

Tổng hợp kiến thức về Deep Learning cho môn học (Các kỹ thuật học sâu và ứng dụng – CS431.L21.KHCL)

NGÔ ĐỨC TUẤN – MSSV: 18520186

GV: TS. NGUYỄN VINH TIỆP

Lưu ý: Đây là các kiến thức về Deep Learning (học sâu) do mình tổng hợp lại trong quá trình học để giúp mình củng cố kiến thức và chia sẻ đến mọi người, có thể không tránh khỏi những sai sót và hổng kiến thức ở một vài chỗ. Xin thông cảm và cảm ơn.

Câu 1: Hãy cho biết triết lý của Deep Learning?

AI (*Artificial Intelligence*), được đề cập từ năm 1955, là 1 nhánh tổng quát chung cho toàn bộ Computer Science, có thể dùng nhiều phương pháp, không quan tâm bên dưới sử dụng các kỹ thuật nào, miễn là mô phỏng được những tri thức của con người thành những hướng tiếp cận heuristics dựa trên kinh nghiệm của con người vào môi trường.

ML (*Machine Learning*), là nhánh con của AI, học trên dữ liệu có sẵn trên thực tế để rút trích được những quy luật tri thức nhằm đưa ra các phán đoán trong tương lai.

DL (*Deep Learning*), là nhánh con của ML, là kỹ thuật học từ rất nhiều dữ liệu dựa trên mạng neural nhân tạo (ANNs), để tìm cách biểu diễn đặc trưng hiệu quả, tối ưu. Về cơ bản, DL cho phép ML giải quyết 1 loạt các vấn đề phức tạp mới, chẳng hạn nhận dạng ảnh, ngôn ngữ và lời nói, bằng cách cho phép máy móc tìm hiểu các đặc trưng trong dữ liệu, kết hợp thành các dạng trừu tượng ngày càng cao hơn.

Câu 2: Các mô hình Deep Learning suy cho cùng dựa trên nguyên lý chung nào?

Các mô hình DL suy cho cùng dựa trên nguyên lý là sự lan truyền và biến đổi thông tin thành những đặc trưng nhiều cấp độ khác nhau từ low-level đến high-level (feed forward và backpropagation).

Câu 3: Các vấn đề nghiên cứu Deep Learning mà người ta đang phải giải quyết?

a) *Overfitting*: số lượng tham số quá nhiều. Giải pháp:





- + Giảm số tham số.

- + Dùng mini-batch -> dùng một 1 phần data trong tập training cho mỗi bước tính đạo hàm Gradient.

- + Dùng dropout (rate % tỉ lệ drop): mỗi lần update hệ số, tác dụng đơn giản hóa model -> model đơn giản với bài toán.

- + Trong CNN thì chỉ lấy các filters dùng depthwise.

b) *Cần nguồn data rất lớn.* Giải pháp là ta tự thu thập nguồn dữ liệu public hoặc ta có thể sinh thêm dữ liệu (data augmentation) với các giải pháp transformation như là scale up, scale down, rotation, translation, thay đổi lightning, thêm noise,...

Original	Flip	Rotation	Random crop	Color shift	Noise addition	Information loss	Contrast change
							
• Image without any modification	• Flipped with respect to an axis for which the meaning of the image is preserved	• Rotation with a slight angle • Simulates incorrect horizon calibration	• Random focus on one part of the image • Several random crops be done in a row	• Nuances of RGB is slightly changed • Captures noise that can occur with light exposure	• Addition of noise • More tolerance to quality variation of inputs	• Parts of image ignored • Mimics potential loss of parts of image	• Luminosity changes • Controls difference in exposition due to time of day

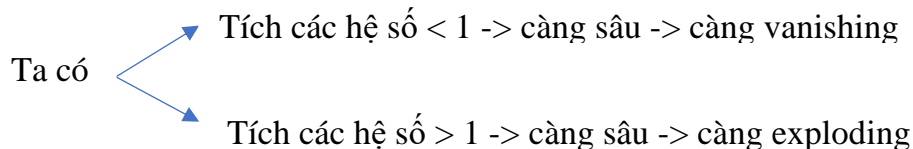
Hình 1: Data augmentation cho việc sinh thêm dữ liệu

c) *Tài nguyên tính toán lớn do các mô hình DL chạy trên GPU.* Giải pháp là ta có thể tự trang bị thêm máy móc, sử dụng các IDE bên ngoài có sẵn GPU như Google Colab. Ngoài ra ta sử dụng Transfer Learning, tức là dùng pre-trained model bao gồm:

+ Feature extraction: lấy làm input cho model mới (lấy feature mới làm input).

+ Fine tune: sử dụng pre-trained model, tận dụng 1 phần hay toàn bộ các layer, thêm / xóa giữa 1 vài layers -> được model mới, các layers đầu được đóng băng lại giữ lại weights (trọng số) -> tận dụng khả năng học trước đó.

d) *Vanishing gradient, exploding gradient:* Backpropagation là thuật toán dùng để tính đạo hàm các hệ số trong ML với loss function để áp dụng gradient.



=> Khởi tạo hệ số phù hợp, activation function phù hợp, sử dụng batch normalization để giảm sự ảnh hưởng của việc khởi tạo tham số ban đầu. Ngoài ra trong CNN có thể giải quyết bằng kết nối tắt (LeNet, Inception),...trong RNN thay các node RNN bằng cơ chế “cổng” (gate) (LSTM, GRU),...

Câu 4: Hãy cho biết khái niệm, chức năng, kiến trúc mạng của CNN.

CNN (Convolutional Neural Network) thường được dùng cho dữ liệu ảnh, loại dữ liệu có hai trục không gian (cao x rộng), được dùng để rút trích đặc trưng ảnh.

Filter: là bộ lọc, là 1 ma trận các số được sử dụng để biểu diễn 1 toán tử (operator). Các bộ lọc khác nhau biểu diễn các bài toán khác nhau. Input là ảnh chưa xử lý (gốc) sau đó qua bộ lọc, thu được output là ảnh mới được xử lý (các thông số có thể khác ảnh gốc).

Filtering: là kỹ thuật để áp dụng toán tử vào ảnh số (digital image). Filtering còn được gọi là “mask”, “kernel” hay “feature detect”.

Mục tiêu của filtering là rút trích (extract) / xử lý (process) thông tin từ ảnh. Một số ví dụ về ứng dụng của filtering: làm nét (shapen), làm mờ (blur), giảm nhiễu (noise reduction), tương phản (contrast),...

Để áp dụng được filter lên ảnh, ta có 2 phương pháp: cross-correlation và convolution.

Cross-correlation (tương quan chéo): Giả sử F là ảnh, H là kernel (size $2k+1 \times 2k+1$), G là đầu ra của ảnh.

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i + u, j + v]$$

Phép toán tương quan chéo: $G = H \otimes F$

Convolution (Tích chập): tương tự như cross-correlation, ngoại trừ kernel bị lật ngược (flipped) theo chiều ngang và chiều dọc (horizontally and vertically).

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i - u, j - v]$$

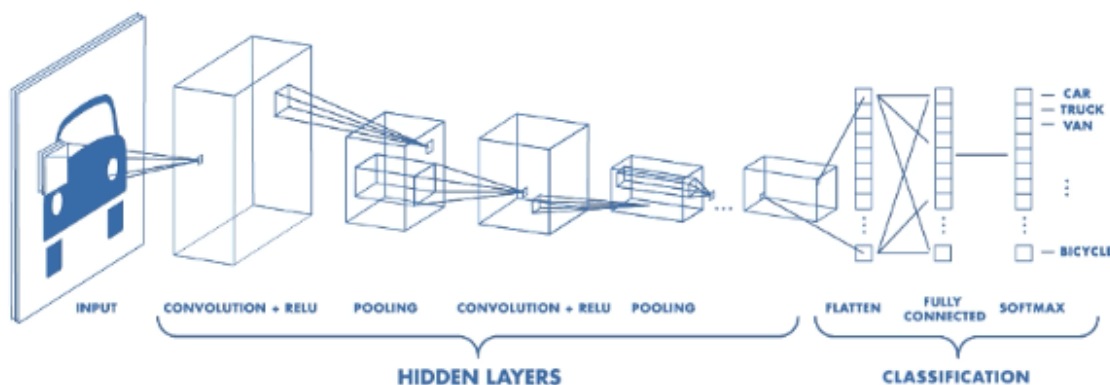
Phép toán tích chập: $G = H * F$

Convolution có tính chất giao hoán (commutative) và kết hợp (associative).

Phép convolutional này là 1 loại cửa sổ trượt (sliding window) nằm trên 1 ma trận. Những convolutional layer sẽ có các parameters được học để điều chỉnh và lấy ra những thông tin chính xác nhất mà không cần phải chọn feature. Convolution chính là nhân các phần tử trong ma trận sliding window (filters) và là loại ma trận có kích thước nhỏ.

Feature: là đặc trưng, các CNN sẽ so sánh ảnh dựa theo từng mảnh và những mảnh này được gọi là feature. Thay vì phải khớp các bức ảnh lại với nhau thì CNN sẽ nhìn ra sự tương đồng khi tìm kiếm các feature khớp với nhau bằng 2 hình ảnh tốt hơn.

Về kiến trúc mạng CNN



Hình 2: Kiến trúc mạng CNN

- Tầng convolution
 - Tầng activation (ReLU)
 - Tầng pooling (optional)
 - Flatten
 - Tầng fully connected
- $\left. \begin{array}{l} \text{Lặp } k \text{ lần} \\ \text{Lặp } n \text{ lần} \end{array} \right\}$

Cụ thể về cách hoạt động của từng tầng:

a) *Tầng convolution*: tìm ra feature map đặc trưng quan trọng cho bài toán. Tầng này sử dụng các bộ lọc (filters, kernels, masks) để thực hiện phép tích chập khi đưa chúng đi qua đầu vào (input) theo các chiều của nó. Những yếu tố quan trọng của 1 convolution layer là: stride, padding, filter map, feature map.

+ CNN dùng các filter để áp dụng vào vùng của ảnh (filter map).

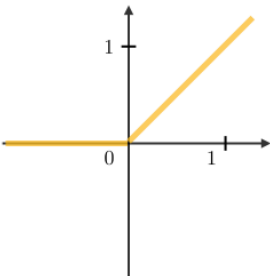
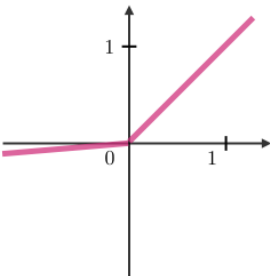
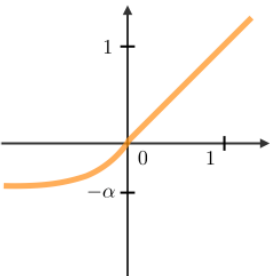
+ Stride: dịch chuyển filter map theo pixel dựa vào giá trị từ trái sang phải.

+ Padding: là các giá trị 0 được thêm vào lớp input để output có thể bằng giá trị input.

+ Feature map: thể hiện kết quả của mỗi lần filter map quét ra input, đây cũng là đầu ra của mạng CNN.

b) *Tầng activation ReLU*: ReLU layer (Tầng rectified linear unit) là hàm kích hoạt trong neural network được sử dụng trên tất cả các thành phần. Trong hàm kích hoạt này thì nó có nghĩa là: ReLU, Leaky ReLU, tanh, sigmoid,...Hiện nay với nhu cầu huấn luyện mạng neural thì hàm ReLU được dùng phổ biến vì ReLU mang lại nhiều ưu điểm nổi bật như khả năng tính toán sẽ nhanh hơn.

Nói về hàm kích hoạt trong neural network, ta có một số loại sau đây:

ReLU	Leaky ReLU	ELU
$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ với $\epsilon \ll 1$	$g(z) = \max(\alpha(e^z - 1), z)$ với $\alpha \ll 1$
		
• Độ phức tạp phi tuyến tính có thể thông dịch được về mặt sinh học	• Gán vấn đề ReLU chết cho những giá trị âm	• Khả vi tại mọi nơi

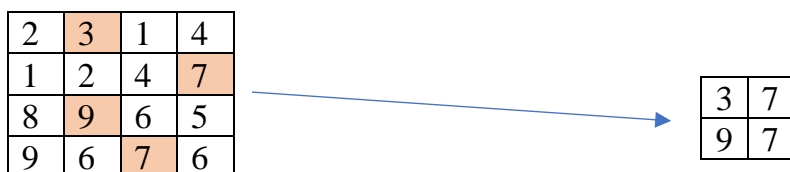
Hình 3: Các biến thể của hàm kích hoạt ReLU

Ngoài ra còn có một hàm kích hoạt là softmax ở sau tầng fully connected cuối cùng và trước khi đưa ra output nhằm đưa ra phân bố xác suất cho bài toán. Bước softmax này được xem là một hàm logistic tổng quát lấy đầu vào là một vector chứa các giá trị $x \in R^n$ và cho ra một vector gồm các xác suất $p \in R^n$.

Lí do ta dùng hàm này vì sau tầng convolution sẽ tới tầng activation do ta kết hợp 1 hàm tuyến tính và 1 hàm phi tuyến nhằm tăng tính phi tuyến tính cho mạng.

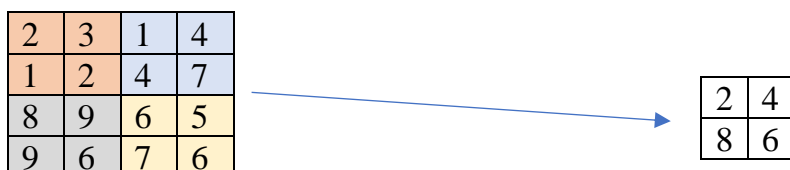
c) *Tầng pooling*: dùng để giảm kích thước dữ liệu (downsampling), thường được dùng sau tầng tích chập, giúp tăng tính bất biến không gian. Khi đầu vào quá lớn, những lớp pooling được xếp vào giữa những lớp Convolution để làm giảm parameter. Hiện nay, có 2 loại pooling là max-pooling (giá trị lớn nhất) và average pooling (giá trị trung bình).

Max pooling: từng phép pooling chọn giá trị lớn nhất trong khu vực mà nó đang được áp dụng. Ví dụ:



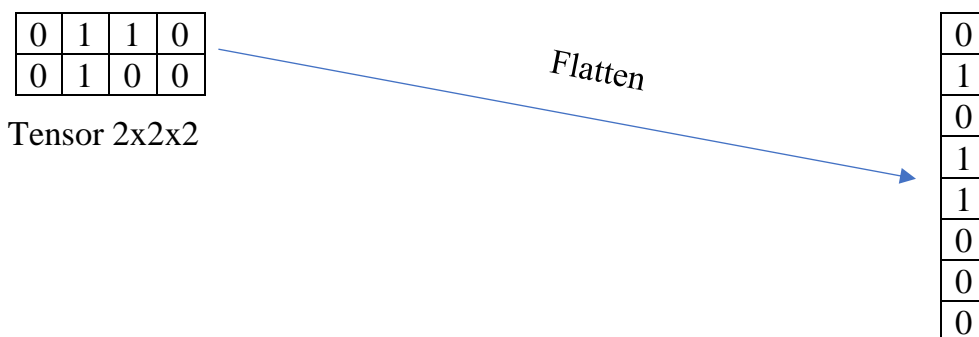
Nhận xét: 1) Bảo toàn được các đặc trưng đã phát hiện. 2) Được sử dụng thường xuyên.

Average pooling: từng phép pooling tính trung bình các giá trị trong khu vực mà nó đang áp dụng. Ví dụ:



Nhận xét: 1) Giảm kích thước feature map. 2) Được sử dụng trong mạng LeNet.

d) *Flatten*: là quá trình “đuỗi” tensor 3D thành tensor 1D để làm đầu vào cho tầng fully connected. Ví dụ:



e) *Tầng fully connected*: lớp này có nhiệm vụ đưa ra kết quả sau khi lớp convolution và lớp pooling đã nhận ra được ảnh truyền. Nó nhận đầu vào là các dữ liệu đã được làm phẳng, mà mỗi đầu vào đó được kết nối đến tất cả neuron. Lúc này thu được kết quả là model đã đọc được thông tin của ảnh để liên kết cũng như cho ra nhiều output hơn thì ta sử dụng lớp này.

Câu 5: Có các loại kiến trúc mạng CNN phổ biến nào?

1) *LeNet (1998)*: convolution + backpropagation. Là kiến trúc đầu tiên áp dụng mạng tích chập 2 chiều, model khá đơn giản chỉ gồm 2 convolutional layers + 3 fully connected layers. Lúc đầu mạng LeNet được áp dụng average-pooling layer cho sub-sampling tuy nhiên do khó hội tụ nên ngày nay được thay thế bằng max-pooling. Đầu vào là ảnh có kích thước nhỏ (32x32), ảnh grayscale nên số lượng tham số của nó cũng ít hơn các mạng sau này.

Xem thêm: Yann LeCun, Leon Bottou, Yoshua Bengio, Patrick Haffner – “Gradient – based learning applied to document recognition”. Submitted in Nov, 1998.

<https://ieeexplore.ieee.org/abstract/document/726791>

2) *Alex Net (2012)*: sigmoid \rightarrow ReLU + GPU + data. Là một sự cải tiến so với LeNet:

+ Tăng kích thước đầu vào và độ sâu của mạng (224x224x3).

+ Sử dụng các bộ lọc (filter, mask, kernel) với kích thước giảm dần qua các layers để phù hợp với kích thước của các features.

+ Sử dụng local normalization để chuẩn hóa các layer giúp quá trình hội tụ nhanh hơn.

+ Lần đầu tiên áp dụng activation là ReLU thay cho sigmoid.

Xem thêm: Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton – “ImageNet Classification with Deep Convolutional Neural Networks”. Submitted in 2012.

<https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>

3) *VGG-16 (Visual Geometry Group / 2014)*: chỉ sử dụng filter 3x3 ở các tầng convolution liên tục.

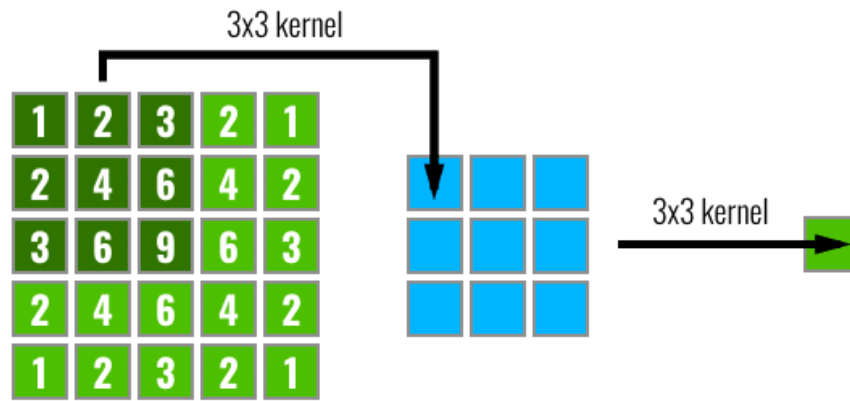
2 phép 3x3 \longleftrightarrow 1 phép 5x5 trong LeNet (về phạm vi
3 phép 3x3 \longleftrightarrow 1 phép 7x7 ảnh hưởng/receptive field)

Ngoài ra, VGG-16 còn có các điểm cải tiến so với AlexNet như:

+ Kiến trúc VGG-16 sâu hơn, gồm 13 convolutional layers + 3 fully connected layers.

+ Lần đầu tiên xuất hiện khái niệm về khối tích chập (block), tức kiến trúc gồm một tập hợp các layers CNN được lặp lại giống nhau.

+ Lần đầu tiên thay đổi thứ tự của các block khi xếp nhiều layers CNN + max pooling thay vì xem kẽ 1 layer CNN + max pooling. Điều này giúp có thể trích lọc đặc trưng tốt hơn.



Hình 4: Receptive field 3x3

Câu hỏi: tại sao dùng filter 3x3 thay vì 5x5? Vì dùng 2 lần filter 3x3 có chiều sâu hơn, sử dụng ít parameters để học hơn, mang lại hiệu quả tính toán hơn (tiết kiệm 72%).

$$\begin{array}{l} 3 \times 3 = 9 \text{ paras cần học} \\ 5 \times 5 = 25 \text{ paras cần học} \end{array} \quad \longrightarrow \quad \frac{2 \times 9}{25} = 72\%$$

Xem thêm: Keren Simonyan, Andrew Zisserman-“Very Deep Convolutional for Large-Scale Image Recognition”. Submitted on Apr 10, 2015. <https://arxiv.org/pdf/1409.1556.pdf>

4) *Google Net / Inception (2014)*: nhiều output hơn + short connection và bổ sung kiến trúc khối Inception gồm 4 nhánh song song, với các bộ lọc có kích thước kernel_size lần lượt là 1x1, 3x3, 5x5 giúp trích lọc được đa dạng đặc trưng trên những vùng có kích thước khác nhau. Ngoài ra, tuy toàn bộ mạng có đến 22 layers tức lớn hơn gần gấp đôi, nhưng nhờ áp dụng tích chập 1x1 giúp tiết kiệm số lượng tham số, ít hơn gần 27 lần so với VGG-16.

Câu hỏi: a) Tại sao cần nhiều output hơn? Có những đối tượng lớn cần nhiều thông tin hơn để quyết định, đối tượng nhỏ cần ít thông tin hơn để quyết định -> sử dụng nhiều output để có thể scale kích thước về cùng loại.

Câu hỏi: b) Khi truyền vào input, cho ra 3 output mâu thuẫn nhau (3 lớp \neq nhau), ta sẽ chọn cái nào? 3 vector 1000 chiều lấy trung bình thành 1 vector 1000 chiều, cuối cùng lấy softmax để ra prediction duy nhất.

Xem thêm: Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich – “Going Deeper with Convolutions”. Submitted on Sep 17, 2014. <https://arxiv.org/abs/1409.4842>

5) *ResNet-50 (2015)*: rút ngắn quá trình backpropagation -> từ loss backpropagation để cập nhật lại, dùng skip connection hoặc shortcut để nhảy qua các lớp.

ResNet là kiến trúc được sử dụng phổ biến nhất ở thời điểm hiện tại. ResNet cũng là kiến trúc sớm nhất áp dụng batch normalization.

Với các mạng trước đây thường cải thiện độ chính xác nhờ tăng chiều sâu của mạng CNN, nhưng đến một ngưỡng độ sâu nào đó thì độ chính xác sẽ bị bão hòa và thậm chí làm cho mô hình kém chính xác hơn, và có thể làm thông tin gốc bị mất đi khi đi qua nhiều tầng độ sâu, do đó ResNet đã giải quyết vấn đề này bằng kết nối tắt (skip connection). Điều này giúp giữ thông tin không bị mất bằng cách kết nối từ layer sớm trước đó tới layer phía sau và bỏ qua một vài layers trung gian.

ResNet có khối tích chập (Convolutional Block) sử dụng bộ lọc có kích thước kernel_size 3x3 giống Google Net. Khối tích chập gồm 2 nhánh tích chập trong đó một nhánh áp dụng tích chập 1x1 trước khi cộng trực tiếp vào nhánh còn lại.

Ngoài ra, ResNet còn có khối xác định (Identity block), nó không áp dụng tích chập 1x1 mà cộng trực tiếp giá trị của nhánh đó vào nhánh còn lại.

Xem thêm: Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun – “Deep Residual Learning for Image Recognition”. Submitted on Dec 10, 2015.

<https://arxiv.org/pdf/1512.03385.pdf>

6) *MobileNet (2017)*: depthwise 3x3 + 1x1 convolution. Điểm cải tiến của mô hình này là việc sử dụng một cách tính tích chập có tên là Depthwise Separable Convolution để giảm kích thước mô hình và giảm độ phức tạp tính toán. Do đó, mô hình sẽ hữu ích khi chạy các ứng dụng trên mobile và các thiết bị nhúng, do đó gọi là MobileNet.

Câu hỏi: Tại sao lại thay 3x3 bình thường thành depthwise 3x3 + 1x1 convolution? Liệu 3x3 có tương đương depthwise 3x3 + 1x1 convolution về receptive field không?

Điểm cải tiến của MobileNet đó là

Mô hình có ít tham số hơn -> kích thước model nhỏ hơn.

Mô hình có ít phép cộng trừ nhân chia hơn -> độ phức tạp nhỏ hơn và chi phí sẽ giảm $\frac{1}{N} + \frac{1}{D_k^2}$. Về cách tính chi phí tính toán giảm bao nhiêu, ta cần quan tâm đến 2 khối sau: depthwise convolution và pointwise convolution. Với M là số lượng input channel, N là số lượng output channel, D_k là kernel_size, D_f là feature map size = 224.

+ Chi phí tính toán của depthwise convolution: $D_k \cdot D_k \cdot M \cdot D_f \cdot D_f$

+ Chi phí tính toán của pointwise convolution: $M \cdot N \cdot D_f \cdot D_f$

+ Tổng chi phí tính toán của depthwise separable convolution:

$$D_k \cdot D_k \cdot M \cdot D_f \cdot D_f + M \cdot N \cdot D_f \cdot D_f$$

+ Nếu ta không sử dụng depthwise separable convolution mà sử dụng phép convolution thông thường, chi phí tính toán là: $D_k \cdot D_k \cdot M \cdot N \cdot D_f \cdot D_f$

+ Do đó, chi phí tính toán sẽ giảm:
$$\frac{D_k \cdot D_k \cdot M \cdot D_f \cdot D_f + M \cdot N D_f \cdot D_f}{D_k \cdot D_k \cdot M \cdot N \cdot D_f \cdot D_f} = \frac{1}{N} + \frac{1}{D_k^2}$$

Giả sử, chúng ta chọn kernel_size $D_k = 3$, ta sẽ giảm được từ 8 đến 9 lần phép nhân.

Xem thêm: Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam – “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. Submitted on Apr 17, 2017. <https://arxiv.org/pdf/1704.04861.pdf>

Ngoài ra, để biết thêm các kiến trúc mạng, diagram và code cụ thể của các mạng CNNs, các bạn có thể tham khảo thêm tại đây.

<https://github.com/Machine-Learning-Tokyo/CNN-Architectures>

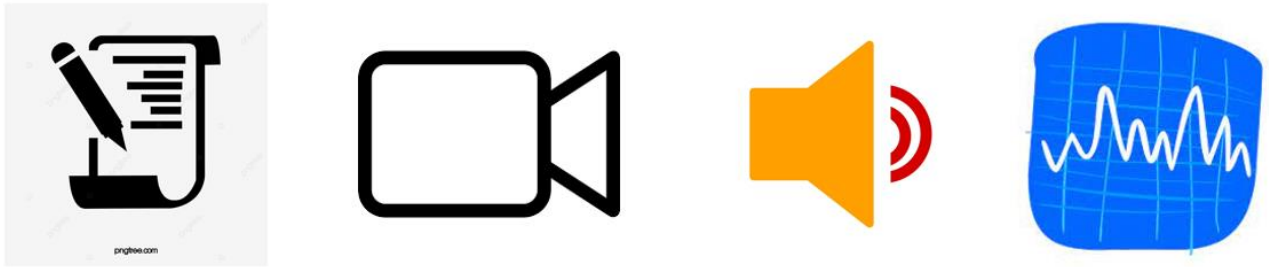
Câu 6: 7 phương thức phải có trong Deep Learning?

class ModelName:	#Test mô hình
#Khởi tạo 1 số tham số chính của mô hình	def test (self):
def _init_(self):	return None
return None	#Giúp debug và in ra kiến trúc của mô hình
#Xây dựng kiến trúc của mô hình	def summary (self):
def build (self):	return None
return None	#Lưu mô hình xuống file
#Phương thức để đưa dữ liệu vào và huấn luyện	def save (self):
def train (self):	return None
return None	#Load mô hình từ file
	def load (self):
	return None

Câu 7: Hãy cho biết kiến trúc mạng RNN. Nhược điểm và cách khắc phục?

RNN (Recurrent Neural Network) là mạng phù hợp cho các bài toán có yếu tố phụ thuộc theo chuỗi thời gian, loại dữ liệu chuỗi 1 chiều mà có tính tạm thời (temporal

component), tại thời điểm người ta đưa ra quyết định thì nó cần phải có đầy đủ thông tin của các thời điểm trong quá khứ $t-1, t-2, t-3, \dots$ và cả trong tương lai $t+1, t+2, t+3, \dots$.



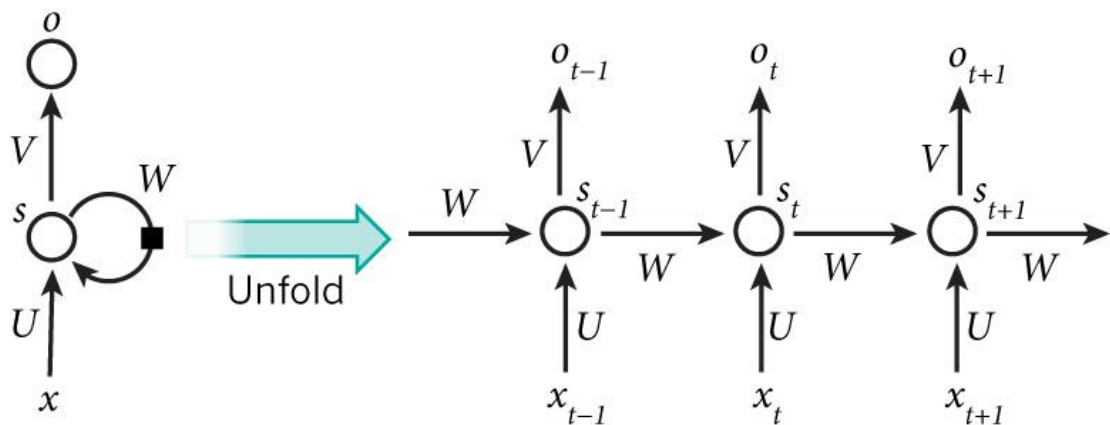
Hình 5: 1 số thông tin dạng chuỗi

Những loại dữ liệu có yếu tố chuỗi thời gian: video, âm thanh, văn bản, thời tiết (có yếu tố tuần hoàn theo mùa), giá vàng, cổ phiếu,...

Một số ứng dụng của RNN: dịch máy (machine translation), phân tích cảm xúc (sentiment analysis), nhận dạng giọng nói (speech recognition),...

Về kiến trúc mạng RNN

Như đã đề cập thì ý tưởng chính của RNN là sử dụng chuỗi các thông tin RNN được gọi là Recurrent bởi lẽ chúng thực hiện cùng một tác vụ cho tất cả các phần tử của một chuỗi với đầu ra phụ thuộc vào các phép tính trước đó. Nói cách khác thì RNN có khả năng nhớ các thông tin được tính toán từ trước.



Hình 6: Kiến trúc mạng RNN

Bên trái là dạng rút gọn của RNN và bên phải là dạng đầy đủ của RNN. Cụ thể thì x_t được gọi là thời điểm hiện tại, là đầu vào tại bước t . Thí dụ ta có câu “Nhiều tiền để làm gì?” thì x_0 vector ứng với từ “nhiều”, x_1 là vector ứng với từ “tiền”.

s_t là trạng thái ẩn tại bước t . Nó chính là bộ nhớ của mạng này. s_t được tính toán dựa trên tất cả các trạng thái ẩn phía trước và đầu vào tại bước đó. Công thức của s_t là:

$$S_t = f(W * s_{t-1} + U * x_t)$$

Thông thường hàm f này là một hàm phi tuyến như tanh hay ReLU hay sigmoid. Do đó công thức này thường sẽ là:

$$S_t = \text{sigmoid}(W * s_{t-1} + U * x_t)$$

Để làm phép toán cho phần tử ẩn đầu tiên ta cần khởi tạo thêm s_{-1} , thường giá trị khởi tạo này được gán bằng 0.

o_t là đầu ra tại bước t . Để biết được output o trong hiện tại thì phải biết được thông tin trong quá khứ s_{t-1} và thời điểm hiện tại x_t . Ví dụ ta muốn dự đoán từ tiếp theo có thể xuất hiện trong câu thì o_t chính là một vector xác suất các từ trong danh sách từ vựng. Vì ta đưa về dạng phân bố xác suất nên sẽ dùng hàm softmax. Công thức là

$$o_t = \text{softmax}(V * s_t)$$

Còn 3 ma trận U , V , W là 3 tham số trong RNN. Cơ chế của RNN là share-weight do đó U , V , W dùng chung theo thời gian.

Nhược điểm

Việc huấn luyện mạng RNN cũng tương tự như các mạng neural truyền thống, tuy nhiên giải thuật lan truyền ngược (backpropagation) phải đổi một chút. Đạo hàm tại mỗi đầu ra phụ thuộc không chỉ vào các tính toán tại bước đó, mà còn phụ thuộc vào các bước trước đó nữa, vì các tham số trong mạng RNN được sử dụng chung cho tất cả các bước trong mạng. Ví dụ để tính đạo hàm tại $t = 4$ ta phải lan truyền ngược cả 3 bước trước đó rồi cộng tổng các đạo hàm của chúng lại với nhau. Việc tính đạo hàm kiểu này được gọi là lan truyền ngược liên hồi (Backpropagation Through Time).

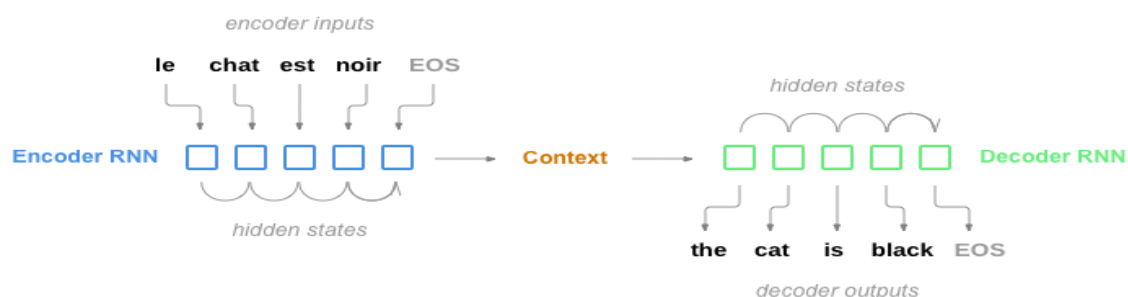
Khi một từ cách chuỗi rất xa bị phụ thuộc vào từ ở đầu câu. Ta muốn tính thời điểm hiện tại o_t thì phải phụ thuộc vào các thời điểm quá khứ s_{t-1}, s_{t-2}, \dots . Khi mà chuỗi phụ thuộc càng dài thì số phép tính cũng dài theo gây ra việc học sẽ càng khó khăn hơn (gặp vấn đề vanishing gradient). Ví dụ có $t = 10$ thì số phép tính sẽ ≈ 40 phép. Ta lấy đạo hàm của hàm hợp này sẽ ra những số bé nhân nhau liên tục. Ta có $1 \text{ số} < 1$ khi mũ 40 lần $\rightarrow 0$ (vanishing gradient) \Rightarrow tức là không cập nhật được trọng số. Do đó khi chuỗi này càng lớn, t mà càng lớn thì hàm lượng thông tin x_1, x_2 sẽ bị triệt tiêu (vanishing gradient). s_t chỉ chứa được những thông tin gần như x_t, x_{t-1}, x_{t-2} mà o_t phụ thuộc vào s_t mà s_t thì quá ít thông tin của x_1, x_2, x_3 dẫn đến trường hợp “long term dependencies”.

Cách khắc phục vấn đề “long term dependencies”

Khắc phục vấn đề “long term dependencies”: LSTM, GRU, ... giúp nhớ cái cần nhớ, có thể xóa bỏ các thông tin không cần thiết, lưu hoặc truyền những thông tin quan trọng bằng cách thay node RNN thành các cơ chế “cổng” (gate) để tính toán hidden states từ đó có thể kiểm soát được thông tin RNN.

Câu 8: Một số ứng dụng của RNN

1) *Dịch máy (machine translation)*: input là ngôn ngữ nguồn (hay ngôn ngữ cần dịch) ví dụ như English và output là ngôn ngữ đích (ngôn ngữ dịch) ví dụ như French. Để dịch được ta cần phải xem xét toàn bộ chuỗi đầu vào cần dịch mới có thể suy luận được. Ứng dụng nổi tiếng về dịch máy đó là Google Translate, với hơn 109 ngôn ngữ.

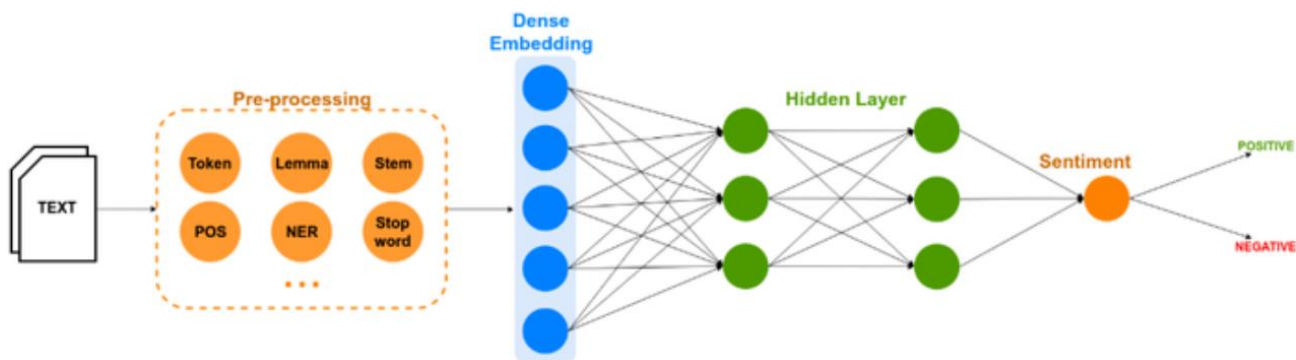


Hình 7: Sơ đồ encoder-decoder cho dịch máy

Để tìm hiểu code về dịch máy, (Việt-Anh), tham khảo tại đây.

[https://github.com/ndtuan10/DeepLearning_CS431.L21.KHCL/tree/main/BaiTap/BaiTap7 Neural Machine Translation%20Viet-Eng dataset](https://github.com/ndtuan10/DeepLearning_CS431.L21.KHCL/tree/main/BaiTap/BaiTap7%20Neural%20Machine%20Translation%20Viet-Eng%20dataset)

2) *Phân tích cảm xúc (sentiment analysis)*: nhằm xác định xem dữ liệu là tích cực, tiêu cực hay trung lập. Input là một đoạn văn bản, sau đó được chuyển đổi thành feature vector có thể có từ 300 đến 1000 chiều, đầu ra tương ứng là 1 tag của 1 trong 3 loại {positive, negative, neutral}. Bài toán thường được các công ty sử dụng để phát hiện cảm xúc trong dữ liệu xã hội, đánh giá danh tiếng của 1 thương hiệu và tìm hiểu mong muốn của khách hàng.



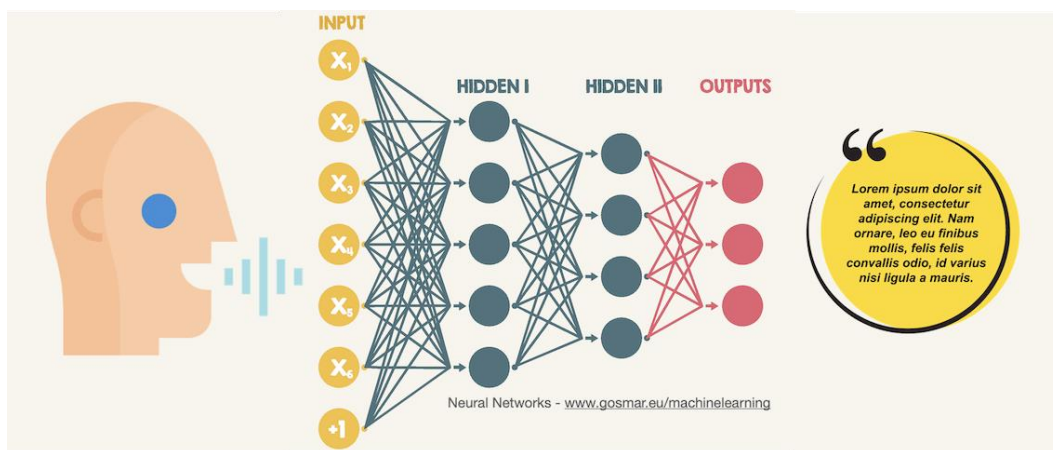
Hình 8: Sentiment analysis sử dụng Deep Learning

Để tìm hiểu code về bài toán phân tích cảm xúc, tham khảo tại đây.

[https://github.com/ndtuan10/DeepLearning_CS431.L21.KHCL/tree/main/BaiTap/BaiTap6 Sentiment-Analysis LSTM%20FoodyReview](https://github.com/ndtuan10/DeepLearning_CS431.L21.KHCL/tree/main/BaiTap/BaiTap6%20Sentiment-Analysis%20LSTM%20FoodyReview)

3) *Nhận dạng giọng nói (speech recognition)*: input là 1 đoạn audio, vì các tín hiệu được ghi âm thì lưu trữ dưới dạng số hóa, ta cần chuyển đổi chúng ở dạng số rồi rạc do đó âm

thanh này sẽ được lấy mẫu ở 1 tần số cụ thể để chuyển đổi sóng âm thành số, phương pháp này được gọi là sampling, sau đó trích xuất các đặc trưng và output của nó là text transcript. Có rất nhiều ứng dụng cho bài toán trên, ví dụ như đặt hàng bằng giọng nói, tìm kiếm thông tin phổ biến như Google Now của Android hay Hey Siri của IOS.

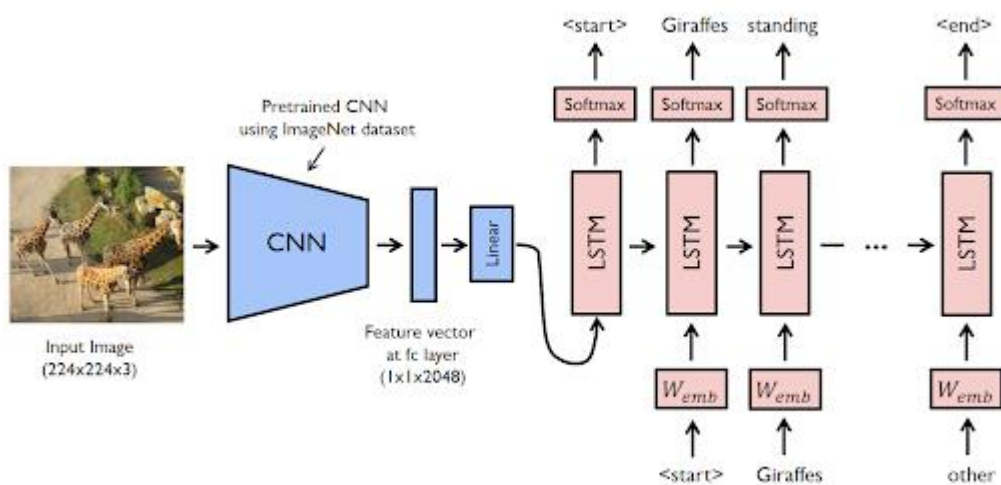


Hình 9: Speech Recognition sử dụng Deep Learning

Để tìm hiểu code về bài toán tổng hợp giọng nói, sử dụng pretrained model Tacotron2 và WaveGlow, tham khảo tại đây.

<https://github.com/ndtuan10/SpeechRecognition-Tacotron2>

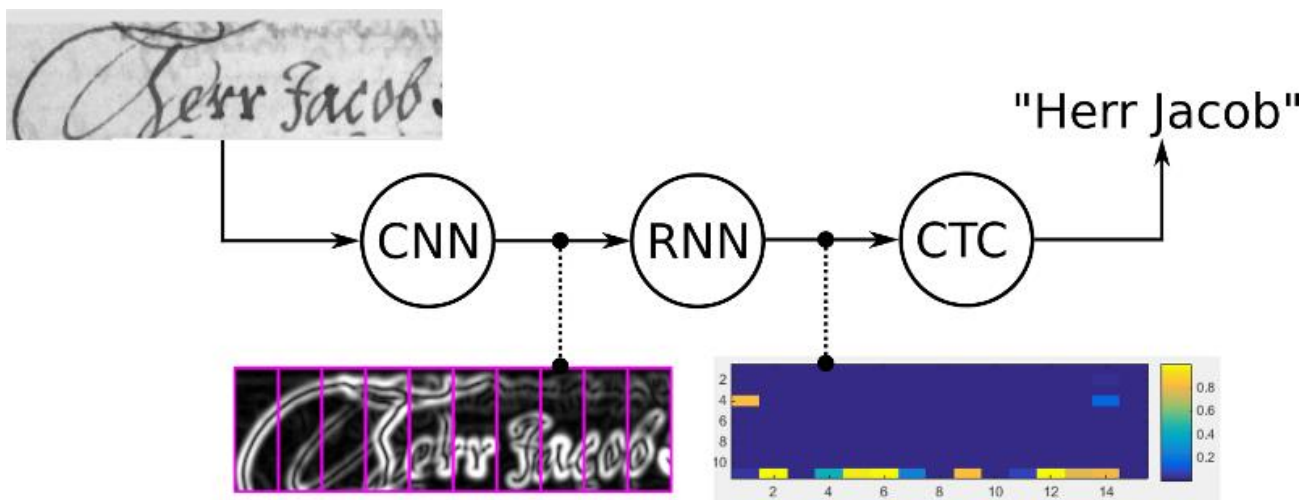
4) Mô tả ảnh (image captioning): input là ảnh tạo ra feature, từ feature cho ra output là text, cho biết mô tả của ảnh đó là gì. Để rút trích đặc trưng cho ảnh ta dùng CNN, sau đó feature này được đổi sang vector để mô tả, do đó, để ra được output là text ta cần mạng LSTM.



Hình 10: Image captioning sử dụng Deep Learning

5) Nhận diện chữ viết tay là tên riêng (Offline Handwriting Recognition): input là ảnh chứa dòng text là tên riêng, output là tên riêng dạng text. Ta sử dụng mạng CNN để rút trích các đặc trưng của ảnh, đưa ra output là các feature map. Ngoài ra còn có mạng RNN cụ thể là

bidirectional LSTM để dự đoán chuỗi đầu ra cho mỗi bước timestep. Thêm vào đó, 1 lớp transcription sử dụng hàm mất mát là CTC loss được dùng để dự đoán đầu ra cho mỗi bước timestep. Kiến trúc mạng kết hợp giữa CNN và RNN này được gọi là CRNN.

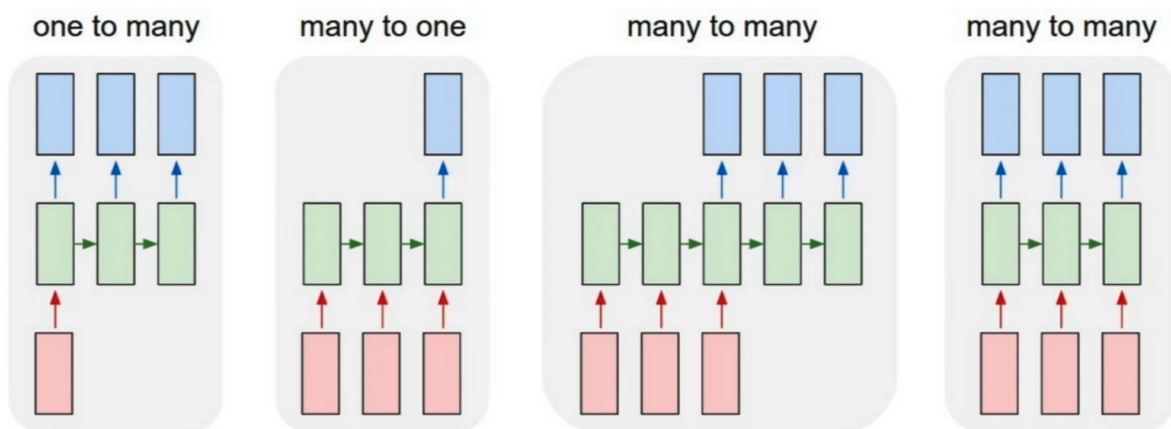


Hình 11: Kiến trúc mạng CRNN

Để tìm hiểu code về bài toán nhận diện chữ viết tay sử dụng CRNN, tham khảo tại đây.

<https://github.com/ndtuan10/MachineLearning-and-ComputerVision/tree/main/CV/DoAn>

Câu 9: Các biến thể của mạng RNN. Cho biết những bài toán nào phù hợp với các biến thể đó?



Hình 12: Một số các biến thể của mạng RNN

1) *one-to-many*:

Phù hợp cho bài toán mô tả ảnh (image captioning),...

2) *many-to-one*:

Phù hợp cho bài toán phân loại cảm xúc (sentiment analysis), bài toán phân loại từ (text classification),...

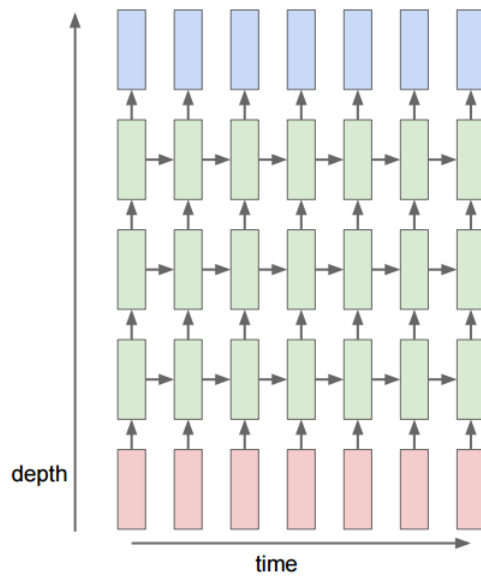
3) *many-to-many (loại 1)* (Hình thứ 4):

Phù hợp cho bài toán gán nhãn tự động (pos-tagging),...

4) *many-to-many (loại 2)* (Hình thứ 3):

Phù hợp cho bài toán dịch máy (machine translation), bài toán tóm tắt văn bản (text summarization),...

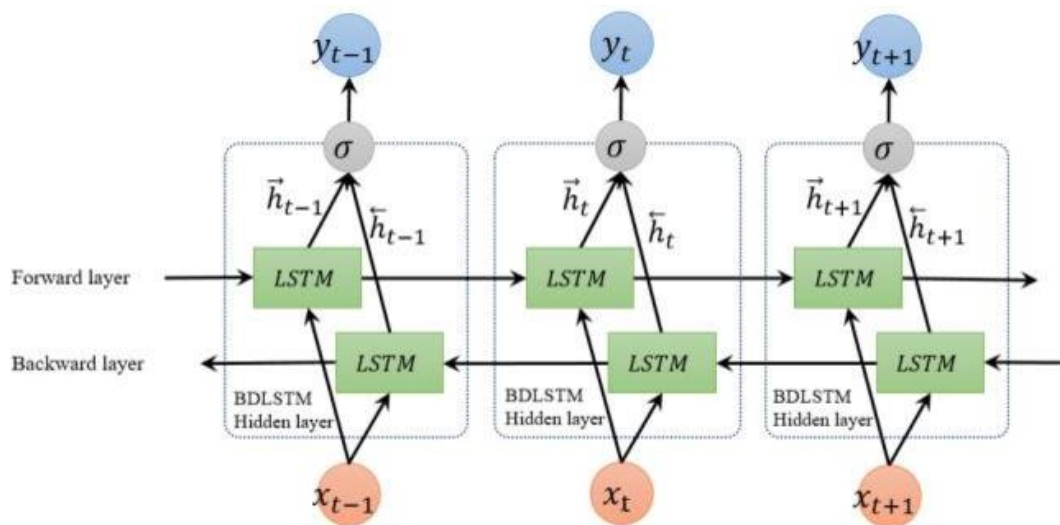
5) *multi-layers*:



Hình 13: Biểu thể *multi-layers RNN*

Phù hợp cho bài toán phân loại cảm xúc (sentiment analysis),...

6) *bidirectional RNN*:



Hình 14: Biểu thể *bidirectional RNN / LSTM*

Phù hợp cho bài toán gán nhãn tự động (pos-tagging),...

Câu 10: Cho biết cải tiến của mạng RNN?

a) LSTM (Long short term memory)

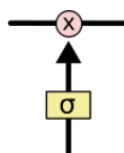
Mục đích chính là giúp mạng RNN có thể tận dụng được những thông tin “long term”, giúp nhớ những cái cần nhớ, có thể xóa bỏ các thông tin không cần thiết và truyền những thông tin quan trọng.

Cách tính hidden units sẽ phức tạp hơn.

Ý tưởng chính của LSTM:

- + Tạo thêm 1 bộ nhớ (memory) để nhớ được thông tin ở xa trước đó.
- + Cho phép thông tin được xuống bộ nhớ và hidden units mạnh hoặc yếu khác nhau tùy thuộc vào input tại thời điểm đó.

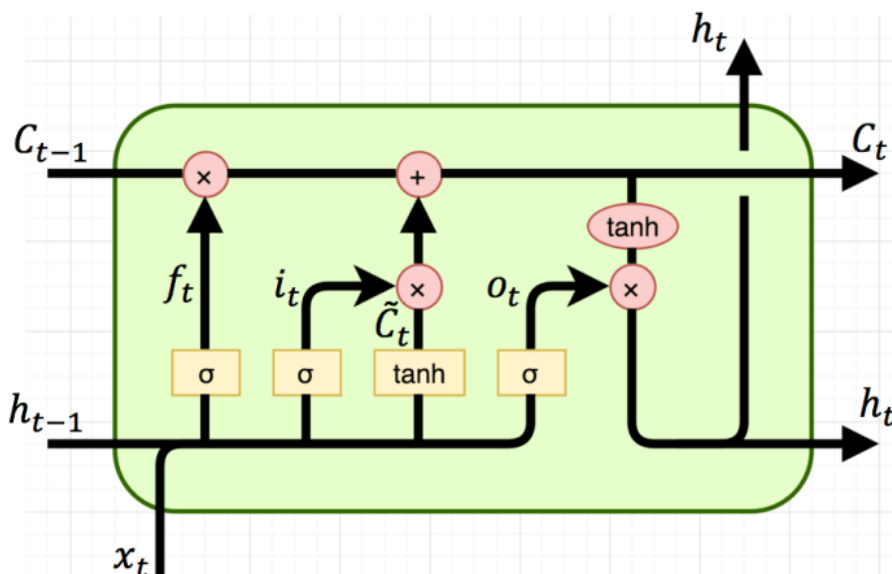
Có khả năng kiểm soát thông tin được thêm vào hoặc bỏ ra thông qua các cổng (gates).



Hình 15: Cơ chế “gate” trong mạng LSTM

Về kiến trúc mạng của LSTM

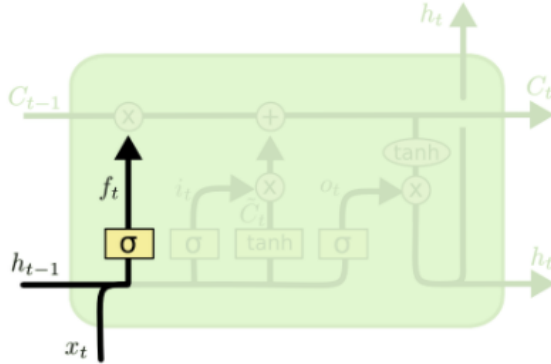
Nó khác với RNN ở chỗ nó có 2 luồng thông tin truyền vào 1 node trạng thái. Ngoài việc sử dụng hai dữ liệu input là current input (x_t) và hidden state (h_{t-1} , trong RNN là o_{t-1}) giống RNN, thì LSTM còn sử dụng thêm một input nữa là cell state (c_{t-1}). Để đưa được ra output c_t ta cần phải biết 3 thông tin đó là thông tin hiện tại x_t và 2 thông tin quá khứ gồm thông tin tổng hợp long term (c_{t-1}) và thông tin short term (h_{t-1}).



Hình 16: Kiến trúc mạng LSTM

Các bước của LSTM

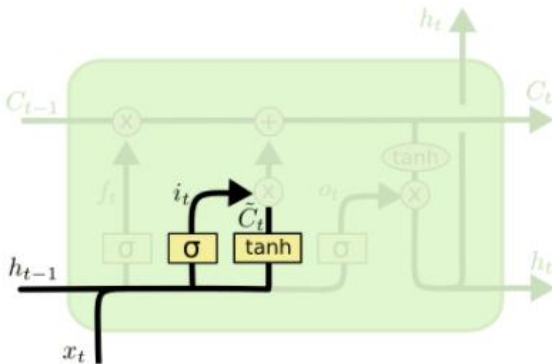
Step 1: “forget gate”: quyết định liệu bao nhiêu thông tin cũ (c_{t-1}) cần được lưu trữ, thông tin nào phải loại bỏ khỏi cell state. Thông tin từ current input và hidden state được chuyển qua sigmoid function với output là f_t nằm trong khoảng từ $[0,1]$. Nếu $f_t \rightarrow 0$ thì quên càng nhiều tức là phải loại bỏ thông tin, $f_t \rightarrow 1$ thì nhớ càng nhiều, tức là thông tin cần được giữ lại.



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

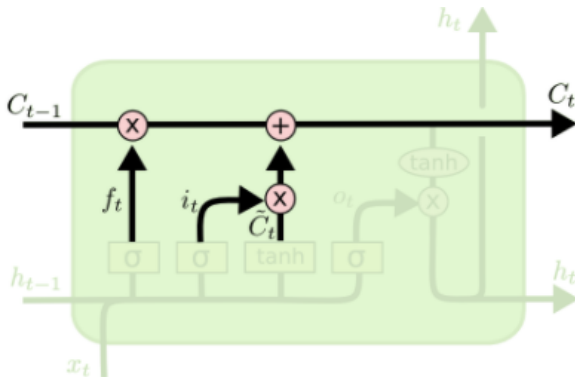
Hình 17: Forget gate

Step 2: “input gate”: có nhiệm vụ cập nhật thông tin mới vào sẽ được lưu vào cell state. Bước này gồm 2 phần tính input gate i_t và tính giá trị \tilde{C} mới sẽ được thêm. Ở đây ta nhân output của sigmoid với output của tanh để quyết định thông tin của current input và hidden state có nên được đưa vào cell state hay không. Sau đó cập nhật giá trị của cell (c) đưa vào giá trị cũ của c (c_{t-1}) vào thông tin mới sẽ thêm vào.



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

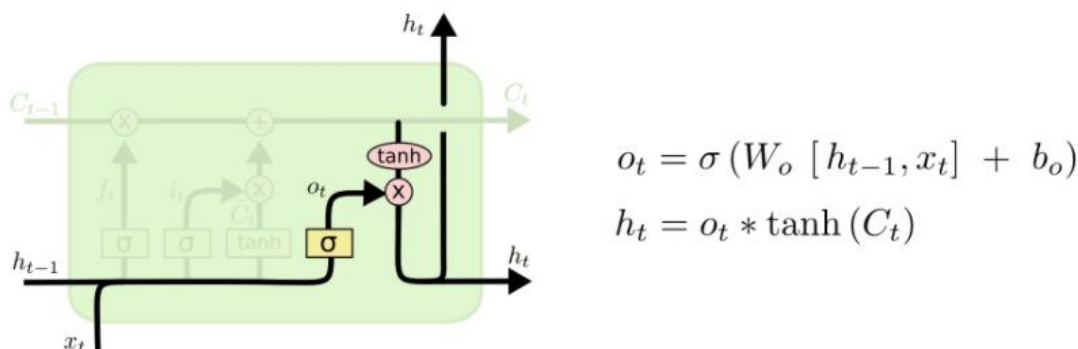
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$



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

Hình 18: Input gate

Step 3: “output gate”: có nhiệm vụ tính giá trị của hidden state cho t kế tiếp. Với việc sử dụng forget gate và input gate, ta có thể tính được giá trị mới của cell state và từ đó kết hợp với current input và hidden state để tính giá trị của hidden state tiếp theo. Ở đây giá trị của hidden state mới này cũng chính là giá trị prediction.

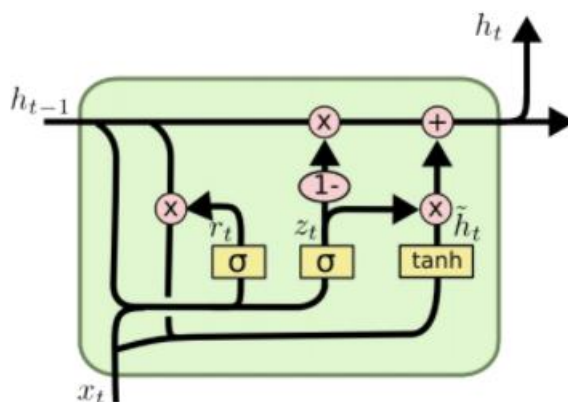


Hình 19: Output gate

Như vậy, trong mạng LSTM có 3 đầu vào là 3 cổng là current input (x_t), hidden state (h_{t-1}), và cell state (c_{t-1}); 2 đầu ra là cell state (c_t) và hidden state (h_t).

Vấn đề của LSTM : Mặc dù là LSTM giải quyết được vấn đề về “long term dependencies” nhưng cũng giống như các mạng liên quan đến RNN, nó vẫn bị vanishing gradient vì chuỗi biến đổi quá dài dẫn đến các đạo hàm khi tối ưu sẽ bị tiệm cận về 0.

b) GRU (Gated Recurrent Unit)



Hình 20: Kiến trúc mạng GRU

Update gate: $z_t = \sigma(W_z x_t + U_z h_{t-1})$

Reset gate: $r_t = \sigma(W_r x_t + U_r h_{t-1})$

Nội dung memory mới:

$$\tilde{h}_t = \tanh(W_h x_t + r_t \circ U_h h_{t-1})$$

Giá trị h được tính kết hợp bởi giá trị h cũ và nội dung mới. Nếu reset gate = 0, thì việc tính \tilde{h}_t không quan tâm giá trị memory cũ mà chỉ quan tâm thông tin của từ mới.

Giá trị h cuối cùng được cập nhật: $h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t$

Câu 11: Tổng hợp một số các câu hỏi về ứng dụng của mạng RNN.

Câu hỏi: Bài toán phân loại cảm xúc văn bản (sentiment analysis) có thể ứng dụng các biến thể nào của RNN?

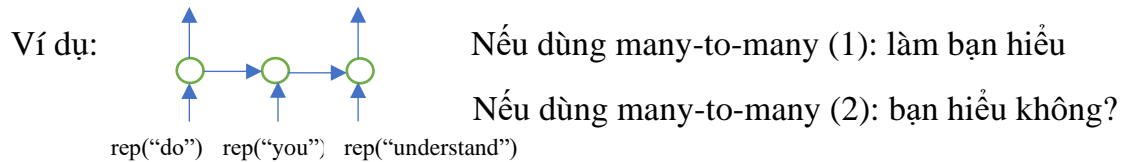
a) LSTM: vì câu dài, để nhớ hết thông tin mà LSTM có cơ chế forget gate để loại bỏ các thông tin cũ mà không cần thiết và nhớ những thông tin quan trọng.

b) many-to-one: vì bài toán có đầu vào là text và đầu ra chỉ trả ra 1 output trong {positive, negative, neutral}.

c) multi-layers: vì thêm layers sẽ rút trích được những đặc trưng hơn thì kết quả classify tốt hơn, ngoài ra output mạng concept trừu tượng nên cần multi-layers để học concept này.

Câu hỏi: Có thể dùng Bidirectional RNN cho bài toán phân loại cảm xúc văn bản (sentiment analysis) không? Không cần thiết vì many-to-one là ta đã biết những thông tin trước đó được lan truyền tới câu cuối (EOS) và đưa ra output, ta không cần phải nghe lại các thông tin trước đó.

Câu hỏi: Có thể dùng many-to-many dạng (1) cho bài toán dịch máy (machine translation) không? KHÔNG vì



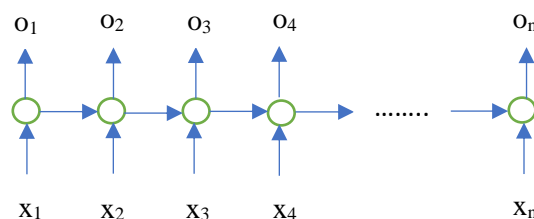
Để có thể dùng many-to-many (1) mà vẫn giúp dịch máy ta cần thêm bidirectional LSTM tuy nhiên chỉ dành khi tương ứng 1-1, 1 input sẽ cho ra 1 output.

Câu hỏi: Có thể dùng many-to-many dạng (1) cho bài toán tóm tắt văn bản (text summarization) không? KHÔNG (tương tự dịch máy).

Câu hỏi: Bài toán gán nhãn tự động (pos-tagging) có thể ứng dụng các biến thể nào của RNN?

a) Bidirectional: vì khi ta đưa vào 1 từ (x_t) thì tại vị trí này ta vẫn chưa đủ thông tin ngữ cảnh để kết luận, ta cần phải nghe hết câu của mình, sau đó lan truyền thông tin về ngược lại để tất cả các từ trước có đầy đủ thông tin.

b) tương tự là many-to-many (1):



Câu 12: Cho biết cơ chế Attention?

Cơ chế attention được áp dụng cho nhiều bài toán sử dụng mạng RNN như: dịch máy (machine translation), tóm tắt văn bản (text summarization), mô tả ảnh (image captioning) hay OCR (Optical Character Recognition),...

Ở đây mình sẽ nói về cơ chế attention trong bài toán dịch máy với mô hình *Seq2Seq*.

Vấn đề của bài toán dịch máy đó là

+ Nó không hoàn toàn tương ứng 1-1 về mặt câu/từ. Ví dụ: (từ “do” -> “có”), (từ “you” -> “bạn”), (từ “know” -> “hiểu”), (từ “what”, “mean” -> “ý”), (từ “I” -> “tôi”).

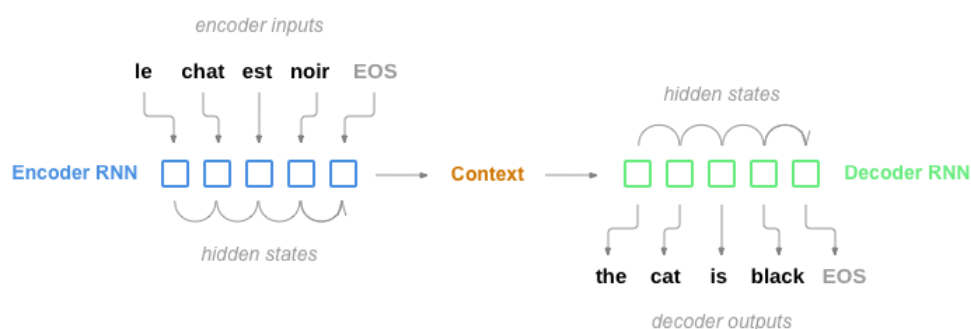
Do you know what I mean?
Bạn có hiểu ý tôi?

+ Khó khăn về văn hóa, quan điểm, ngôn ngữ của từng quốc gia.

+ Đồng thời tại thời điểm t nào đó, nó không biết chính xác phải tập trung nội dung dịch vào từ nào để dịch.

=> Từ đó áp dụng *cơ chế attention*.

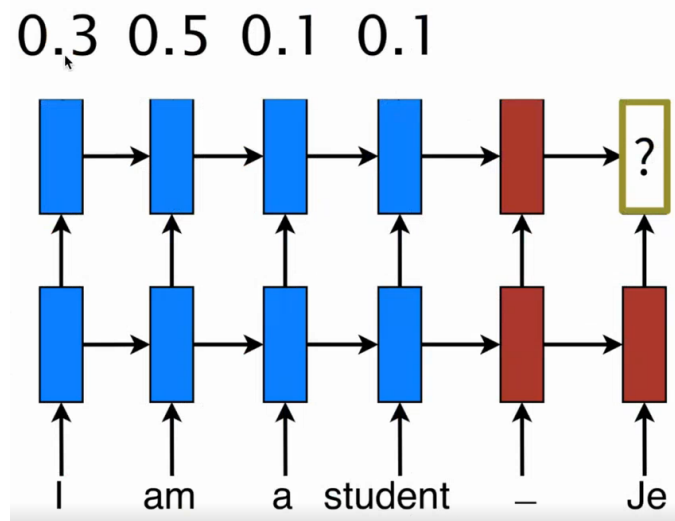
Trong mô hình seq2seq dùng cho bài toán dịch máy bao gồm 2 mạng RNN, 1 mạng RNN đóng vai trò làm Encoder với đầu vào là câu ở ngôn ngữ gốc, đầu ra tại layer cuối cùng của Encoder gọi là 1 context vector. Với ý nghĩa lượng thông tin từ câu của Encoder sẽ tóm gọn lại trong vector đầu ra cuối cùng. 1 mạng RNN khác đóng vai trò là Decoder, sử dụng chính context vector đó, cùng với hidden state và từ trước đó để predict từ tiếp theo qua từng timestep.



Hình 21: Encoder-Decoder trong mô hình seq2seq trong bài toán dịch máy

Ý tưởng của cơ chế này là thay vì mong phần encoder nén toàn bộ thông tin, ta cho phần decoder được quan sát toàn bộ output của encoder. Nó là một hàm tính trọng số của từ nào mà mình đang quan tâm, sử dụng alignment score (h_{t-1} , \bar{h}_s) để đánh trọng số cao nhất trong đó h_{t-1} là thông tin hidden state trước đó. Ví dụ: Khi ta dịch câu “I am a student” trong

tiếng Anh sang tiếng Pháp, ta được câu su “*Je suis étudiant*”. Đây rõ ràng là không có sự tương ứng 1-1 trong cách dịch, vì trong tiếng Pháp không có mạo từ, do đó (từ “I” -> “Je”), (từ “am” -> “suis”), (từ “a”, “student” -> “étudiant”). Khi ta đánh trọng số từ sẽ xuất hiện cao nhất sau từ “Je”, nhìn hình bên dưới ta thấy, từ có khả năng dịch kế tiếp đó là từ “am” (alignment score = 0.5), do vậy từ sẽ xuất hiện sau từ “Je” khi dịch xong là từ “suis”.



Hình 22: Ví dụ cách đánh trọng số alignment score

Về cách tính alignment score, có nhiều cách tính khác nhau, mình sẽ đề một vài cách ở đây:

- Content-base Attention: [Neural Turing Machine](#)

$$\text{score}(h_{t-1}, \bar{h}_s) = \text{cosine}[h_{t-1}, \bar{h}_s]$$

- Additive Attention, [Bahdanau's Paper: NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE](#)

$$\text{score}(h_{t-1}, \bar{h}_s) = v_a^T \tanh(W_1 h_{t-1} + W_2 \bar{h}_s)$$

- Multiplicative Attention hay General Attention, [Luong's Paper: Effective Approaches to Attention-based Neural Machine Translation](#)

$$\text{score}(h_{t-1}, \bar{h}_s) = (h_{t-1})^T W_a \bar{h}_s$$

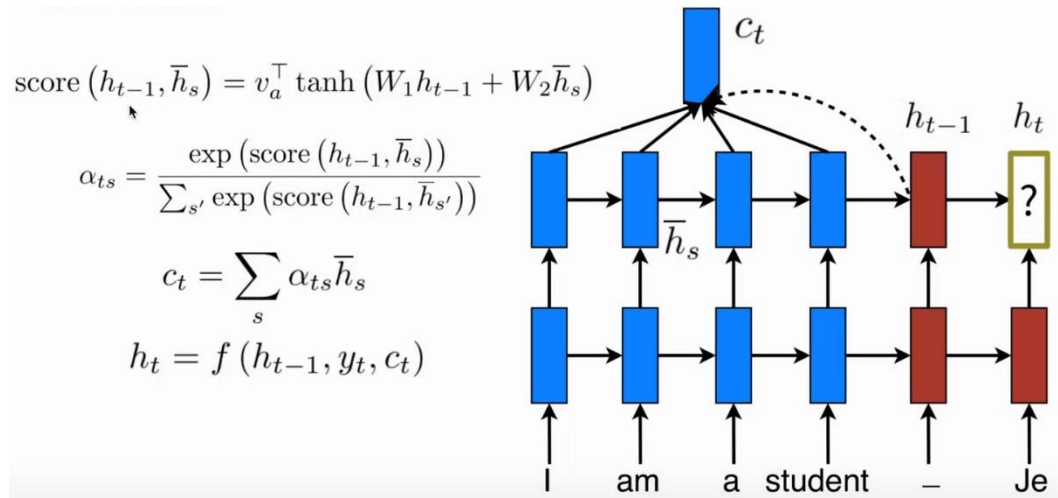
- Dot Product (simple mechanism), [Luong's Paper: Effective Approaches to Attention-based Neural Machine Translation](#)

$$\text{score}(h_{t-1}, \bar{h}_s) = (h_{t-1})^T \bar{h}_s$$

Ví dụ, ta sử dụng $\text{score}(h_{t-1}, \bar{h}_s) = v_a^T \tanh(W_1 h_{t-1} + W_2 \bar{h}_s)$ thì ở đây v_a , W_1 , W_2 là các tham số được học.

Context vector c_i được tính dưới dạng tổng trọng số của h_j $c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$

Trọng số α_{ij} của mỗi h_i được tính bằng $\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$, với $e_{ij} = \alpha(s_{i-1}, h_j)$



Hình 23: Ví dụ về tính alignment score

Thank you and have you enjoy it.