network (RNN) ra đời.

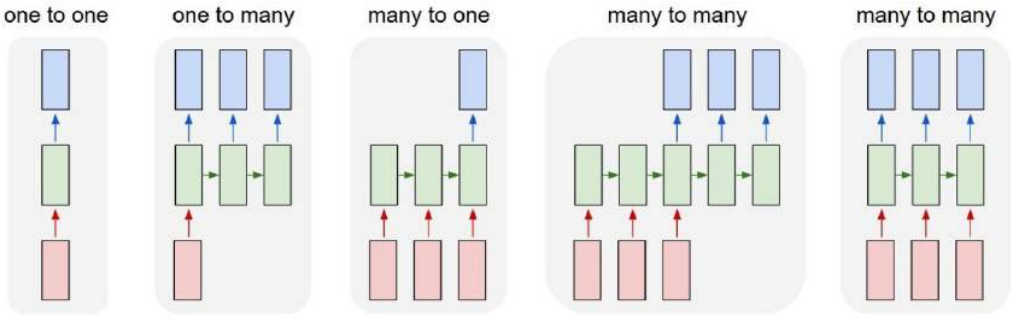
16.1.1 Dữ liệu dạng sequence

Dữ liệu có thứ tự như các ảnh tách từ video ở trên được gọi là sequence, time-series data.

Trong bài toán dự đoán đột quỵ tim cho bệnh nhân bằng các dữ liệu tim mạch khám trước đó. Input là dữ liệu của những lần khám trước đó, ví dụ i1 là lần khám tháng 1, i2 là lần khám tháng 2,... i8 là lần khám tháng 8. (i1,i2,..i8) được gọi là sequence data. RNN sẽ học từ input và dự đoán xem bệnh nhân có bị đột quỵ tim hay không.

Ví dụ khác là trong bài toán dịch tự động với input là 1 câu, ví dụ "tôi yêu Việt Nam" thì vị trí các từ và sự xắp xếp cực kì quan trọng đến nghĩa của câu và dữ liệu input các từ ['tôi', 'yêu', 'việt', 'nam'] được gọi là sequence data. **Trong bài toán xử lý ngôn ngữ (NLP) thì không thể xử lý cả câu được và người ta tách ra từng từ (chữ) làm input, giống như trong video người ta tách ra các ảnh (frame) làm input**.

16.1.2 Phân loại bài toán RNN



Hình 16.1: Các dạng bài toán RNN

One to one: mẫu bài toán cho Neural Network (NN) và Convolutional Neural Network (CNN), 1 input và 1 output, ví dụ với bài toán phân loại ảnh MNIST input là ảnh và output là ảnh đấy là số nào.

One to many: bài toán có 1 input nhưng nhiều output, ví dụ với bài toán caption cho ảnh, input là 1 ảnh nhưng output là nhiều chữ mô tả cho ảnh đấy, dưới dạng một câu.

Hình 16.2: Ví dụ image captioning [10]

Many to one: bài toán có nhiều input nhưng chỉ có 1 output, ví dụ bài toán phân loại hành động trong video, input là nhiều ảnh (frame) tách ra từ video, ouptut là hành động trong video

Many to many: bài toán có nhiều input và nhiều output, ví dụ bài toán dịch từ tiếng anh sang tiếng việt, input là 1 câu gồm nhiều chữ: "I love Vietnam" và output cũng là 1 câu gồm nhiều chữ "Tôi yêu Việt Nam". Để ý là độ dài sequence của input và output có thể khác nhau.

16.1.3 Ứng dụng bài toán RNN

Về cơ bản nếu bạn thấy sequence data hay time-series data và bạn muốn áp dụng deep learning thì bạn nghĩ ngay đến RNN. Dưới đây là một số ứng dụng của RNN:

- **Speech to text**: Chuyển giọng nói sang text.
- **Sentiment classification**: Phân loại bình luận của người dùng, tích cực hay tiêu cực.
- **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.

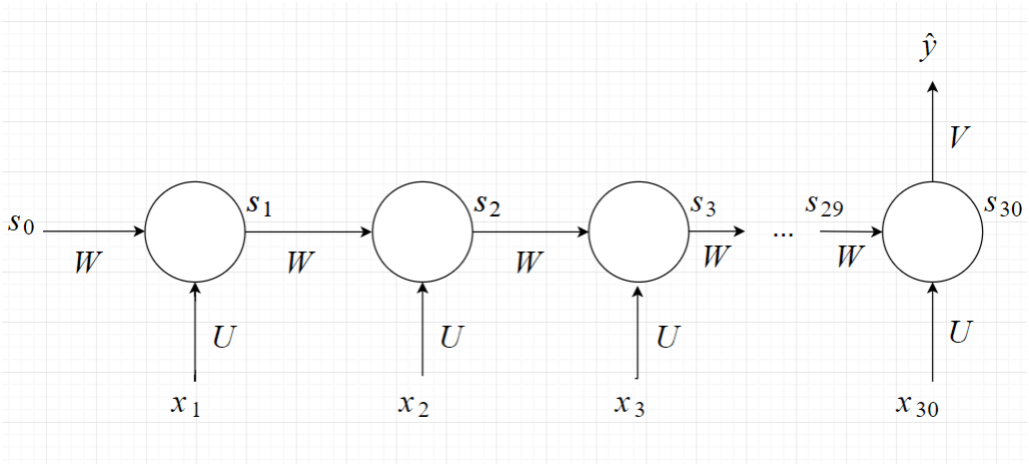
16.2 Mô hình bài toán RNN

16.2.1 Mô hình RNN

Bài toán: Nhận diện hành động trong video 30s. Đây là dạng bài toán many to one trong RNN, tức nhiều input và 1 output.

Input ta sẽ tách video thành 30 ảnh (mỗi giây một ảnh). Các ảnh sẽ được cho qua pretrained model CNN để lấy ra các feature (feature extraction) vector có kích thước n*1. Vector tương ứng với ảnh ở giây thứ i là x_i.

Output là vector có kích thước d*1 (d là số lượng hành động cần phân loại), softmax function được sử dụng như trong bài phân loại ảnh.



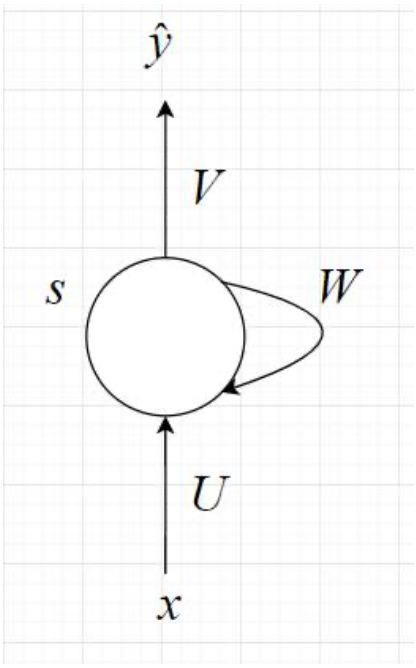
Hình 16.3: Mô hình RNN cho bài toán.

Ta có:

- Mô hình có 30 input và 1 output, các input được cho vào model đúng với thứ tự ảnh trong video $x_1, x_2, ...x_{30}$.

- Mỗi hình tròn được gọi là 1 state, state t có input là x_t và s_{t-1} (output của state trước); output là $s_t = f(U * x_t + W * s_{t-1})$. f là activation function thường là Tanh hoặc ReLU.
- Có thể thấy s_t mang cả thông tin từ state trước (s_{t-1}) và input của state hiện tại => s_t giống như memory nhớ các đặc điểm của các input từ x_1 đến x_t
- s_0 được thêm vào chỉ cho chuẩn công thức nên thường được gán bằng 0 hoặc giá trị ngẫu nhiên. Có thể hiểu là ban đầu chưa có dữ liệu gì để học thì memory rỗng.
- Do ta chỉ có 1 output, nên sẽ được đặt ở state cuối cùng, khi đó s_{30} học được thông tin từ tất cả các input. $\hat{y} = g(V * s_{30})$. g là activation function, trong bài này là bài toán phân loại nên sẽ dùng softmax.

Ta thấy là ở mỗi state các hệ số W, U là giống nhau nên model có thể được viết lại thành:



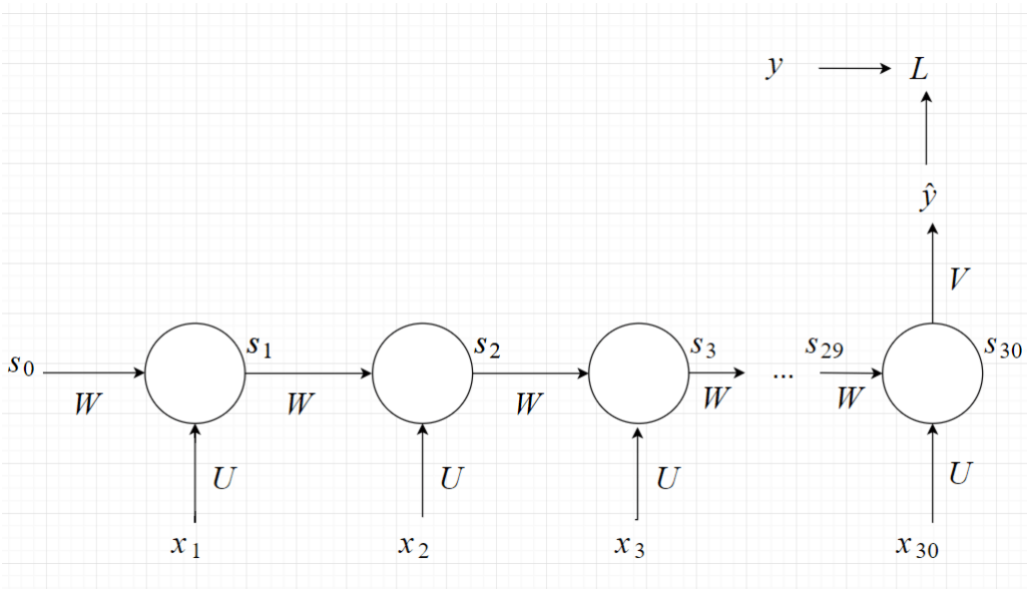
Hình 16.4: Mô hình RNN rút gọn

Tóm lại:

- x_i là vector có kích thước n*1, s_i là vector có kích thước m*1, y_i là vector có kích thước d*1. U là ma trận có kích thước m*n, W là ma trận có kích thước m*m và V là ma trận có kích thước d*m.
- $s_0 = 0, s_t = f(U * x_t + W * s_{t-1})$ với t >= 1
- $\hat{y} = g(V * s_{30})$

16.2.2 Loss function

Loss function của cả mô hình bằng tổng loss của mỗi output, tuy nhiên ở mô hình trên chỉ có 1 output và là bài toán phân loại nên categorical cross entropy loss sẽ được sử dụng.



Hình 16.5: Loss function

16.2.3 Backpropagation Through Time (BPTT)

Có 3 tham số ta cần phải tìm là W, U, V. Để thực hiện gradient descent, ta cần tính: $\frac{\partial L}{\partial U}, \frac{\partial L}{\partial V}, \frac{\partial L}{\partial W}$.

Tính đạo hàm với V thì khá đơn giản:

$$\frac{\partial L}{\partial V} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial V}$$

Tuy nhiên với U, W thì lại khác.

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial s_{30}} * \frac{\partial s_{30}}{\partial W}$$

Do $s_{30} = f(W * s_{29} + U * x_{30})$ có s_{29} phụ thuộc vào W. Nên áp dụng công thức hồi cấp 3 bạn học: $(f(x) * g(x))' = f'(x) * g(x) + f(x) * g'(x)$. Ta có

$\frac{\partial s_{30}}{\partial W} = \frac{\partial s'_{30}}{\partial W} + \frac{\partial s_{30}}{\partial s_{29}} * \frac{\partial s_{29}}{\partial W}$, trong đó $\frac{\partial s'_{30}}{\partial W}$ là đạo hàm của s_{30} với W khi coi s_{29} là constant với W.

Tương tự trong biểu thức s_{29} có s_{28} phụ thuộc vào W, s_{28} có s_{27} phụ thuộc vào W ... nên áp dụng công thức trên và chain rule:

$\frac{\partial L}{\partial W} = \sum_{i=0}^{30} \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial s_{30}} * \frac{\partial s_{30}}{\partial s_i} * \frac{\partial s'_i}{\partial W}$, trong đó $\frac{\partial s_{30}}{\partial s_i} = \prod_{j=i}^{29} \frac{\partial s_{j+1}}{\partial s_j}$ và $\frac{\partial s'_i}{\partial W}$ là đạo hàm của s_i với W khi coi s_{i-1} là constant với W.

Nhìn vào công thức tính đạo hàm của L với W ở trên ta có thể thấy hiện tượng vanishing gradient ở các state đầu nên ta cần mô hình tốt hơn để giảm hiện tượng vaninshing gradient => Long short term memory (LSTM) ra đời và sẽ được giới thiệu ở bài sau. Vì trong bài toán thực tế liên quan đến

time-series data thì LSTM được sử dụng phổ biến hơn là mô hình RNN thuần nên bài này không có code, bài sau sẽ có code ứng dụng với LSTM.

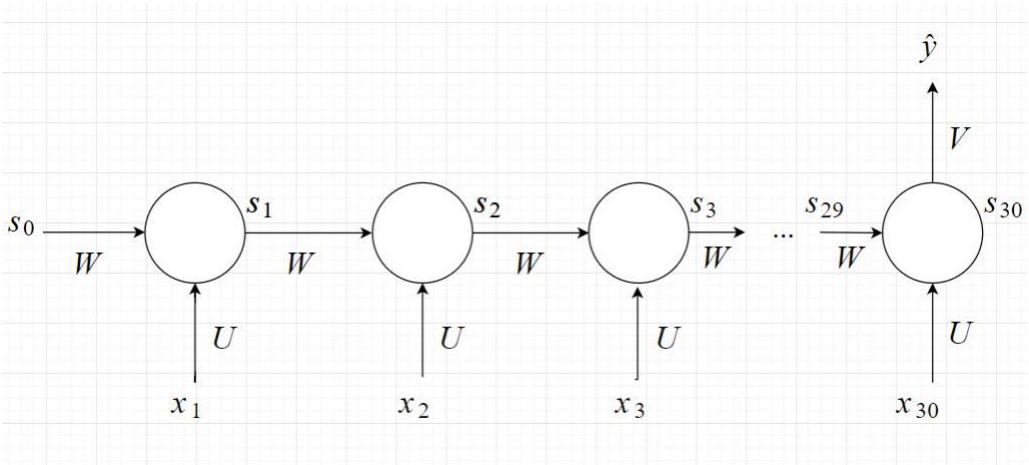
16.3 Bài tập

1. Hệ số trong RNN là gì?
2. Thiết kế và train model RNN dự báo giá Bitcoin, tải dữ liệu ở đây.
3. Tự tìm hiểu và sử dụng mô hình Bidirectional cho bài toán trên.

17. Long short term memory (LSTM)

17.1 Giới thiệu về LSTM

Bài trước tôi đã giới thiệu về recurrent neural network (RNN). RNN có thể xử lý thông tin dạng chuỗi (sequence/ time-series). Như ở bài dự đoán hành động trong video ở bài trước, RNN có thể mang thông tin của frame (ảnh) từ state trước tới các state sau, rồi ở state cuối là sự kết hợp của tất cả các ảnh để dự đoán hành động trong video.



Hình 17.1: Mô hình RNN

Đạo hàm của L với W ở state thứ i: $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial s_{30}} * \frac{\partial s_{30}}{\partial s_i} * \frac{\partial s_i'}{\partial W}$, trong đó $\frac{\partial s_{30}}{\partial s_i} = \prod_{j=i}^{29} \frac{\partial s_{j+1}}{\partial s_j}$

Giả sử activation là tanh function, $s_t = tanh(U * x_t + W * s_{t-1})$

$$\frac{\partial s_t}{\partial s_{t-1}} = (1-s_t^2) * W => \frac{\partial s_{30}}{\partial s_i} = W^{30-i} * \prod_{j=i}^{29}(1-s_j^2).$$

Ta có $s_j < 1, W < 1$ => Ở những state xa thì $\frac{\partial s_{30}}{\partial s_i} \approx 0$ hay $\frac{\partial L}{\partial W} \approx 0$, hiện tượng vanishing gradient

Ta có thể thấy là các state càng xa ở trước đó thì càng bị vanishing gradient và các hệ số không được update với các frame ở xa. Hay nói cách khác là RNN không học được từ các thông tin ở trước đó xa do vanishing gradient.

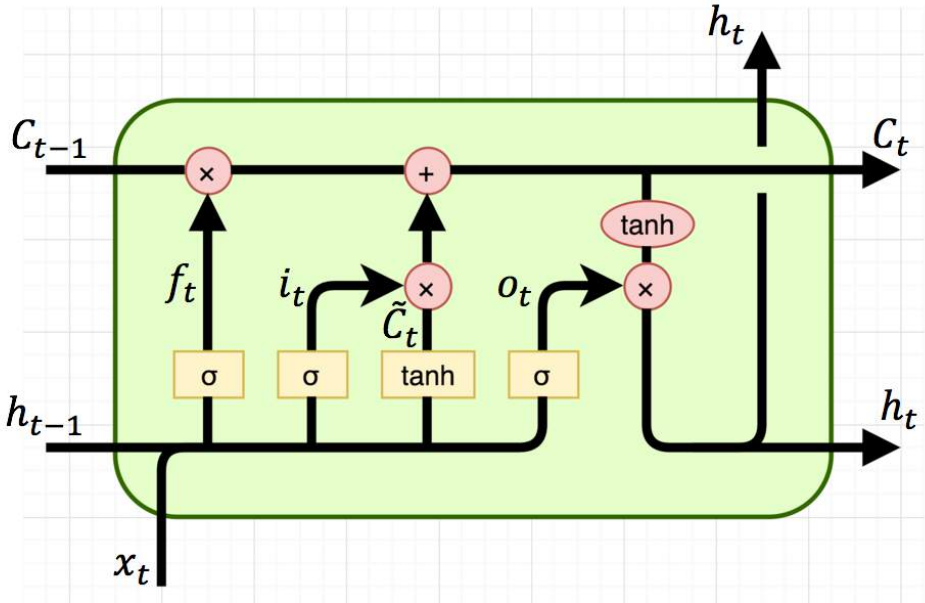
Như vậy về lý thuyết là RNN có thể mang thông tin từ các layer trước đến các layer sau, nhưng thực tế là thông tin chỉ mang được qua một số lượng state nhất định, sau đó thì sẽ bị vanishing gradient, hay nói cách khác là model chỉ học được từ các state gần nó => short term memory.

Cùng thử lấy ví dụ về short term memory nhé. 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 đấy 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 => cần long term memory điều mà RNN không làm được => Cần một mô hình mới để giải quyết vấn đề này => Long short term memory (LSTM) ra đời.

17.2 Mô hình LSTM

Ở state thứ t của mô hình LSTM:

- Output: c_t, h_t, ta gọi c là cell state, h là hidden state.
- Input: c_{t-1}, h_{t-1}, x_t. Trong đó x_t là input ở state thứ t của model. c_{t-1}, h_{t-1} là output của layer trước. h đóng vai trò khá giống như s ở RNN, trong khi c là điểm mới của LSTM.



Hình 17.2: Mô hình LSTM [25]

Cách đọc biểu đồ trên: bạn nhìn thấy kí hiệu σ, tanh ý là bước đấy dùng sigma, tanh activation function. Phép nhân ở đây là element-wise multiplication, phép cộng là cộng ma trận.

f_t, i_t, o_t tương ứng với forget gate, input gate và output gate.

- Forget gate: $f_t = \sigma(U_f * x_t + W_f * h_{t-1} + b_f)$
- Input gate: $i_t = \sigma(U_i * x_t + W_i * h_{t-1} + b_i)$
- Output gate: $o_t = \sigma(U_o * x_t + W_o * h_{t-1} + b_o)$

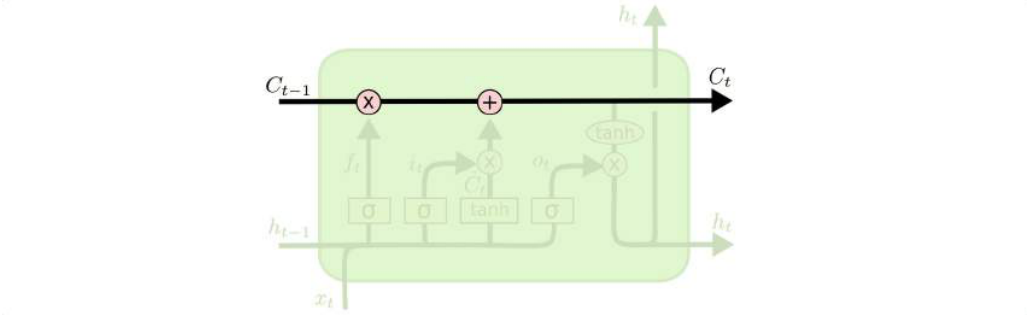
Nhận xét: $0 < f_t, i_t, o_t < 1$; b_f, b_i, b_o là các hệ số bias; hệ số W, U giống như trong bài RNN.

$\tilde{c_t} = \tanh(U_c * x_t + W_c * h_{t-1} + b_c)$, bước này giống hệt như tính s_t trong RNN.

$c_t = f_t * c_{t-1} + i_t * \tilde{c_t}$, **forget gate** quyết định xem cần lấy bao nhiêu từ cell state trước và **input** gate sẽ quyết định lấy bao nhiêu từ input của state và hidden layer của layer trước.

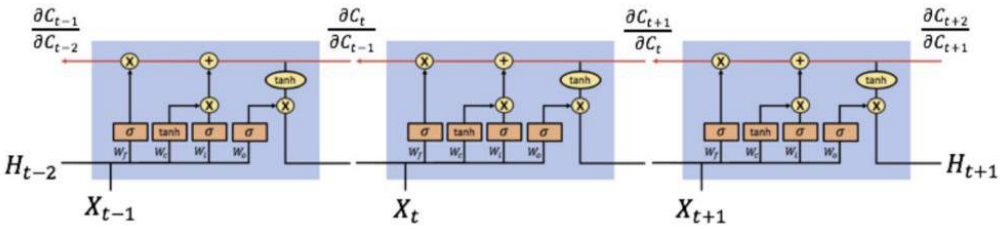
$h_t = o_t * tanh(c_t)$, **output gate** quyết định xem cần lấy bao nhiêu từ cell state để trở thành output của hidden state. Ngoài ra h_t cũng được dùng để tính ra output y_t cho state t.

Nhận xét: $h_t, \tilde{c_t}$ khá giống với RNN, nên model có short term memory. Trong khi đó c_t giống như một băng chuyền ở trên mô hình RNN vậy, thông tin nào cần quan trọng và dùng ở sau sẽ được gửi vào và dùng khi cần => có thể mang thông tin từ đi xa=> long term memory. Do đó mô hình LSTM có cả short term memory và long term memory.



Hình 17.3: cell state trong LSTM

17.3 LSTM chống vanishing gradient



Hình 17.4: Mô hình LSTM [9]

Ta cũng áp dụng thuật toán back propagation through time cho LSTM tương tự như RNN.

Thành phần chính gây là vanishing gradient trong RNN là $\frac{\partial s_{t+1}}{\partial s_t} = (1 - s_t^2) * W$, trong đó $s_t, W < 1$.

Tương tự trong LSTM ta quan tâm đến $\frac{\partial c_t}{\partial c_{t-1}} = f_t$. Do $0 < f_t < 1$ nên về cơ bản thì LSTM vẫn bị vanishing gradient nhưng bị ít hơn so với RNN. Hơn thế nữa, khi mang thông tin trên cell state thì ít khi cần phải quên giá trị cell cũ, nên $f_t \approx 1$ => Tránh được vanishing gradient.

Do đó LSTM được dùng phổ biến hơn RNN cho các toán thông tin dạng chuỗi. Bài sau tôi sẽ giới thiệu về ứng dụng LSTM cho image captioning.

17.4 Bài tập

1. Mô hình LSTM tốt hơn mô hình RNN ở điểm nào?
2. Dùng mô hình LSTM cho bài toán dự đoán bitcoin ở bài RNN.