# VII Recurrent Neural Network


**16 Recurrent neural network . . . . . 227**
16.1 Recurrent Neural Network là gì?
16.2 Mô hình bài toán RNN
16.3 Bài tập

**17 Long short term memory (LSTM) . . . . . 233**
17.1 Giới thiệu về LSTM
17.2 Mô hình LSTM
17.3 LSTM chống vanishing gradient
17.4 Bài tập

**18 Ứng dụng thêm mô tả cho ảnh . . . . . 237**
18.1 Ứng dụng
18.2 Dataset
18.3 Phân tích bài toán
18.4 Các bước chi tiết
18.5 Python code

**19 Seq2seq và attention . . . . . 257**
19.1 Giới thiệu
19.2 Mô hình seq2seq
19.3 Cơ chế attention

# 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.

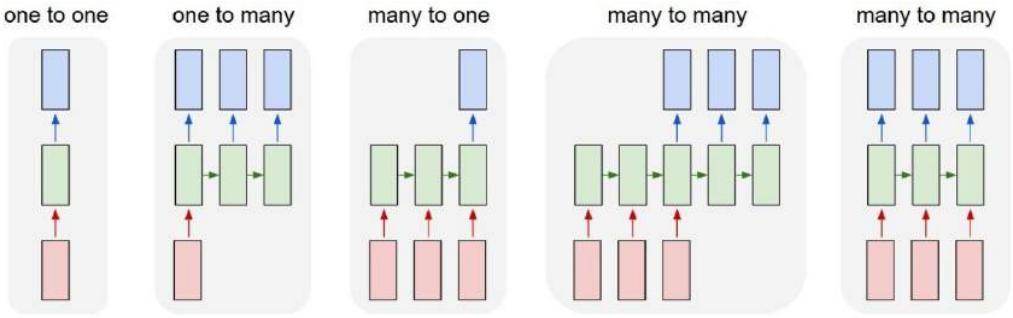
### 16.1.1 Dữ liệu dạng sequence

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.

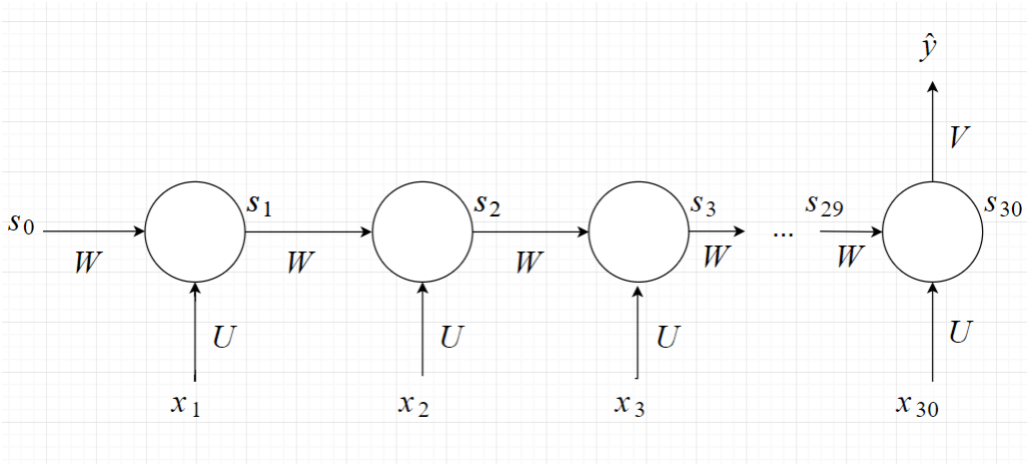
## 16.2 Mô hình bài toán RNN

### 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.



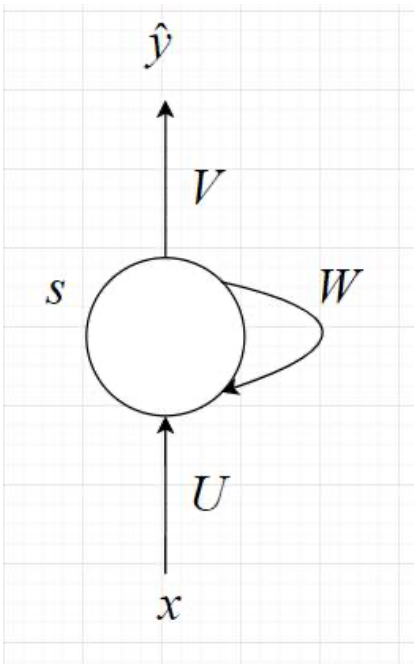

Hình 16.3: Mô hình RNN cho bài toán.

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:



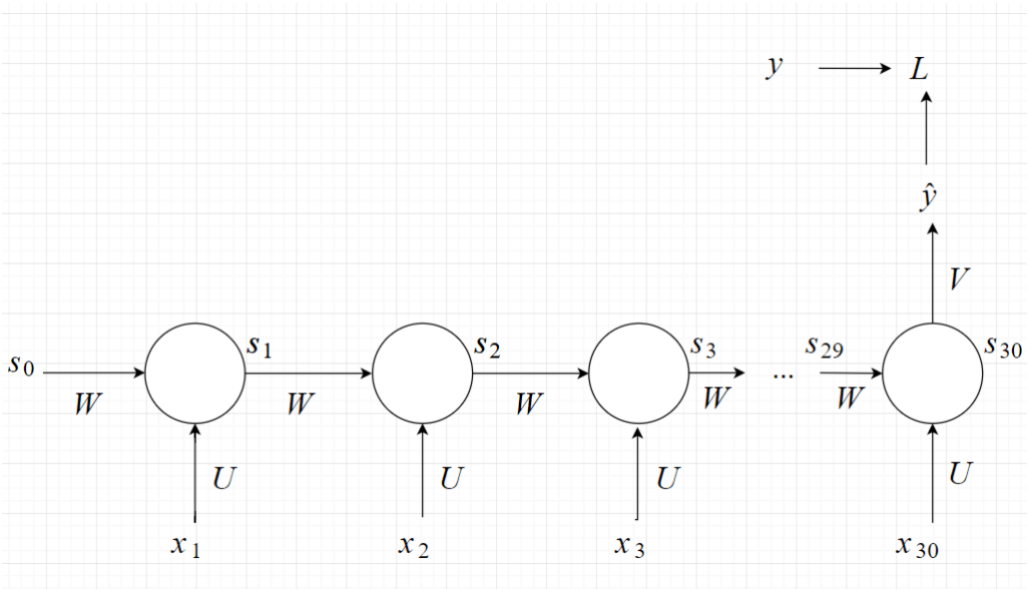

Hình 16.4: Mô hình RNN rút gọn

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.

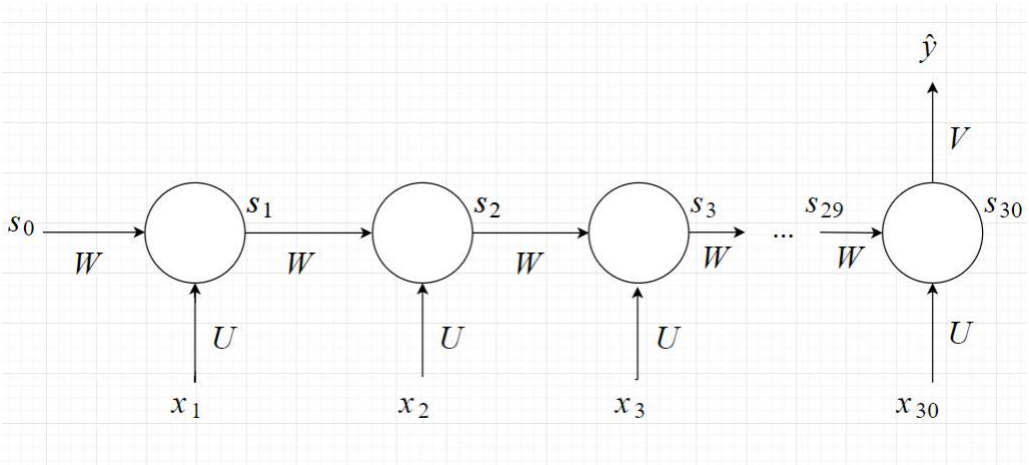
## 16.3 Bài tập

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.

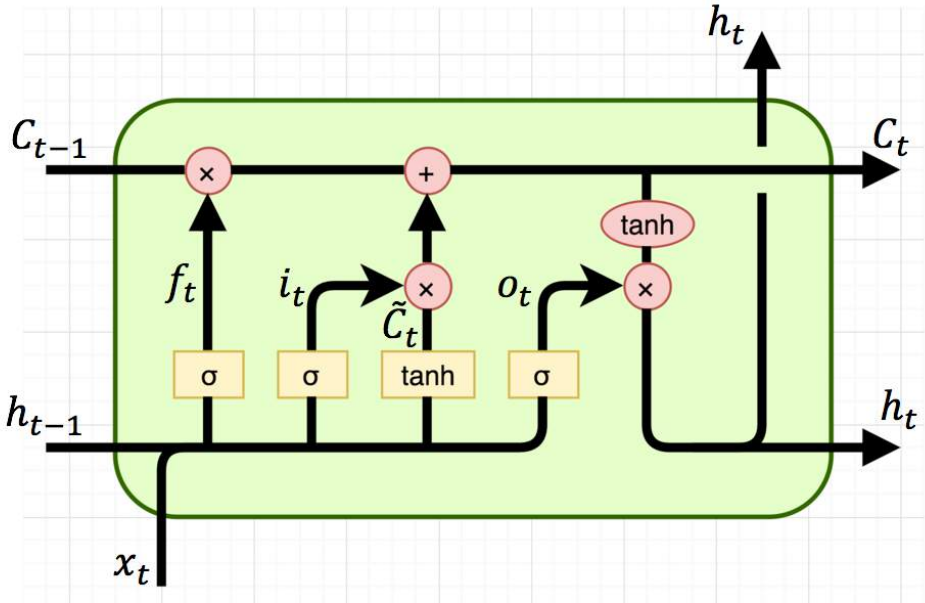
Như vậy về lý thuyết là RNN có thể mang thông tin từ các layer trước đến các layer sau, nhưng thực tế là thông tin chỉ mang được qua một số lượng state nhất định, sau đó thì sẽ bị vanishing gradient, hay nói cách khác là model chỉ học được từ các state gần nó => short term memory.

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 $\sigma$, 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: $o_t = \sigma(U_o * x_t + W_o * h_{t-1} + b_o)$

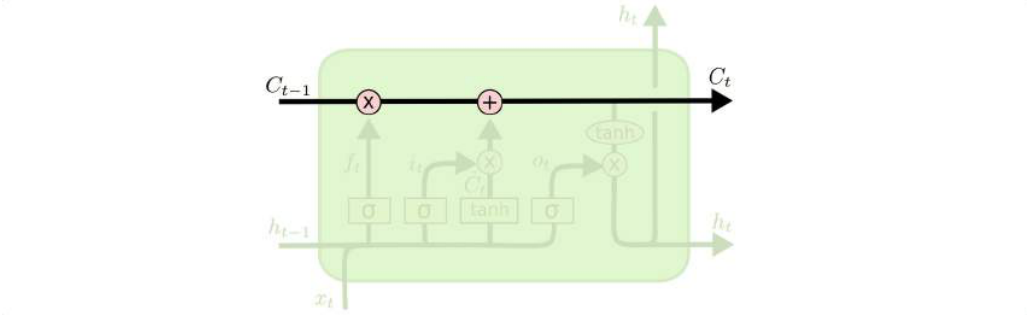
Nhận xét: $0 < f_t, i_t, o_t < 1$; $b_f, b_i, b_o$ là các hệ số bias; hệ số W, U giống như trong bài RNN.

$\tilde{c_t} = \tanh(U_c * x_t + W_c * h_{t-1} + b_c)$, 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.

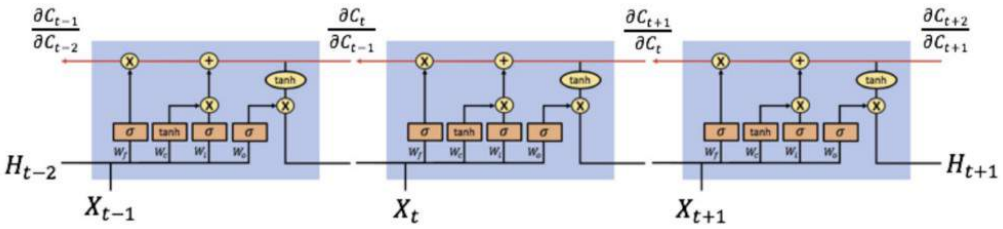
$h_t = o_t * tanh(c_t)$, **output gate** quyết định xem cần lấy bao nhiêu từ cell state để trở thành output của hidden state. Ngoài ra $h_t$ cũng được dùng để tính ra output $y_t$ cho state t.

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.

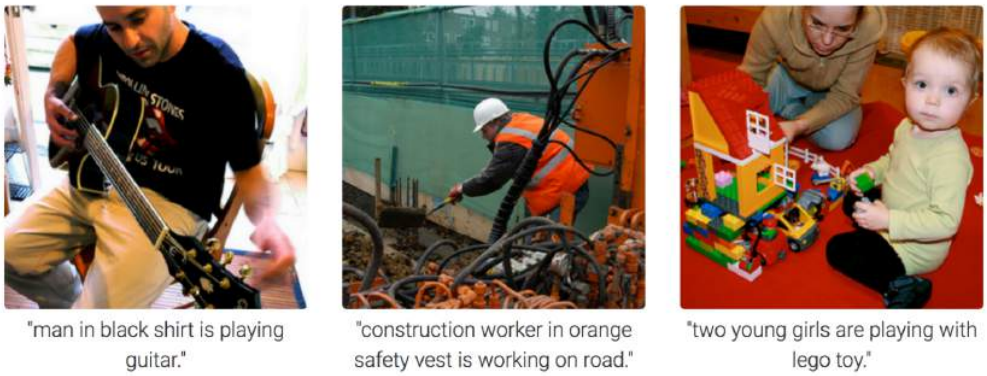
## 17.4 Bài tập

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.

# 18. Ứng dụng thêm mô tả cho ảnh

Ở những bài trước tôi đã giới thiệu về mô hình Recurrent Neural Network (RNN) cho bài toán dữ liệu dạng chuỗi. Tuy nhiên RNN chỉ có short term memory và bị vanishing gradient. Tiếp đó tôi đã giới thiệu về Long short term memory (LSTM) có cả short term memory và long term memory, hơn thế nữa tránh được vaninishing gradient. Bài này tôi sẽ viết về ứng dụng của LSTM cho ứng dụng image captioning.

## 18.1 Ứng dụng




Hình 18.1: Ví dụ image captioning [10]

Ta có thể thấy ngay 2 ứng dụng của image captioning:

- Để giúp những người già mắt kém hoặc người mù có thể biết được cảnh vật xung quanh hay hỗ trợ việc di chuyển. Quy trình sẽ là: Image -> text -> voice.
- Giúp google search có thể tìm kiếm được hình ảnh dựa vào caption.

## 18.2 Dataset

Dữ liệu dùng trong bài này là Flickr8k Dataset. Mọi người tải ở đây. Dữ liệu gồm 8000 ảnh, 6000 ảnh cho training set, 1000 cho dev set (validation set) và 1000 ảnh cho test set.

Bạn tải về có 2 folder: Flicker8k_Dataset và Flicker8k_Text. Flicker8k_Dataset chứa các ảnh với tên là các id khác nhau. Flicker8k_Text chứa:

- Flickr_8k.testImages, Flickr_8k.devImages, Flickr_8k.trainImages, Flickr_8k.devImages chứa id các ảnh dùng cho việc test, train, validation.
- Flickr8k.token chứa các caption của ảnh, mỗi ảnh chứa 5 captions.

Ví dụ ảnh ở hình 18.2 có 5 captions:

- A child in a pink dress is climbing up a set of stairs in an entry way.
- A girl going into a wooden building.
- A little girl climbing into a wooden playhouse.
- A little girl climbing the stairs to her playhouse.
- A little girl in a pink dress going into a wooden cabin.

Thực ra 1 ảnh nhiều caption cũng hợp lý vì bức ảnh có thể được mô tả theo nhiều cách khác nhau. Một ảnh 5 caption sẽ cho ra 5 training set khác nhau: (ảnh, caption 1), (ảnh, caption 2), (ảnh, caption 3), (ảnh, caption 4), (ảnh, caption 5). Như vậy training set sẽ có 6000 * 5 = 40000 dataset.

## 18.3 Phân tích bài toán

Input là ảnh và output là text, ví dụ "man in black shirt is playing guitar".

Nhìn chung các mô hình machine learning hay deep learning đều không xử lý trực tiếp với text như 'man', 'in', 'black',... mà thường phải quy đổi (encode) về dạng số. Từng từ sẽ được encode sang dạng vector với độ dài số định, phương pháp đấy gọi là word embedding.

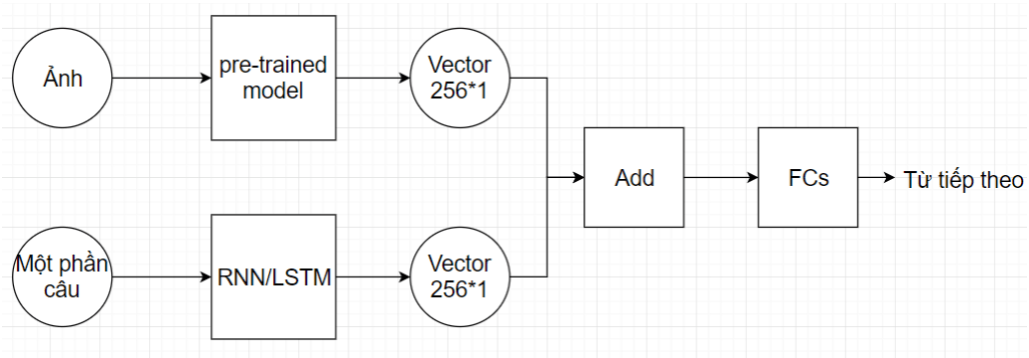
Nhìn thấy output là text nghĩ ngay đến RNN và sử dụng mô hình LSTM.

Input là ảnh thường được extract feature qua pre-trained model với dataset lớn như ImageNet và model phổ biến như VGG16, ResNet, quá trình được gọi là embedding và output là 1 vector.

**Ý tưởng sẽ là dùng embedding của ảnh và dùng các từ phía trước để dự đoán từ tiếp theo trong caption.**

Ví dụ:

- Embedding vector + A -> girl
- Embedding vector + A girl -> going
- Embedding vector + A girl going -> into
- Embedding vector + A girl going into -> a.
- Embedding vector + A girl going into a -> wooden building .
- Embedding vector + A girl going into a wooden -> building .



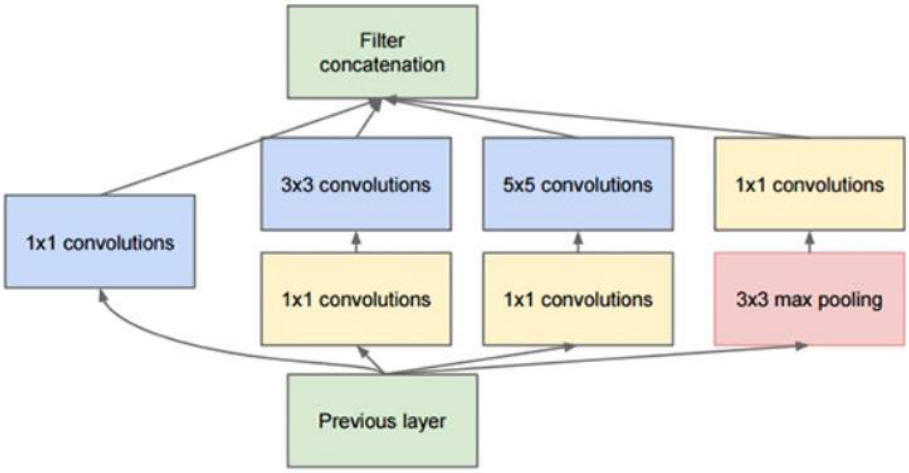

Hình 18.3: Mô hình của bài toán

Để dự đoán từ tiếp theo ta sẽ xây dựng từ điển các từ xuất hiện trong training set (ví dụ 2000 từ) và bài toán trở thành bài toán phân loại từ, xem từ tiếp theo là từ nào, khá giống như bài phân loại ảnh.

## 18.4 Các bước chi tiết

### 18.4.1 Image embedding với Inception

Có lẽ cái tên GoogLeNet sẽ quen thuộc hơn và gặp nhiều hơn so với Inception, GoogLeNet là version 1 của Inception, hiện giờ mô hình phổ biến là Inception v3.




Hình 18.4: Mô hình Googlenet, Going Deeper with Convolutions, Szegedy et al

Thay vì trong mỗi Conv layer chỉ dùng 1 kernel size nhất định như 3*3, 5*5, thì giờ ở một layer có nhiều kernel size khác nhau, do đó mô hình có thể học được nhiều thuộc tính khác nhau của ảnh trong mỗi layer.

Cụ thể hơn mọi người xem thêm ở đây.

Ta sẽ sử dụng pre-trained model Inception v3 với dataset Imagenet. Do là pre-trained model

yêu cầu ảnh đầu vào là 229*229 nên ra sẽ resize ảnh về kích thước này. Sau khi qua pre-trained model ta sẽ lấy được embedding vector của ảnh, kích thước 256*1

### 18.4.2 Text preprocessing

Ta xử lý text qua một số bước cơ bản.

- Chuyển chữ hoa thành chữ thường, "Hello" -> "hello"
- Bỏ các kí tự đặc biệt như "
- Loại bỏ các chữ có số như hey199

Sau đó ta sẽ thêm 2 từ "startseq" và "endseq" để biểu thị sự bắt đầu và kết thúc của caption. Ví dụ: "startseq a girl going into a wooden building endseq". "endseq" dùng khi test ảnh thì biết kết thúc của caption.

Ta thấy có 8763 chữ khác nhau trong số 40000 caption. Tuy nhiên ta không quan tâm lắm những từ mà chỉ xuất hiện 1 vài lần, vì nó giống như là nhiễu vậy và không tốt cho việc học và dự đoán từ của model, nên ta chỉ giữ lại những từ mà xuất hiện trên 10 lần trong số tất cả các caption. Sau khi bỏ những từ xuất hiện ít hơn 10 lần ta còn 1651 từ.

Tuy nhiên do độ dài các sequence khác nhau, ví dụ: "A", " A girl going", " A girl going into a wooden", nên ta cần padding thêm để các chuỗi có cùng độ dài bằng với độ dài của chuỗi dài nhất là 34. Do đó số tổng số từ (từ điển) ta có là 1651 + 1 (từ dùng để padding).

### 18.4.3 Word embedding

Để có thể đưa text vào mô hình deep learning, việc đầu tiên chúng ta cần làm là số hóa các từ đầu vào (embedding). Ở phần này chúng ta sẽ thảo luận về các mô hình nhúng từ (word embedding) và sự ra đời của mô hình word2vec rất nổi tiếng được google giới thiệu vào năm 2013.

#### Các phương pháp trước đây

#### One hot encoding

Phương pháp này là phương pháp đơn giản nhất để đưa từ về dạng số hóa vector với chiều bằng với kích thước bộ từ điển. Mỗi từ sẽ được biểu diễn bởi 1 vector mà giá trị tại vị trí của từ đó trong từ điển bằng 1 và giá trị tại các vị trí còn lại đều bằng 0.
Ví dụ: Ta có 3 câu đầu vào: "Tôi đang đi học", "Mình đang bận nhé", "Tôi sẽ gọi lại sau". Xây dựng bộ từ điển: "Tôi, đang , đi, học, Mình, bận, nhé, sẽ , gọi, lại, sau". Ta có các biểu diễn one hot encoding của từng từ như sau:

Tôi: $[1,0,0,0,0,0,0,0,0,0,0]$,
đang: $[0,1,0,0,0,0,0,0,0,0,0]$,
...
Mình: $[0,0,0,0,1,0,0,0,0,0,0]$,
...
sau: $[0,0,0,0,0,0,0,0,0,0,1]$.

Cách biểu diễn này rất đơn giản, tuy nhiên ta có thể nhận thấy ngay các hạn chế của phương pháp này. Trước hết, one hot encoding không thể hiện được thông tin về ngữ nghĩa của từ, ví dụ như khoảng cách(vector(Tôi) - vector(Mình)) = khoảng cách(vector(Tôi) - vector(đang)), trong khi rõ ràng từ "Tôi" và từ "Mình" trong ngữ cảnh như trên có ý nghĩa rất giống nhau còn từ "Tôi" và từ "đang" lại khác nhau hoàn toàn. Tiếp nữa, mỗi từ đều được biểu diễn bằng một vector có độ dài bằng kích thước bộ từ điển, như bộ từ điển của google gồm 13 triệu từ, thì mỗi one hot vector sẽ dài 13 triệu chiều. Cách biểu diễn này tốn rất nhiều tài nguyên nhưng thông tin biểu diễn được lại rất

hạn hẹp.
=> Cần một cách biểu diễn từ ít chiều hơn và mang nhiều thông tin hơn.

#### Co-occurrence Matrix

Năm 1957, nhà ngôn ngữ học J.R. Firth phát biểu rằng: "Bạn sẽ biết nghĩa của một từ nhờ những từ đi kèm với nó.". Điều này cũng khá dễ hiểu. Ví dụ nhắc đến Việt Nam, người ta thường có các cụm từ quen thuộc như "Chiến tranh Việt Nam", "Cafe Việt Nam", "Việt Nam rừng vàng biển bạc", dựa vào những từ xung quanh ta có thể hiểu hoặc mường tượng ra được "Việt Nam" là gì, như thế nào. Co-occurrence Matrix được xây dựng dựa trên nhận xét trên, co-occurrence đảm bảo quan hệ ngữ nghĩa giữa các từ, dựa trên số lần xuất hiện của các cặp từ trong "context window". Một context window được xác dựa trên kích thước và hướng của nó, ví dụ của context window:

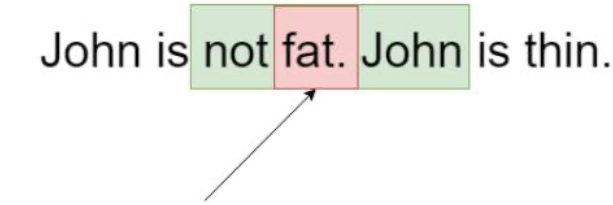



*Figure 1-Ví dụ về Context Window*

Hình 18.5: Ví dụ về context window

Co-ocurrence matrix là một ma trận vuông đối xứng, mỗi hàng, mỗi cột sẽ làm vector đại diện cho từ tương ứng. Từ ví dụ trên ta tiếp tục xây dựng co-ocurrence matrix:

| | John | is | not | fat | thin |
|---|---|---|---|---|---|
| John | 0 | 2 | 0 | 1 | 0 |
| is | 2 | 0 | 1 | 0 | 1 |
| not | 0 | 1 | 0 | 1 | 0 |
| fat | 1 | 0 | 1 | 0 | 0 |
| thin | 0 | 1 | 0 | 0 | 0 |

*Figure 2-Ví dụ về Co-occurence Matrix*

Hình 18.6: Ví dụ về co-ocurrence matrix

Trong đó, giá trị tại ô [i, j] là số lần xuất hiện của từ i nằm trong context window của từ j.
Cách biểu diễn trên mặc dù đã giữ được thông tin về ngữ nghĩa của một từ, tuy vẫn còn các hạn chế như sau:

- Khi kích thước bộ từ điển tăng, chiều vector cũng tăng theo.
- Lưu trữ co-ocurrence matrix cần rất nhiều tài nguyên về bộ nhớ.

- Các mô hình phân lớp bị gặp vấn đề với biểu diễn thưa (có rất nhiều giá trị 0 trong ma trận).

Để làm giảm kích thước của co-ocurrence matrix người ta thường sử dụng phép SVD (Singular Value Decomposition) để giảm chiều ma trận. Ma trận thu được sau SVD có chiều nhỏ hơn, dễ lưu trữ hơn và ý nghĩa của từ cũng cô đọng hơn. Tuy nhiên, SVD có độ phức tạp tính toán cao, tăng nhanh cùng với chiều của ma trận ($O(mn^2)$ với m là chiều của ma trận trước SVD, n là chiều của ma trận sau SVD và $n < m$ ) , ngoài ra phương pháp này cũng gặp khó khăn khi thêm các từ vựng mới vào bộ từ điển.

=> Cần phương pháp khác lưu trữ được nhiều thông tin và vector biểu diễn nhỏ.

#### Word to vec (Word2vec)

Với tư tưởng rằng ngữ cảnh và ý nghĩa của một từ có sự tương quan mật thiết đến nhau, năm 2013 nhóm của Mikolov đề xuất một phương pháp mang tên Word2vec.

Ý tưởng chính của Word2vec

- Thay thế việc lưu thông tin số lần xuất hiện của các từ trong context window như co-occurrence matrix, word2vec học cách dự đoán các từ lận cận.
- Tính toán nhanh hơn và có thể transfer learning khi thêm các từ mới vào bộ từ điển.

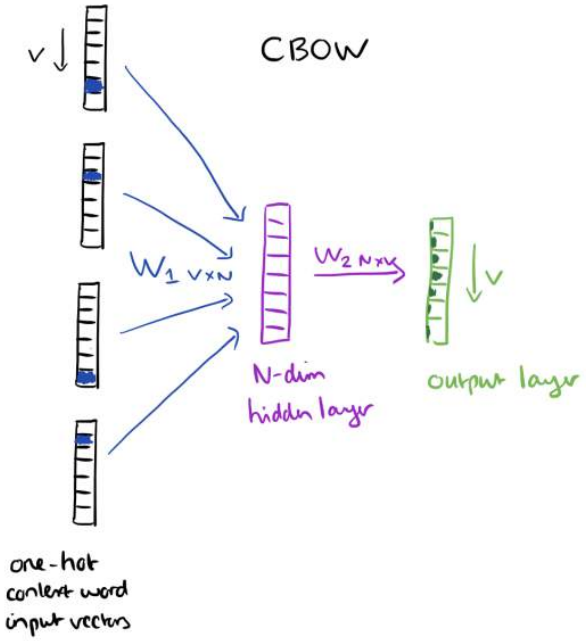
Phương pháp:
Với mỗi từ t trong bộ từ điển ta dự đoán các từ lân cận trong bán kính $m$ của nó.
Hàm mục tiêu nhằm tối ưu xác suất xuất hiện của các từ ngữ cảnh (context word) đối với từ đang xét hiện tại: $J(\theta) = -\frac{1}{T}\prod_{t=1}^{T}\prod_{j=-m,j\neq 0}^{m} p(w_{t+j}|w_t;\theta)$

Có 2 kiến trúc khác nhau của word2vec, là CBoW và Skip-Gram:

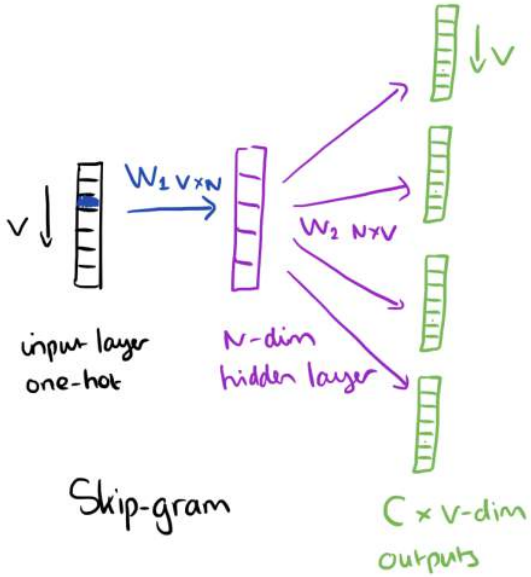
- Cbow: Cho trước ngữ cảnh ta dự đoán xác suất từ đích. Ví dụ: "I ... you", với đầu vào là 2 từ "I" và "you" ta cố gắng dự đoán từ còn thiếu, chẳng hạn "love".
- Skip-Gram: Cho từ đích ta dự đoán xác suất các từ ngữ cảnh (nằm trong context window) của nó. Ví dụ: "... love ...", cho từ "love" ta dự đoán các từ là ngữ cảnh của nó, chẳng hạn "I", "you".

Trong bài báo giới thiệu word2vec, Mikolov và cộng sự có so sánh và cho thấy 2 mô hình này cho kết quả tương đối giống nhau.

Chi tiết mô hình:




Hình 18.7: Mô hình Cbow



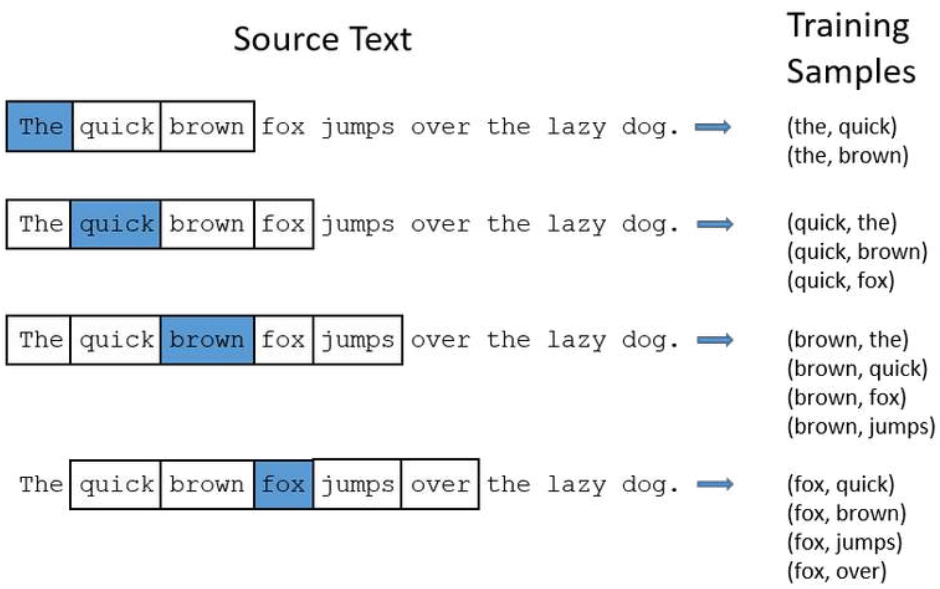

Hình 18.8: Mô hình Skip-Gram

Do 2 kiến trúc khá giống nhau nên ta chỉ thảo luận về Skip-Gram.

Mô hình Skip-Gram sẽ input từ đích và dự đoán ra các từ ngữ cảnh. Thay vì input từ đích và output ra nhiều từ ngữ cảnh trong 1 mô hình, họ xây dựng model để input từ đích và output ra 1 từ ngữ cảnh.

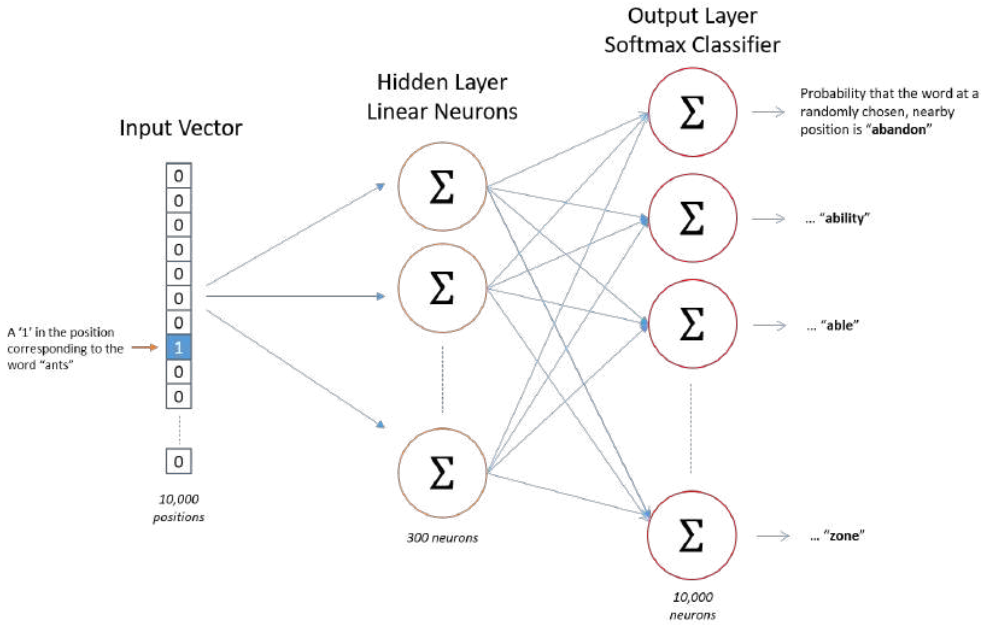
Mô hình là một mạng neural network 2 lớp, với chỉ 1 hidden layer. Input là một từ trong từ điển đã được mã hóa thành dạng one hot vector chiều $V * 1$ với V là kích thước từ điển. Hidden layer không sử dụng activation function có N node, trong đó N chính là độ dài vector embedding của mỗi từ. Output layer có V node, sau đó softmax activation được sử dụng để chuyển về dạng xác suất. Categorical cross entropy loss function được học để dự đoán được từ ngữ cảnh với input là từ đích.

Ví dụ của xây dựng tranning data:



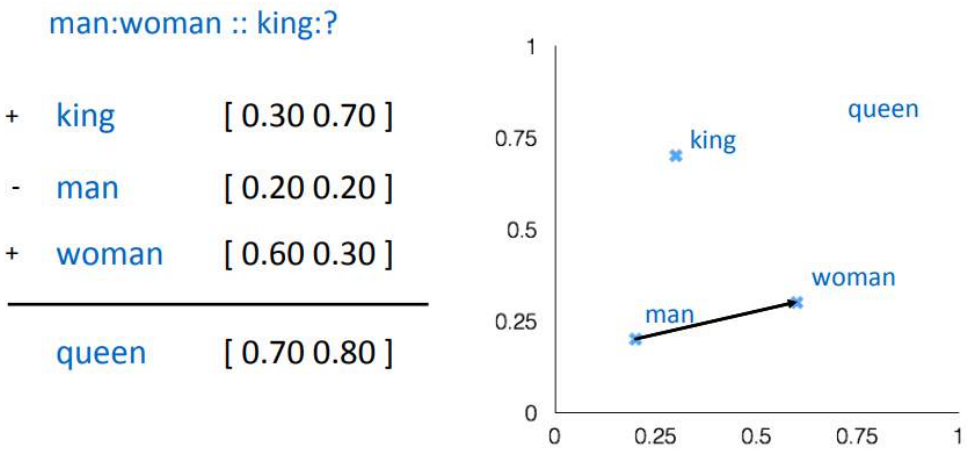

Hình 18.9: Ví dụ của xây dựng training data, window size = 2, tức là lấy 2 từ bên trái và 2 từ bên phải mỗi từ trung tâm làm từ ngữ cảnh (context word)

Ví dụ của model Skip-gram:



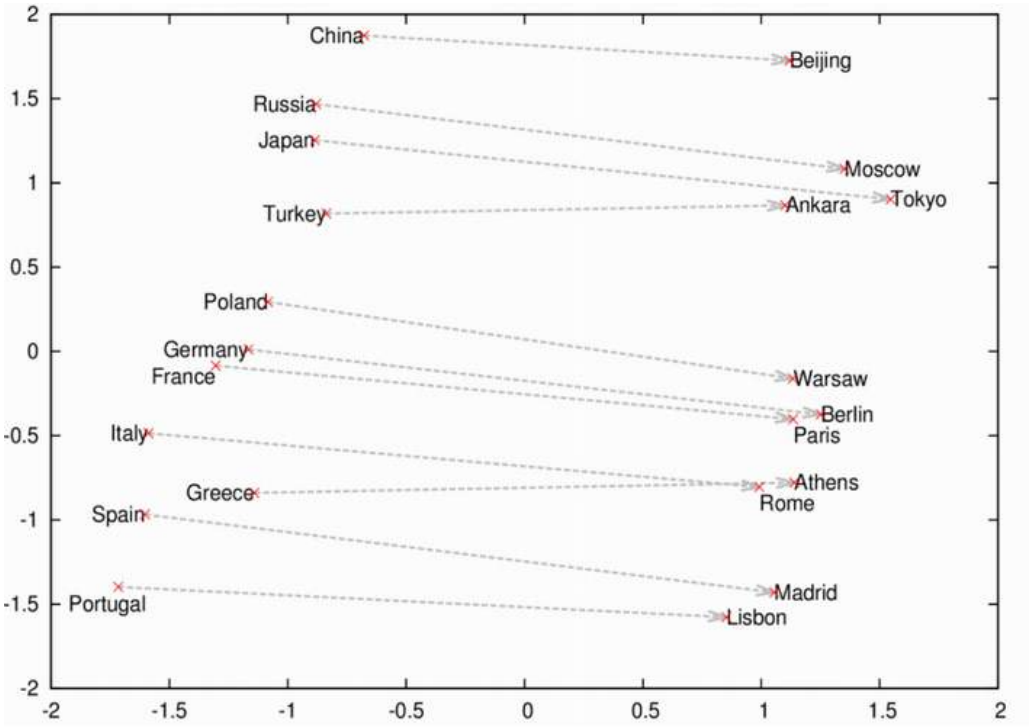

Hình 18.10: Ví dụ của xây dựng Skip-gram

Một số kết quả của Word2vec:




Hình 18.11: Vị trí các word vector trong không gian

Ví dụ trên là ví dụ kinh điển của Word2vec cho thấy các vector biểu diễn tốt quan hệ về mặt ngữ nghĩa của từ vựng như thế nào.




Hình 18.12: Mối quan hệ của đất nước và thủ đô tương ứng

Pre-trained GLOVE Model được sử dụng cho quá trình word embedding.

Mọi người vào link này để tải file glove.6B.zip

Từng dòng trong file sẽ lưu text và encoded vector khích thước 200*1

### 18.4.4 Output

Bài toán là dự đoán từ tiếp theo trong chuỗi ở input với ảnh hiện tại, nên output là từ nào trong số 1652 từ trong từ điển mà ta có. Với bài toán phân loại thì softmax activation và categorical_crossentropy loss function được sử dụng.

### 18.4.5 Model

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
input_3 (InputLayer)         (None, 34)                0
--------------------------------------------------------------------------------------------------
input_2 (InputLayer)         (None, 2048)              0
--------------------------------------------------------------------------------------------------
embedding_1 (Embedding)      (None, 34, 200)           330400      input_3[0][0]
--------------------------------------------------------------------------------------------------
dropout_1 (Dropout)          (None, 2048)              0           input_2[0][0]
--------------------------------------------------------------------------------------------------
dropout_2 (Dropout)          (None, 34, 200)           0           embedding_1[0][0]
--------------------------------------------------------------------------------------------------
```

```
dense_1 (Dense)                 (None, 256)          524544      dropout_1[0][0]
__________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 256)          467968      dropout_2[0][0]
__________________________________________________________________________________________
add_1 (Add)                     (None, 256)          0           dense_1[0][0]
                                                                 lstm_1[0][0]
__________________________________________________________________________________________
dense_2 (Dense)                 (None, 256)          65792       add_1[0][0]
__________________________________________________________________________________________
dense_3 (Dense)                 (None, 1652)         424564      dense_2[0][0]
==========================================================================================
Total params: 1,813,268
Trainable params: 1,813,268
Non-trainable params: 0
```

## 18.5 Python code

```
# -*- coding: utf-8 -*-

# Commented out IPython magic to ensure Python compatibility.
# Thêm thư viện
import numpy as np
from numpy import array
import pandas as pd
import matplotlib.pyplot as plt
# %matplotlib inline
import string
import os
from PIL import Image
import glob
from pickle import dump, load
from time import time

from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import LSTM, Embedding, TimeDistributed, Dense, RepeatVector,\
                         Activation, Flatten, Reshape, concatenate, \
                         Dropout, BatchNormalization
from keras.optimizers import Adam, RMSprop
from keras.layers.wrappers import Bidirectional
from keras.layers.merge import add
from keras.applications.inception_v3 import InceptionV3
from keras.preprocessing import image
from keras.models import Model
from keras import Input, layers
from keras import optimizers
from keras.applications.inception_v3 import preprocess_input
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
```

```
# Đọc file các caption
def load_doc(filename):
        # open the file as read only
        file = open(filename, 'r')
        # read all text
        text = file.read()
        # close the file
        file.close()
        return text

filename = "Flickr8k/Flickr8k_text/Flickr8k.token.txt"

doc = load_doc(filename)
print(doc[:300])

# Lưu caption dưới dạng key value:
#id_image : ['caption 1', 'caption 2', 'caption 3',' caption 4', 'caption 5']
def load_descriptions(doc):
        mapping = dict()
        # process lines
        for line in doc.split('\n'):
                # split line by white space
                tokens = line.split()
                if len(line) < 2:
                        continue
                # take the first token as the image id, the rest as the description
                image_id, image_desc = tokens[0], tokens[1:]
                # extract filename from image id
                image_id = image_id.split('.')[0]
                # convert description tokens back to string
                image_desc = ' '.join(image_desc)
                # create the list if needed
                if image_id not in mapping:
                        mapping[image_id] = list()
                # store description
                mapping[image_id].append(image_desc)
        return mapping

descriptions = load_descriptions(doc)
print('Loaded: %d ' % len(descriptions))

descriptions['1000268201_693b08cb0e']

# Preprocessing text
def clean_descriptions(descriptions):
        # prepare translation table for removing punctuation
        table = str.maketrans('', '', string.punctuation)
        for key, desc_list in descriptions.items():
```

```
                for i in range(len(desc_list)):
                        desc = desc_list[i]
                        # tokenize
                        desc = desc.split()
                        # convert to lower case
                        desc = [word.lower() for word in desc]
                        # remove punctuation from each token
                        desc = [w.translate(table) for w in desc]
                        # remove hanging 's' and 'a'
                        desc = [word for word in desc if len(word)>1]
                        # remove tokens with numbers in them
                        desc = [word for word in desc if word.isalpha()]
                        # store as string
                        desc_list[i] =  ' '.join(desc)

# clean descriptions
clean_descriptions(descriptions)

descriptions['1000268201_693b08cb0e']

# Lưu description xuống file
def save_descriptions(descriptions, filename):
        lines = list()
        for key, desc_list in descriptions.items():
                for desc in desc_list:
                        lines.append(key + ' ' + desc)
        data = '\n'.join(lines)
        file = open(filename, 'w')
        file.write(data)
        file.close()

save_descriptions(descriptions, 'descriptions.txt')

# Lấy id ảnh tương ứng với dữ liệu train, test, dev
def load_set(filename):
        doc = load_doc(filename)
        dataset = list()
        # process line by line
        for line in doc.split('\n'):
                # skip empty lines
                if len(line) < 1:
                        continue
                # get the image identifier
                identifier = line.split('.')[0]
                dataset.append(identifier)
        return set(dataset)

# load training dataset (6K)
filename = 'Flickr8k/Flickr8k_text/Flickr_8k.trainImages.txt'
```

```
train = load_set(filename)
print('Dataset: %d' % len(train))

# Folder chứa dữ ảnh
images = 'Flickr8k/Flicker8k_Dataset/'
# Lấy lấy các ảnh jpg trong thư mục
img = glob.glob(images + '*.jpg')

# File chứa các id ảnh để train
train_images_file = 'Flickr8k/Flickr8k_text/Flickr_8k.trainImages.txt'
# Read the train image names in a set
train_images = set(open(train_images_file, 'r').read().strip().split('\n'))

# Create a list of all the training images with their full path names
train_img = []

for i in img: # img is list of full path names of all images
    if i[len(images):] in train_images: # Check if the image belongs to training set
        train_img.append(i) # Add it to the list of train images

# File chứa các id ảnh để test
test_images_file = 'Flickr8k/Flickr8k_text/Flickr_8k.testImages.txt'
# Read the validation image names in a set# Read the test image names in a set
test_images = set(open(test_images_file, 'r').read().strip().split('\n'))

# Create a list of all the test images with their full path names
test_img = []

for i in img: # img is list of full path names of all images
    if i[len(images):] in test_images: # Check if the image belongs to test set
        test_img.append(i) # Add it to the list of test images

# Thêm 'startseq', 'endseq' cho chuỗi
def load_clean_descriptions(filename, dataset):
        # load document
        doc = load_doc(filename)
        descriptions = dict()
        for line in doc.split('\n'):
                # split line by white space
                tokens = line.split()
                # split id from description
                image_id, image_desc = tokens[0], tokens[1:]
                # skip images not in the set
                if image_id in dataset:
                        # create list
                        if image_id not in descriptions:
                                descriptions[image_id] = list()
                        # wrap description in tokens
                        desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
```

```
                        # store
                        descriptions[image_id].append(desc)
        return descriptions

# descriptions
train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))

# Load ảnh, resize về khích thước mà Inception v3 yêu cầu.
def preprocess(image_path):
    # Convert all the images to size 299x299 as expected by the inception v3 model
    img = image.load_img(image_path, target_size=(299, 299))
    # Convert PIL image to numpy array of 3-dimensions
    x = image.img_to_array(img)
    # Add one more dimension
    x = np.expand_dims(x, axis=0)
    # preprocess the images using preprocess_input() from inception module
    x = preprocess_input(x)
    return x

# Load the inception v3 model
model = InceptionV3(weights='imagenet')

# Tạo model mới, bỏ layer cuối từ inception v3
model_new = Model(model.input, model.layers[-2].output)

# Image embedding thành vector (2048, )
def encode(image):
    image = preprocess(image) # preprocess the image
    fea_vec = model_new.predict(image) # Get the encoding vector for the image
    fea_vec = np.reshape(fea_vec, fea_vec.shape[1]) # reshape from (1, 2048) to (2048, )
    return fea_vec

# Gọi hàm encode với các ảnh trong training set
start = time()
encoding_train = {}
for img in train_img:
    encoding_train[img[len(images):]] = encode(img)
print("Time taken in seconds =", time()-start)

# Lưu image embedding lại
with open("Flickr8k/Pickle/encoded_train_images.pkl", "wb") as encoded_pickle:
    dump(encoding_train, encoded_pickle)

# Encode test image
start = time()
encoding_test = {}
for img in test_img:
    encoding_test[img[len(images):]] = encode(img)
```

```
print("Time taken in seconds =", time()-start)

# Save the bottleneck test features to disk
with open("Flickr8k/Pickle/encoded_test_images.pkl", "wb") as encoded_pickle:
    dump(encoding_test, encoded_pickle)

train_features = load(open("Flickr8k/Pickle/encoded_train_images.pkl", "rb"))
print('Photos: train=%d' % len(train_features))

# Tạo list các training caption
all_train_captions = []
for key, val in train_descriptions.items():
    for cap in val:
        all_train_captions.append(cap)
len(all_train_captions)

# Chỉ lấy các từ xuất hiện trên 10 lần
word_count_threshold = 10
word_counts = {}
nsents = 0
for sent in all_train_captions:
    nsents += 1
    for w in sent.split(' '):
        word_counts[w] = word_counts.get(w, 0) + 1

vocab = [w for w in word_counts if word_counts[w] >= word_count_threshold]
print('preprocessed words %d -> %d' % (len(word_counts), len(vocab)))

ixtoword = {}
wordtoix = {}

ix = 1
for w in vocab:
    wordtoix[w] = ix
    ixtoword[ix] = w
    ix += 1

vocab_size = len(ixtoword) + 1 # Thêm 1 cho từ dừng để padding
vocab_size

# convert a dictionary of clean descriptions to a list of descriptions
def to_lines(descriptions):
        all_desc = list()
        for key in descriptions.keys():
                [all_desc.append(d) for d in descriptions[key]]
        return all_desc

# calculate the length of the description with the most words
def max_length(descriptions):
```

```
        lines = to_lines(descriptions)
        return max(len(d.split()) for d in lines)

# determine the maximum sequence length
max_length = max_length(train_descriptions)
print('Description Length: %d' % max_length)

# data generator cho việc train theo từng batch model.fit_generator()
def data_generator(descriptions, photos, wordtoix, max_length, num_photos_per_batch):
    X1, X2, y = list(), list(), list()
    n=0
    # loop for ever over images
    while 1:
        for key, desc_list in descriptions.items():
            n+=1
            # retrieve the photo feature
            photo = photos[key+'.jpg']
            for desc in desc_list:
                # encode the sequence
                seq = [wordtoix[word] for word in desc.split(' ') if word in wordtoix]
                # split one sequence into multiple X, y pairs
                for i in range(1, len(seq)):
                    # split into input and output pair
                    in_seq, out_seq = seq[:i], seq[i]
                    # pad input sequence
                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                    # encode output sequence
                    out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                    # store
                    X1.append(photo)
                    X2.append(in_seq)
                    y.append(out_seq)
            # yield the batch data
            if n==num_photos_per_batch:
                yield [[array(X1), array(X2)], array(y)]
                X1, X2, y = list(), list(), list()
                n=0

# Load Glove model
glove_dir = ''
embeddings_index = {} # empty dictionary
f = open(os.path.join(glove_dir, 'glove.6B.200d.txt'), encoding="utf-8")

for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
```

```
print('Found %s word vectors.' % len(embeddings_index))

embeddings_index['the']

embedding_dim = 200

# Get 200-dim dense vector for each of the 10000 words in out vocabulary
embedding_matrix = np.zeros((vocab_size, embedding_dim))

for word, i in wordtoix.items():
    #if i < max_words:
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # Words not found in the embedding index will be all zeros
        embedding_matrix[i] = embedding_vector

embedding_matrix.shape

# Tạo model
inputs1 = Input(shape=(2048,))
fe1 = Dropout(0.5)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, embedding_dim, mask_zero=True)(inputs2)
se2 = Dropout(0.5)(se1)
se3 = LSTM(256)(se2)
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)
model = Model(inputs=[inputs1, inputs2], outputs=outputs)

model.summary()

# Layer 2 dùng GLOVE Model nên set weight thẳng và không cần train
model.layers[2].set_weights([embedding_matrix])
model.layers[2].trainable = False

model.compile(loss='categorical_crossentropy', optimizer='adam')

model.optimizer.lr = 0.0001
epochs = 10
number_pics_per_bath = 6
steps = len(train_descriptions)//number_pics_per_bath

for i in range(epochs):
    generator = data_generator(train_descriptions, train_features, wordtoix, max_length,
    model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)

model.save_weights('./model_weights/model_30.h5')
```

```
images = 'Flickr8k/Flicker8k_Dataset/'

with open("Flickr8k/Pickle/encoded_test_images.pkl", "rb") as encoded_pickle:
    encoding_test = load(encoded_pickle)

# Với mỗi ảnh mới khi test, ta sẽ bắt đầu chuỗi với 'startseq' rồi
# sau đó cho vào model để dự đoán từ tiếp theo.
# Ta thêm từ vừa được dự đoán vào chuỗi và tiếp tục cho đến khi gặp 'endseq'
# là kết thúc hoặc cho đến khi chuỗi dài 34 từ.
def greedySearch(photo):
    in_text = 'startseq'
    for i in range(max_length):
        sequence = [wordtoix[w] for w in in_text.split() if w in wordtoix]
        sequence = pad_sequences([sequence], maxlen=max_length)
        yhat = model.predict([photo,sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = ixtoword[yhat]
        in_text += ' ' + word
        if word == 'endseq':
            break
    final = in_text.split()
    final = final[1:-1]
    final = ' '.join(final)
    return final

z=5
pic = list(encoding_test.keys())[z]
image = encoding_test[pic].reshape((1,2048))
x=plt.imread(images+pic)
plt.imshow(x)
plt.show()
print(greedySearch(image))
```

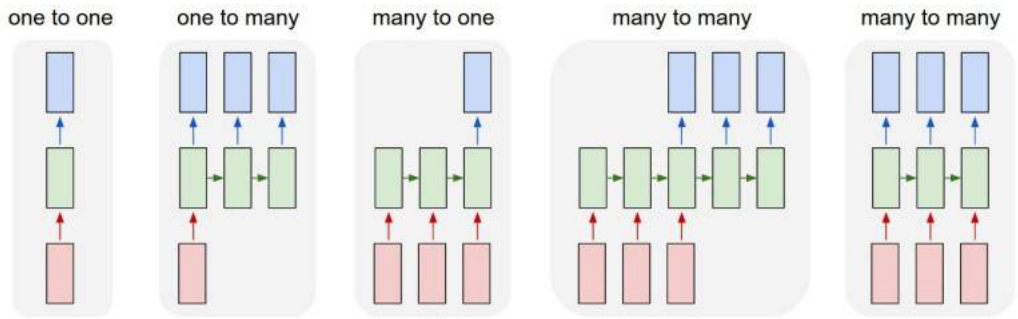

Hình 18.13: Các caption dự đoán từ model

Hình 18.2: Ảnh trong Flickr8k dataset

# 19. Seq2seq và attention

## 19.1 Giới thiệu

Mô hình RNN ra đời để xử lý các dữ liệu dạng chuỗi (sequence) như text, video.




Hình 19.1: Các dạng bài toán RNN

Bài toán RNN được phân làm một số dạng:

- **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 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.
- **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.

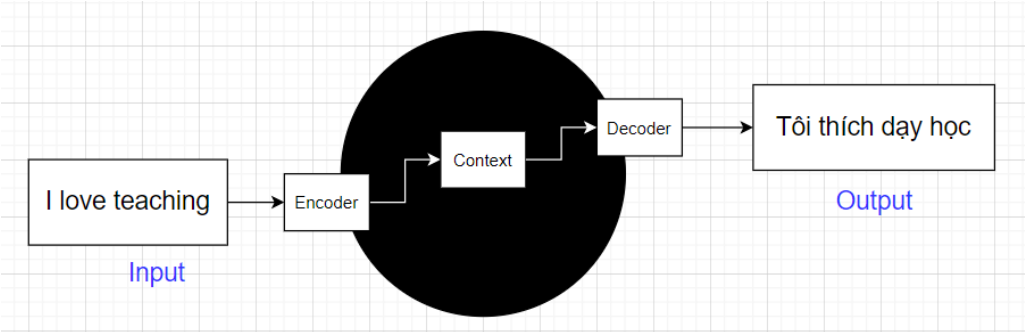
Mô hình sequence to sequence (seq2seq) sinh ra để giải quyết bài toán many to many và rất thành công trong các bài toán: dịch, tóm tắt đoạn văn. Bài này mình sẽ cùng tìm hiểu về mô hình seq2seq với bài toán dịch từ tiếng anh sang tiếng việt.

## 19.2 Mô hình seq2seq

Input của mô hình seq2seq là một câu tiếng anh và output là câu dịch tiếng việt tương ứng, độ dài hai câu này có thể khác nhau. Ví dụ: input: I love teaching -> output: Tôi thích dạy học, input 1 câu 3 từ, output 1 câu 4 từ.


Hình 19.2: Seq2seq model

Mô hình seq2seq gồm 2 thành phần là encoder và decoder.

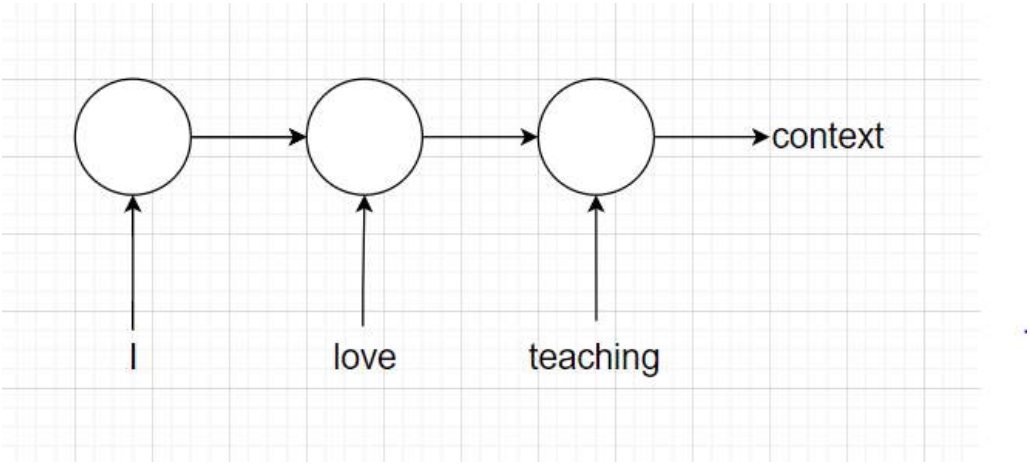



Hình 19.3: Seq2seq model

Encoder nhận input là câu tiếng anh và output ra context vector, còn decoder nhận input là context vector và output ra câu tiếng việt tương ứng. Phần encoder sử dụng mô hình RNN (nói là mô hình RNN nhưng có thể là các mô hình cải tiến như GRU, LSTM) và context vector được dùng là hidden states ở node cuối cùng. Phần decoder cũng là một mô hình RNN với $s_0$ chính là context vector rồi dần dần sinh ra các từ ở câu dịch.

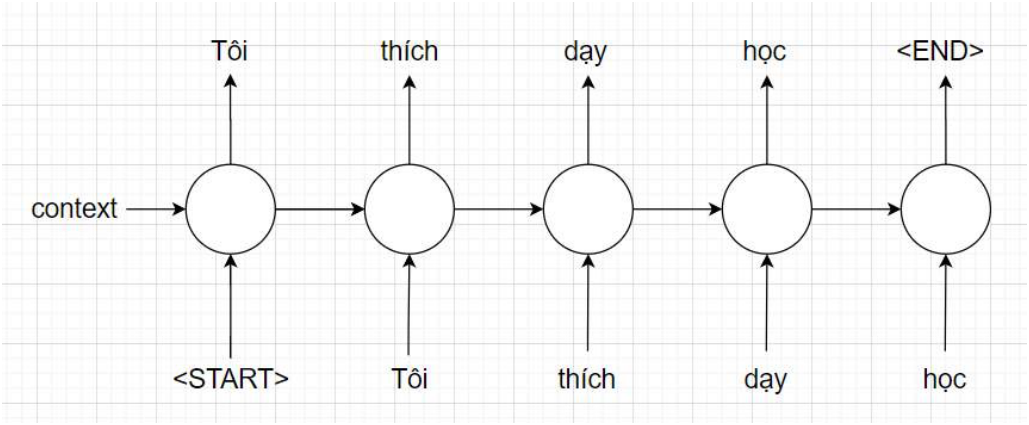
Phần decoder này giống với bài toán image captioning. Ở bài image captioning mình cũng cho ảnh qua pre-trained model để lấy được embedding vector, sau đó cho embedding vector làm $s_0$ của mô hình RNN rồi sinh ta caption tương ứng với ảnh.

Bài trước mình đã biết model RNN chỉ nhận input dạng vector nên dữ liệu ảnh (từ) sẽ được encode về dạng vector trước khi cho vào model.




Hình 19.4: Mô hình encoder

Các từ trong câu tiếng anh sẽ được embedding thành vector và cho vào mô hình RNN, hidden state ở node cuối cùng sẽ được dùng làm context vector. Về mặt lý thuyết thì context vector sẽ mang đủ thông tin của câu tiếng anh cần dịch và sẽ được làm input cho decoder.




Hình 19.5: Mô hình decoder

2 tag <START> và <END> được thêm vào câu output để chỉ từ bắt đầu và kết thúc của câu dịch. Mô hình decoder nhận input là context vector. Ở node đầu tiên context vector và tag <START> sẽ output ra chữ đầu tiên trong câu dịch, rồi tiếp tục mô hình sinh chữ tiếp theo cho đến khi gặp tag <END> hoặc đến max_length của câu output thì dừng lại.

Ví dụ code seq2seq cho bài toán dịch với mô hình LSTM mọi người tham khảo ở đây.

**Vấn đề**: Mô hình seq2seq encode cả câu tiếng anh thành 1 context vector, rồi dùng context vector để sinh ra các từ trong câu dịch tưng ứng tiếng việt. Như vậy khi câu dài thì rất khó cho decoder chỉ dùng 1 context vector có thể sinh ra được câu output chuẩn. Thêm vào đó các mô hình RNN đều bị mất ít nhiều thông tin ở các node ở xa nên bản thân context vector cũng khó để học được thông tin ở các từ ở phần đầu của encoder.

=> Cần có cơ chế để lấy được thông tin các từ ở input cho mỗi từ cần dự đoán ở ouptut thay vì chỉ dựa vào context vector => **Attention** ra đời.

## 19.3 Cơ chế attention

### 19.3.1 Motivation

Attention tiếng anh nghĩa là chú ý, hay tập trung. Khi dịch ở mỗi từ tiếng việt ta cần chú ý đến 1 vài từ tiếng anh ở input, hay nói cách khác là có 1 vài từ ở input có ảnh hưởng lớn hơn để dịch từ đấy.

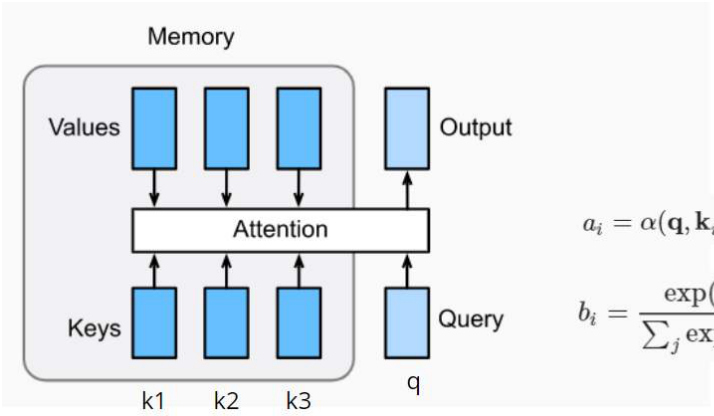

Hình 19.6: Dịch tiếng anh sang tiếng việt, độ quan trọng các từ khi dịch

Ta thấy từ I có trọng số ảnh hưởng lớn tới việc dịch từ tôi, hay từ teaching có ảnh hưởng nhiều tới việc dịch từ dạy và từ học.

=> Do đó khi dịch mỗi từ ta cần chú ý đến các từ ở câu input tiếng anh và đánh trọng số khác nhau cho các từ để dịch chuẩn hơn.

### 19.3.2 Cách hoạt động

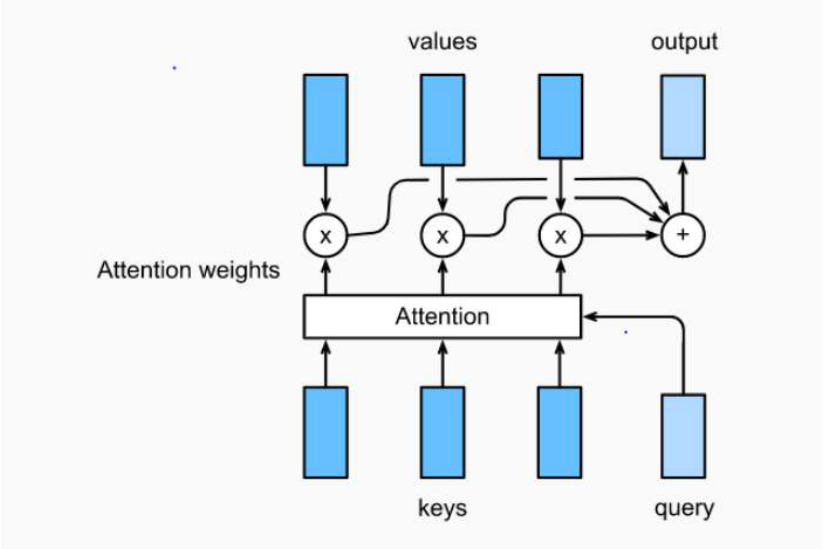
Attention sẽ định nghĩa ra 3 thành phần query, key, value.




Hình 19.7: Các thành phần attention

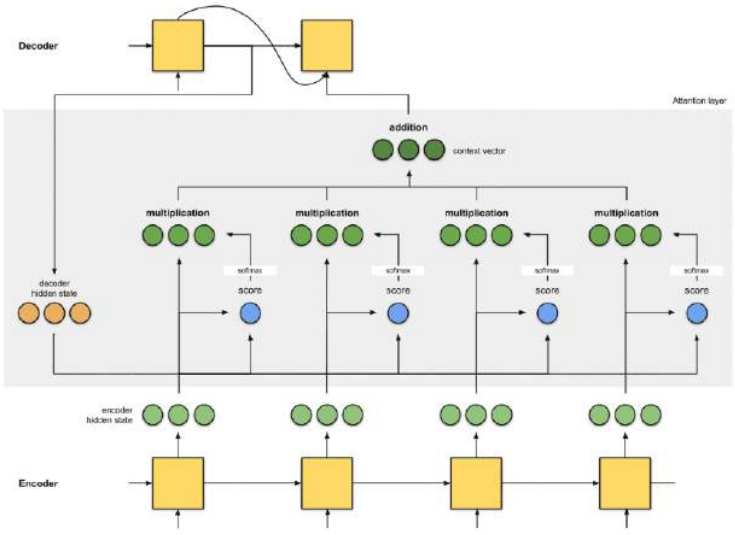
Query (q) lấy thông tin từ từ tiếp theo cần dịch (ví dụ từ dạy). Mỗi từ trong câu input tiếng anh sẽ cho ra 2 thành phần tương ứng là key và value, từ thứ i kí hiệu là $k_i$, $v_i$.

Mỗi bộ q, $k_i$ qua hàm $\alpha$ sẽ cho ra $a_i$ tương ứng, $a_i$ chính là độ ảnh hưởng của từ thứ i trong input lên từ cần dự đoán. Sau đó các giá trị $a_i$ được normalize theo hàm softmax được $b_i$.




Hình 19.8: Các thành phần attention

Cuối cùng các giá trị $v_i$ được tính tổng lại theo hệ số $b_i$, output = $\sum_{i=1}^{N} b_i * v_i$, trong đó N là số từ trong câu input. Việc normalize các giá trị $a_i$ giúp output cùng scale với các giá trị value.




Hình 19.9: Các bước trong attention

Ở phần encoder, thông thường mỗi từ ở input thì hidden state ở mỗi node được lấy làm cả giá trị key và value của từ đấy. Ở phần decoder, ở node 1 gọi input là $x_1$, output $y_1$ và hidden state $s_1$; ở node 2 gọi input là $x_2$, output $y_2$. Query là hidden state của node trước của node cần dự đoán từ tiếp theo ($s_1$). Các bước thực hiện:

- Tính score: $a_i = \alpha(q, k_i)$
- Normalize score: $b_i$

- Tính output: output_attention = $\sum_{i=1}^{N} b_i * v_i$
- Sau đó kết hợp hidden state ở node trước $s_1$, input node hiện tại $x_2$ và giá trị output_attention để dự đoán từ tiếp theo $y_2$.

| Name | Alignment score function |
|---|---|
| Content-base attention | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$ |
| Additive(*) | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$ |
| Location-Base | $\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$<br>Note: This simplifies the softmax alignment to only depend on the target position. |
| General | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$<br>where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer. |
| Dot-Product | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$ |
| Scaled Dot-Product(^) | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$<br>Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state. |

Hình 19.10: Một số hàm $\alpha$ hay được sử dụng

Nhận xét: Cơ chế attention không chỉ dùng context vector mà còn sử dụng hidden state ở từng từ trong input với trọng số ảnh hưởng tương ứng, nên việc dự đoán từ tiếp theo sẽ tốt hơn cũng như không sợ tình trạng từ ở xa bị mất thông tin ở context vector.

Ngoài ra các mô hình deep learning hay bị nói là hộp đen (black box) vì mô hình không giải thích được, attention phần nào giúp visualize được kết quả dự đoán, ví dụ từ nào ở ouptut ảnh hưởng nhiều bởi từ nào trong input. Do đó model học được quan hệ giữa các từ trong input và output để đưa ra kết quả dự đoán.

Lúc đầu cơ chế attention được dùng trong bài toán seq2seq, về sau do ý tưởng attention quá hay nên được dùng trong rất nhiều bài toán khác, ví dụ như trong CNN người ta dùng cơ chế attention để xem pixel nào quan trọng đến việc dự đoán, feature map nào quan trọng hơn trong CNN layer,.. Giống như resnet, attention cũng là 1 đột phá trong deep learning. Mọi người để ý thì các mô hình mới hiện tại đều sử dụng cơ chế attention.