TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN ĐHQG-HCM

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO NGHIÊN CỨU VỀ DEEP NEURAL NETWORK

Môn: Nhập môn học máy

Giảng viên:

ThS Nguyễn Ngọc Đức

Sinh viên thực hiện:

Nguyễn Tấn Phát	20127588
Lê Ngọc Tường	20127383
Đặng Minh Đức	20127136

Contents

GIỚI THIỆU

1. Thông tin nhóm

STT	MSSV	Họ và tên	Đóng góp
1	20127588	**Nguyễn Tấn Phát**	40%
2	20127383	Lê Ngọc Tường	33%
3	20127136	Đặng Minh Đức	27%

2. Nội dung nghiên cứu

- **Đề tài: AlexNet CNN - Deep Neural Network for Image Classification**
- Neural Network
- Activation Function
- Backpropagation
- Các khái niệm về xử lí ảnh
- Convolutional Neural Network (CNN)
- AlexNet CNN

3. Lý do chọn đề tài

Trong những năm gần đây, trí truệ nhân tạo – AI đã và đang được phát triển trên đỉnh cao. Sẽ không có gì bất ngờ nếu có một chiếc điện thoại có khả năng nhận diện được khuôn mặt của chúng ta; hay một ứng dụng mua – bán (shopee, lazada,…) luôn biết được sở thích mình và đưa ra lời gợi ý; và một con robot có thể trò chuyện như một con người thực thụ thì cũng không có gì là lạ. Vì được gọi là "trí tuệ nhân tạo" nên con người chúng ta luôn muốn rằng "những gì con người có thể làm thì AI cũng có thể làm, thậm chí làm làm tốt hơn". Từ đó, từng chút một, các vấn đề của con người dần được AI giải quyết.

Unstructured data là kiểu dữ liệu phổ biến nhất, 80-90% dữ liệu được tạo ra, thu thập bởi các tổ chức nằm ở dạng này và trong đó có ảnh. Việc rút ra từ khóa của một câu văn dễ hơn rất nhiều so với việc nêu ra một dung của một bức ảnh. Ảnh cũng là một dạng phương tiện trực quan và phổ biến vì vậy nhu cầu phân tích, phân loại ảnh (phân loại các hình ảnh theo ngày, sự kiện mà không phải dùng thao tác thủ công mất thời gian) là cần thiết. Trong khi các mô hình classical machine learning chưa có đủ tính thực nghiệm cho các vấn đề thực, còn Deep Neural Network (dựa trên ANN và có thêm nhiều lớp ẩn) đã tạo ra sự đột phá mới giải quyết vấn đề này.

Với sự phát triển của truyền thông xã hội, hình ảnh đã trở nên phổ biến với người tiêu dùng nói riêng cũng như là các trang truyền thông khác nói chung. Tuy nhiên, hình ảnh có thể bị xuống cấp do chất lượng của ảnh giảm đi một cách đáng kể. Cụ thể, việc tạo nội dung ảnh cũng như phân tích, xử lý ảnh của máy đã làm giảm đi chất lượng của ảnh dẫn đến chúng ta không thể biết được ảnh này trông như thế nào hay ảnh đó nói về cái gì. Vì thế, chúng ta dùng mạng nơ-ron sâu (Deep Neural Network - DNN) để phân tích ảnh chi tiết hơn cũng như cho ra chất lượng của ảnh tăng lên dẫn đến độ chính xác của output càng cao, tóm gọn là để khắc phục việc hình ảnh bị xuống cấp.

DNN cũng giúp mô phỏng và nhận diện hình ảnh. Chắc hẳn, chúng ta đều đã từng thấy máy tính tự động nhận diện các hình ảnh. Ví dụ: Facebook có thể tự động gắn thẻ chính bản thân và bạn bè. Tương tự, Google Photos có thể tự động gắn nhãn ảnh của bạn để tìm kiếm dễ dàng hơn, Google Search có thể tự tìm kiếm dựa vào hình ảnh, Google Translate có thể dịch văn bản dưới dạng hình ảnh.

Tesla Inc. – công ty sản xuất xe điện của tỷ phú giàu nhất thế giới cũng nghiên cứu về điện xe tự hành. Vấn đề trong việc phát triển này chính là các nhà phân tích phải xây dựng nên các kịch bản có thể xảy ra trong cuộc sống và lập trình việc xử lý các tình huống tích hợp trong chiếc xe hơi. Bên cạnh đó chu kỳ kiểm tra và triển khai thường xuyên các thuật toán deep learning để đảm bảo sự an toàn xảy ra với nhiều tình huống và hàng ngàn kịch bản khác nhau trong đời sống. Và ảnh là một trong những dữ liệu cần thiết để có thể xác định được các phương hướng, các biển báo, các tuyến đường phù hợp.

4. Nội dung công việc

Nội dung	Chi tiết	Thực hiện
Tìm hiểu	Neural Network	Tất cả thành viên
	Nhiệm vụ activation function	Tất cả thành viên
	Gradient Descent	Tất cả thành viên
	Backpropagation	Tất cả thành viên
	Các khái niệm về xử lí ảnh	Tất cả thành viên
	Convolutional Neural Network	Tất cả thành viên
	AlexNet	Tất cả thành viên

Nội dung Báo cáo	1. Neural Network	Ngọc Tường
	2. Nhiệm vụ activation function	Ngọc Tường
	3. Backpropagation	Tấn Phát
	4. Các khái niệm về xử lí ảnh	Minh Đức
	5. Convolutional Neural Network	Minh Đức
	6.1 AlexNet: Giới thiệu	Minh Đức
	6.2 AlexNet: DataSet	Minh Đức
	6.3 AlexNet: Kiến trúc	Ngọc Tường
	6.4 AlexNet: Overfitting	Ngọc Tường
	6.5 AlexNet: Quá trình học của mạng	Tấn Phát
	KẾT LUẬN: ƯU ĐIỂM	Minh Đức
	KẾT LUẬN: NHƯỢC ĐIỂM	Ngọc Tường
	KẾT LUẬN: KẾT QUẢ	Tấn Phát
Kiểm duyệt, thống nhất, điều chỉnh		Tấn Phát

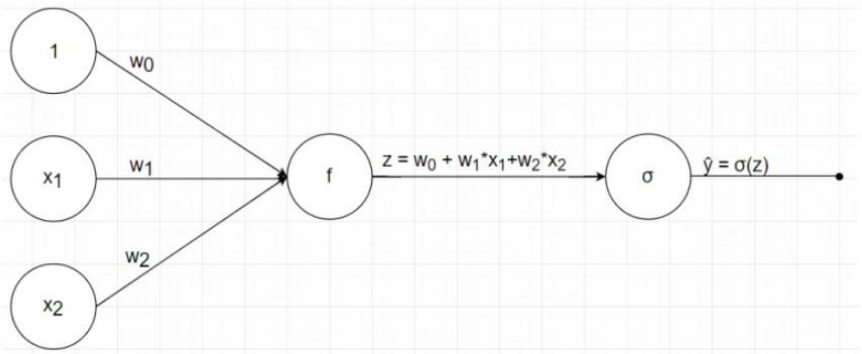
NỘI DUNG

1. Neural Network

1.1 Nhắc lại về Logistics Regression

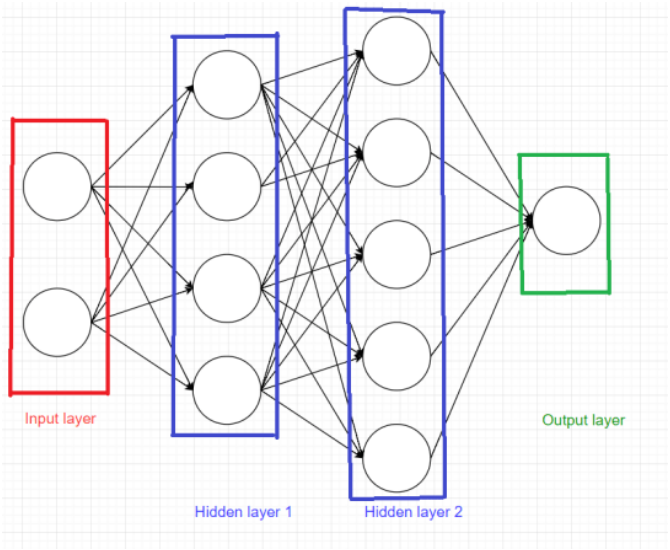
Logistic regression là mô hình neural network đơn giản nhất chỉ với input layer và output layer. Mô hình của logistic regression có 2 bước:

- Tính tổng linear: $z = w_0 + x_1 w_1 + x_2 w_2$
- Áp dụng sigmoid function: $\hat{y} = \sigma(z)$



Logistics Regression

1.2 Sơ lược về Neural Network



(Mô hình neural network đơn giản với 3 lớp)

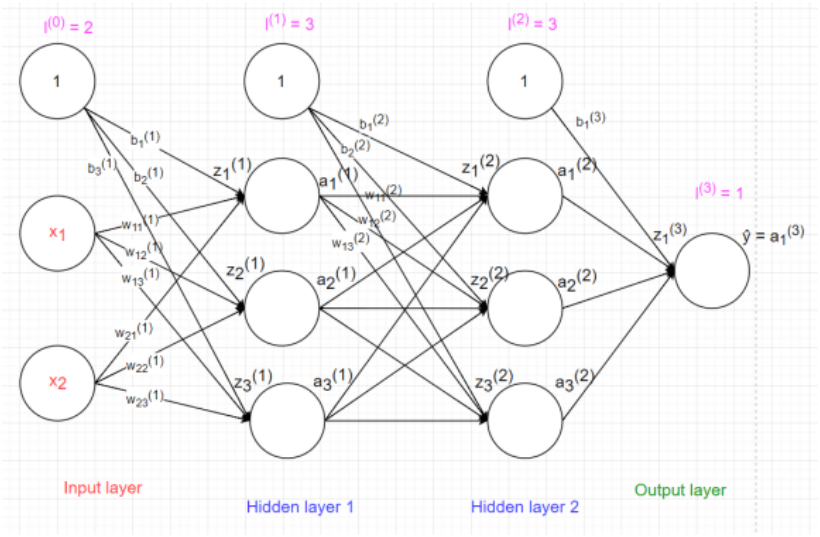
Layer đầu tiên là input layer, các layer ở giữa được gọi là hidden layer, layer cuối cùng được gọi là output layer. Các hình tròn được gọi là node.

Mỗi mô hình luôn có 1 input layer, 1 output layer, có thể có hoặc không các hidden layer. Tổng số layer trong mô hình được quy ước là số layer – 1 (Không tính input layer).

Mỗi node trong hidden layer và output layer :

- Liên kết với tất cả các node ở layer trước đó với các hệ số w riêng.
- Mỗi node có 1 hệ số bias b riêng.
- Diễn ra 2 bước (tương tự logistics regression): tính tổng linear và áp dụng activation function

1.3 Các kí hiệu



(Mô hình neural network đơn giản với 3 lớp)

Tên gọi	Kí hiệu
Số node trong hidden layer thứ i:	$\boldsymbol{l^{(i)}}$
Ma trận hệ số giữa layer (k-1) và layer k: $(\boldsymbol{l^{(k-1)} * l^{(k)}})$	$\boldsymbol{w^{(k)}}$
Hệ số kết nối từ node thứ i của layer k-1 đến node thứ j của layer k	$\boldsymbol{w_{ij}^{(k)}}$
Hệ số bias của các node trong layer k ($\boldsymbol{l^{(k)} * 1}$)	$\boldsymbol{b^{(k)}}$
Bias của node thứ i trong layer k	$\boldsymbol{b_{ij}^{(k)}}$

Phương pháp tính node i trong layer l: $\boldsymbol{b_i^{(l)}}$

- Tổng tất cả các node trong layer trước nhân với hệ số w tương ứng, rồi cộng với bias b: $\boldsymbol{z_i^{(l)} = \sum_{j=1}^{l^{(l-1)}} a_j^{(l-1)} w_{ji}^l + b_i^{(l)}}$
- Activation function: $\boldsymbol{a_i^{(l)} = \sigma(z_i^{(l)})}$

Giá trị các node trong layer k sau bước tính tổng: $\boldsymbol{z^{(k)}}$ ($\boldsymbol{shape}$: $\boldsymbol{l^{(k)}\ x\ 1}$)

Giá trị của các node trong layer k sau activation function: $\boldsymbol{a^{(k)}}$ ($\boldsymbol{shape}$: $\boldsymbol{l^{(k)}\ x\ 1}$)

1.4 Feedforward

VD: Input layer $\mathbf{2 * 1}$ – tức là có 2 thuộc tính. Ta thêm 1 bias thì sẽ được ma trận kích thước $\mathbf{3 * 1}$ là $[\boldsymbol{a_1}^{(0)} \quad \boldsymbol{a_2}^{(0)} \quad \boldsymbol{a_3}^{(0)}]^{\boldsymbol{T}}$. Giả sử lớp ẩn đầu tiên có 3 thuộc tính thì ta sẽ có cách tính như sau:

$$z^{(1)} = \begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \end{bmatrix} = \begin{bmatrix} a_1^{(0)} * w_{11}^{(1)} + a_2^{(0)} * w_{21}^{(1)} + a_3^{(0)} * w_{31}^{(1)} + b_1^{(1)} \\ a_1^{(0)} * w_{12}^{(1)} + a_2^{(0)} * w_{22}^{(1)} + a_3^{(0)} * w_{32}^{(1)} + b_2^{(1)} \\ a_1^{(0)} * w_{13}^{(1)} + a_2^{(0)} * w_{23}^{(1)} + a_3^{(0)} * w_{33}^{(1)} + b_3^{(1)} \end{bmatrix}$$

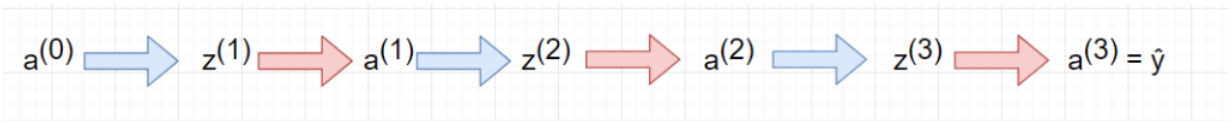
$$= (W^{(1)})^T * a^{(0)} + b^{(1)}$$

$$a^{(1)} = \sigma(z^{(1)})$$

(Tính các giá trị của lớp ẩn đầu tiên)

Ta có cách tính tương tự cho $z^{(2)}$, $z^{(3)}$, $a^{(2)}$, và $\hat{y} = a^{(3)}$.

Từ đó ta có cách biểu diễn đơn giản hơn.



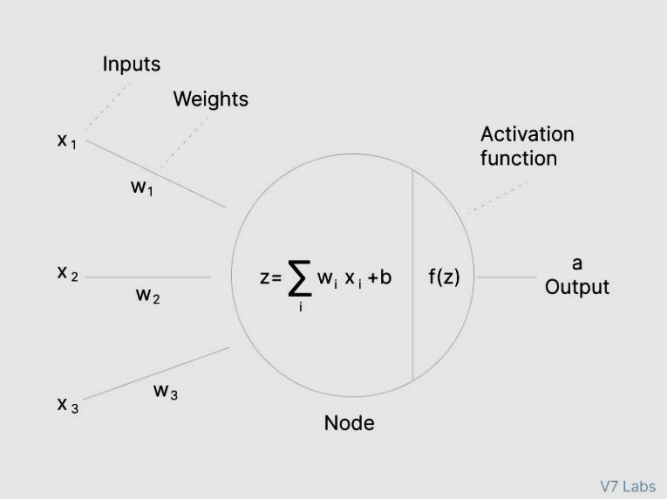
(Mô hình ở dạng ngắn gọn)

Giờ từ input X ta có thể tính được giá trị dự đoán $\hat{\boldsymbol{y}}$, tuy nhiên việc chính cần làm là đi tìm hệ số W và b, vì vậy ta cần gradient descent và tìm đạo hàm của các hệ số đối với loss function. Và việc tính đạo hàm của các hệ số trong neural network được thực hiện bởi thuật toán **backpropagation**, sẽ được trình bày ở phần 3.

2. Activation Function

2.1 Khái niệm hàm phi tuyến

Activation functions là những hàm kích hoạt được áp dụng vào đầu ra của các nơ-ron trong tầng ẩn của một mô hình mạng, và được sử dụng làm input data cho tầng tiếp theo.



Minh họa: f(z) là activation function

Hàm kích hoạt mô phỏng tỷ lệ truyền xung qua axon của một neuron thần kinh. Trong một mạng nơ-ron nhân tạo, hàm kích hoạt đóng vai trò là thành phần phi tuyến tại output của các nơ-ron.

2.2 Tại sao lại cần các hàm kích hoạt phi tuyến?

Nếu không có hàm kích hoạt phi tuyến thì mạng neural dù có nhiều lớp đến đâu vẫn chỉ có hiệu quả như lớp tuyến tính.

VD: Ta sẽ xem xét một mạng nơ-ron nhỏ gồm 2 lớp (có các hàm kích hoạt tuyến tính tương ứng). Mạng này sẽ nhận vào input là X và trả ra output Y:

Ở lớp thứ nhất: ta có tổng z_1 và output a_1

- $z_1 = w_1^T x + b_1$
- $a_1 = \sigma_1(z_1) = cz_1 + d$

Để đơn giản, ta sẽ xem c $= 1$ và d $= 0 \rightarrow a_1 = z_1$

Ở lớp thứ hai: ta có tổng z_2 và output a_2:

- $z_2 = w_2^T z_1 + b_2 = w_2^T(w_1^T x + b_1) + b_2 = w'x + b'$
- $a_2 = \sigma_2(z_2) = cz_2 + d$

Qua hai lớp thì output a2 vẫn chỉ là một hàm tuyến tính của x ban đầu, vì vậy dù có nhiều lớp thì vẫn chỉ có tác dụng như một lớp tuyến tính.

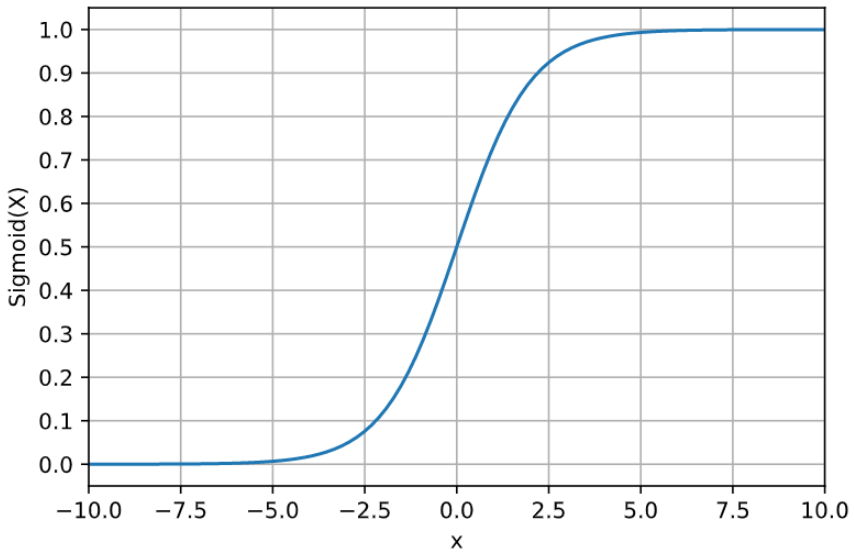
Hãy tưởng tượng rằng thay vì áp dụng 1 hàm phi tuyến, ta chỉ áp dụng 1 hàm tuyến tính vào đầu ra của mỗi neuron. Vì phép biến đổi không có tính chất phi tuyến, việc này không khác gì chúng ta thêm một tầng ẩn nữa vì phép biến đổi cũng chỉ đơn thuần là nhân đầu ra với các weights. Với chỉ những phép tính đơn thuần như vậy, trên thực tế mạng neural sẽ không thể phát hiện ra những quan hệ phức tạp của dữ liệu (ví dụ như: dự đoán chứng khoán, các bài toán xử lý ảnh hay các bài toán phát hiện ngữ nghĩa của các câu trong văn bản).

Nói cách khác nếu không có các activation functions, khả năng dự đoán của mạng neural sẽ bị giới hạn và giảm đi rất nhiều, sự kết hợp của các activation functions giữa các tầng ẩn là để giúp mô hình học được các quan hệ phi tuyến phức tạp tiềm ẩn trong dữ liệu.

Bây giờ ta sẽ xem qua một số hàm kích hoạt phi tuyến phổ biến:

2.3 Sigmoid

$$\boldsymbol{\sigma(x) = \frac{1}{1 + e^{-x}} \rightarrow \sigma'(x) = \sigma(x) * (1 - \sigma(x))}$$



(Đồ thị hàm Sigmoid)

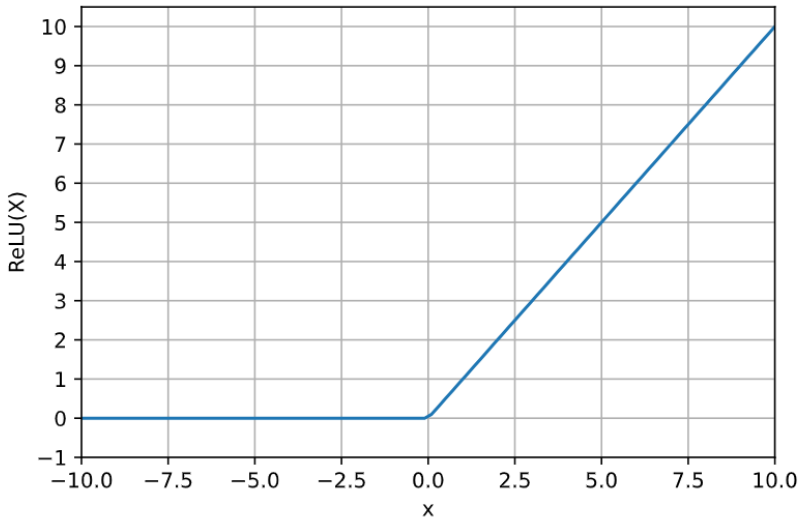
Hàm Sigmoid nhận đầu vào là một số thực và chuyển thành một giá trị trong khoảng (0;1)

Đầu vào là số thực âm rất nhỏ sẽ cho đầu ra tiệm cận với 0, ngược lại, nếu đầu vào là một số thực dương lớn sẽ cho đầu ra là một số tiệm cận với 1. Trong quá khứ Hàm Sigmoid hay được dùng vì có đạo hàm rất đẹp. Tuy nhiên hiện nay hàm Sigmoid rất ít được dùng vì những nhược điểm sau:

- *Hàm Sigmoid bão hào và triệt tiêu gradient:* Một nhược điểm dễ nhận thấy là khi đầu vào có trị tuyệt đối lớn (rất âm hoặc rất dương), gradient của hàm số này sẽ rất gần với 0. Điều này đồng nghĩa với việc các hệ số tương ứng với unit đang xét sẽ gần như không được cập nhật (còn được gọi là vanishing gradient).
- *Hàm Sigmoid không có trung tâm là 0 gây khó khăn cho việc hội tụ:* chúng ta có thể giải quyết vấn đề này bằng cách chuẩn hoá dữ liệu về dạng có trung tâm là 0 (zero-centered) với các thuật toán batch/layer normalization.

2.3 ReLU

$$\boldsymbol{f(x) = max(0, x)} \rightarrow \boldsymbol{f'(x)} = \begin{cases} \boldsymbol{0\ if\ x < 0} \\ \boldsymbol{1\ if\ x > 0} \end{cases}$$



(Đồ thị của ReLU)

Hàm ReLU đang được sử dụng khá nhiều trong những năm gần đây khi huấn luyện các mạng neuron. ReLU đơn giản lọc các giá trị < 0. Nhìn vào công thức chúng ta dễ dàng hiểu được cách hoạt động của nó. Một số ưu điểm khá vượt trội của nó so với Sigmoid và Tanh:

Những ưu điểm có thể kể đến của ReLU như:

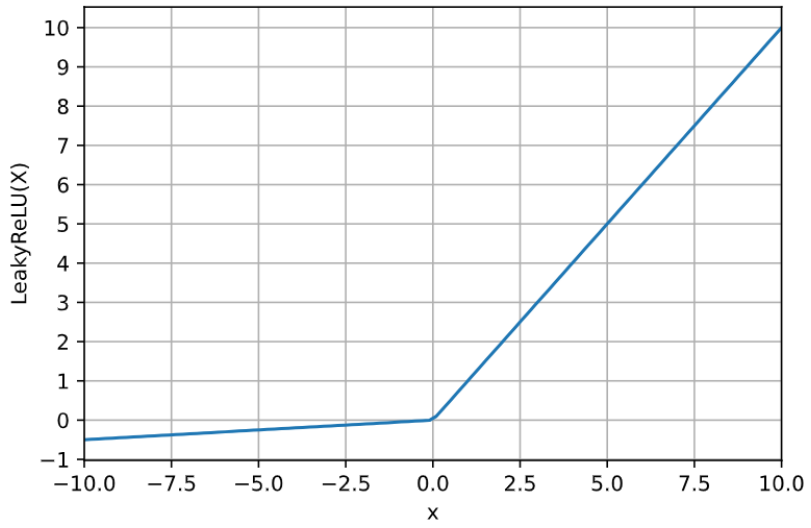
- Tốc độ hội tụ nhanh hơn hẳn. ReLU có tốc độ hội tụ nhanh gấp nhiều lần Tanh. Điều này có thể do ReLU không bị bão hoà ở 2 đầu như Sigmoid và Tanh.
- Tính toán nhanh hơn. Tanh và Sigmoid sử dụng hàm exp và công thức phức tạp hơn ReLU rất nhiều do vậy sẽ tốn nhiều chi phí hơn để tính toán.

Tuy vậy vẫn có một số nhược điểm đối với ReLU:

- Với các node có giá trị nhỏ hơn 0, qua ReLU activation sẽ thành 0, hiện tượng đấy gọi là "Dying ReLU". Nếu các node bị chuyển thành 0 thì sẽ không có ý nghĩa với bước linear activation ở lớp tiếp theo và các hệ số tương ứng từ node đấy cũng không được cập nhật với gradient descent. Để khắc phục nhược điểm này Leaky ReLU đã ra đời.
- Khi learning rate lớn, các trọng số (weights) có thể thay đổi theo cách làm tất cả neuron dừng việc cập nhật.

2.5 Leaky ReLU

$$f(x) = \alpha x \; if \; (x < 0) \; else \; x \rightarrow f'(x) = \begin{cases} \alpha \; if \; x < 0 \\ 1 \; if \; x > 0 \end{cases}$$



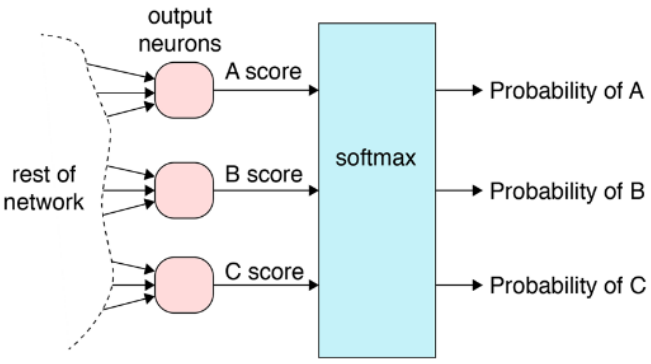
(Đồ thị của Leaky ReLU)

Leaky ReLU loại bỏ "Dying ReLU". Thay vì trả về giá trị 0 với các đầu vào <0 thì Leaky ReLU tạo ra một đường xiên có độ dốc nhỏ.

Ngoài Leaky ReLU có một biến thể cũng khá nổi tiếng của ReLU là PReLU. PReLU tương tự Leaky ReLU nhưng cho phép neuron tự động chọn hệ số α tốt nhất.

2.6 Softmax (Hàm trung bình mũ)

$$f_i(x) = f_i(x_1, \dots x_k) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}$$



(Minh họa Softmax)

Là hàm kích hoạt ở lớp ra của một mạng neural, luôn trả về giá trị thuộc nửa đoạn (0;1].

Hàm này lấy tất cả các output của mạng và chuẩn hóa chúng về xác xuất.

Những lợi ích của Softmax:

- Hàm softmax là tối ưu khi tính toán xác suất tối đa trong tham số mô hình.
- Tính chất của hàm softmax khiến hàm phù hợp với sự thông dịch xác suất, rất hữu ích trong Machine Learning.
- Chuẩn hóa softmax là một cách để giảm thiểu ảnh hưởng của những giá trị cực trị hay dữ liệu ngoại lai trong dữ liệu mà không phải chỉnh sửa dữ liệu ban đầu.

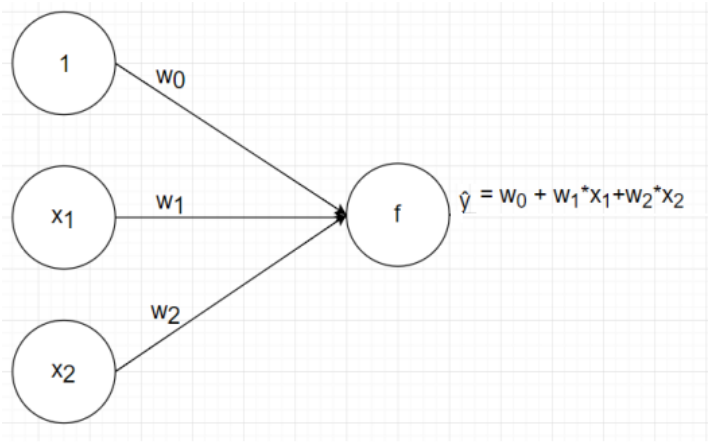
Kết luận: Tùy bài toán ta sẽ chọn hàm kích hoạt phi tuyến thích hợp. Khi tìm hiểu về các cấu trúc mạng cụ thể, các activation khác nhau sẽ được sử dụng, tuỳ vào độ sâu của mạng, output mong muốn, thậm chí là dữ liệu của bài toán. Chúng ta không thể nói hàm nào tốt hơn khi chưa xét đên điều kiện cụ thể.

3. Backpropagation

3.1 Gradient Descent với Linear Regression

Gradient Descent (GD) sinh ra nhằm mục đích tính xấp xỉ trọng số W sao cho giá trị của hàm lỗi – Loss Function đạt giá trị thấp nhất có thể. Ở đây, ta chỉ cố gắng tìm W_0 thõa mãn $Loss(W_0)' \approx 0$.

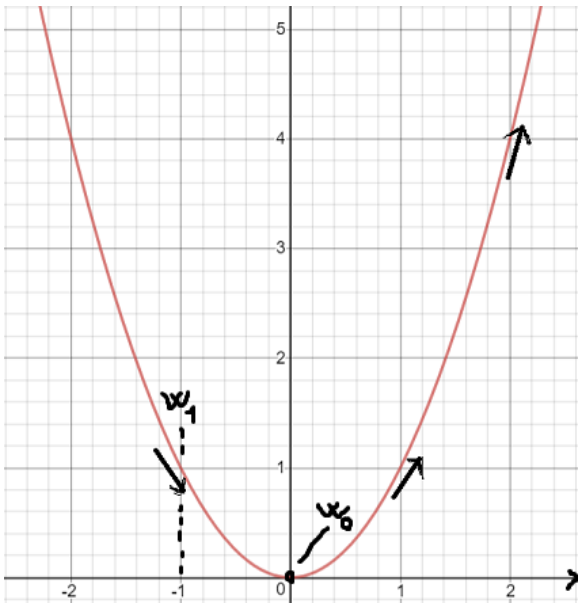
Giả sử ta có mô hình Linear Regresstion sau:



Ta cần đi tìm w_0,w_1,w_2 hay ma trận [W]

Ta cần đi tìm w_0, w_1, w_2 hay ma trận $[W]$

Hàm lỗi: $\boldsymbol{J = \frac{1}{2N} * (\sum_{i=1}^{N} (\hat{y}_i - y_i)^2)}$ là một hàm bậc hai, có dạng parabol

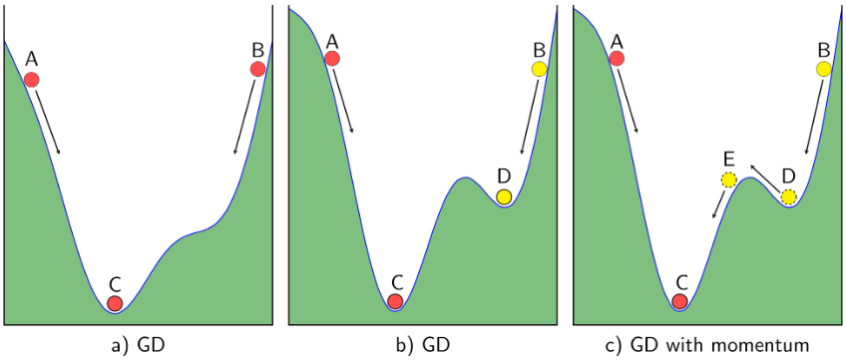


w_1 = -1 (là giá trị random đầu tiên) w_0 = 0 là giá trị cần tìm

Có thể thấy, tại $\boldsymbol{w_1 = -1}$: $\boldsymbol{J(w_1)' < 0}$ hay giá trị Loss Function có xu hướng giảm. Do đó, ta phải tăng $\boldsymbol{w_1}$ lên $\boldsymbol{w_2 = w_1 + \varepsilon}$ để được $\boldsymbol{J(w_2) < J(w_1)}$. Tương tự, nếu Loss Function có xu hướng tăng thì ta phải giảm giá trị của $\boldsymbol{w_1}$ để được hàm lỗi nhỏ hơn. Đây là lí do người ta gọi phương pháp này là Gradient Descent – đi ngược đạo hàm.

Tuy nhiên, thuật toán này chỉ dừng lại ở việc tìm điểm cực tiểu chứ chưa thực sự là điểm có hàm lỗi đạt giá trị nhỏ nhất(min), ta cần cải thiện Gradient Descent thông qua thuật toán GD với Momentum – Quán tính.

VD: Vị trí rơi của hòn bi – ta coi C, D là 2 điểm cực tiểu của GD (C tối ưu nhất)



So sánh Gradient Descent với các hiện tượng vật lý

Chúng ta cần tính lượng thay đổi ở thời điểm $\boldsymbol{t}$ để cập nhật vị trí mới cho nghiệm (tức hòn bi). Nếu chúng ta coi đại lượng này như vận tốc $\boldsymbol{v_t}$ trong vật lý, vị trí mới của hòn bi sẽ là $\boldsymbol{w_{t+1} = w_t - v_t}$. Dấu trừ thể hiện việc di chuyển ngược với đạo hàm.

Công việc của chúng ta bây giờ là tính đại lượng $\boldsymbol{v_t}$ sao cho nó vừa mang thông tin của độ dốc (tức đạo hàm), vừa mang thông tin của đà, tức vận tốc trước đó $\boldsymbol{v_{t-1}}$ (chúng ta coi như vận tốc ban đầu $\boldsymbol{v_0 = 0}$). Một cách đơn giản nhất, ta có thể cộng (có trọng số) hai đại lượng này lại:

$$\boldsymbol{v_t = \gamma v_{t-1} + \epsilon \nabla_w J(w)}$$

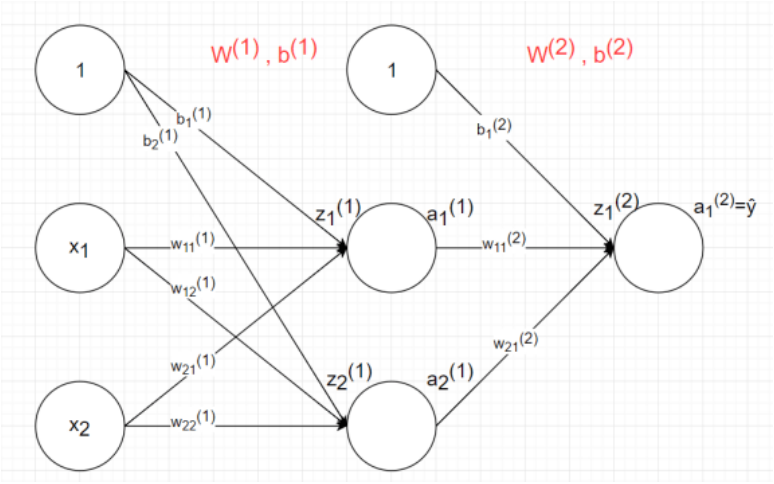
$$\boldsymbol{\rightarrow \quad w_{t+1} = w_t - v_t}$$

Trong đó $\boldsymbol{\gamma}$ thường được chọn là một giá trị khoảng 0.9, $\boldsymbol{v_t}$ là vận tốc tại thời điểm trước đó, $\boldsymbol{\nabla_w J(w)}$ chính là độ dốc của điểm trước đó.

Thuật toán đơn giản này tỏ ra rất hiệu quả trong các bài toán thực tế (trong không gian nhiều chiều, cách tính toán cũng hoàn tòan tương tự). Dưới đây là một ví dụ trong không gian một chiều.

3.2 Backpropagation

Giả sử ta có mô hình Neural Network như sau:



Neural Network có 2 lớp

Với feed for ward ta có:

- $z_1^{(1)} = b_1^{(1)} + x_1 . w_{11}^{(1)} + x_2 . w_{21}^{(1)} \rightarrow a_1^{(1)} = \sigma(z_1^{(1)})$
- $z_2^{(1)} = b_1^{(1)} + x_1 . w_{12}^{(1)} + x_2 . w_{22}^{(1)} \rightarrow a_2^{(1)} = \sigma(z_2^{(1)})$
- $z_1^{(2)} = b_1^{(2)} + a_1^{(1)} . w_{11}^{(2)} + a_2^{(1)} . w_{21}^{(2)}$
- $\rightarrow \hat{y} = a_1^{(2)} = \sigma(z_1^{(2)})$

Các trọng số không có sẵn mà phải random chính là:

- $\begin{bmatrix} 1 & w_{11}^{(1)} & w_{12}^{(1)} \\ 1 & w_{21}^{(1)} & w_{22}^{(1)} \end{bmatrix} = W^{(1)}$
- $\begin{bmatrix} 1 & w_{11}^{(2)} & w_{21}^{(2)} \end{bmatrix} = W^{(2)}$

Công việc bây giờ chính là tìm ma trận $W^{(1)}, W^{(2)}$ qua Gradient Descent với hàm lỗi – loss function như sau:

- $L = -(y_i * log(\hat{y}_i) + (1 - y_i) * log(1 - \hat{y}_i))$
- $J = -\sum_{i=1}^{N} (y_i * log(\hat{y}_i) + (1 - y_i) * log(1 - \hat{y}_i))$

Theo Gradient Descent thì đúng ra ta phải tính đạo hàm của hàm tổng lỗi ***J*** chứ không phải hàm lỗi **L**. Tuy nhiên, cơ bản thì hàm ***J*** cũng chỉ là tổng các giá trị của hàm **L** nên nó không ảnh hưởng gì đến việc tính đạo hàm. Hay nói cách khác, ta chỉ cần tính đạo hàm của **L** rồi tính tổng thì sẽ được đạo hàm của ***J***.

Vậy giả sử ta muốn biết sau mỗi bước của Gradient Descent thì giá trị của $w_{11}{}^{(2)}$ sẽ thay đổi như thế nào, thì cần phải đi tính đạo hàm của hàm lỗi L theo $w_{11}{}^{(2)}$.

Áp dụng chain rule: $\frac{\partial L}{\partial w_{11}{}^{(2)}} = \frac{\partial L}{\partial \hat{y}_i} * \frac{\partial \hat{y}_i}{\partial z_1{}^{(2)}} * \frac{\partial z_1{}^{(2)}}{\partial w_{11}{}^{(2)}}$

- $\frac{\partial L}{\partial \hat{y}_i} = -\left(\frac{y_i}{\hat{y}_i} - \frac{(1-y_i)}{(1-\hat{y}_i)}\right)$
- $\frac{\partial \hat{y}_i}{\partial z_1{}^{(2)}} = \sigma\left(z_1{}^{(2)}\right) * \left(1 - \sigma\left(z_1{}^{(2)}\right)\right) = \hat{y}_i * (1 - \hat{y}_i)$
- $\frac{\partial z_1{}^{(2)}}{\partial w_{11}{}^{(2)}} = a_1{}^{(1)}$
- $\rightarrow \quad \boldsymbol{\frac{\partial L}{\partial w_{11}{}^{(2)}} = -\left(\frac{y_i}{\hat{y}_i} - \frac{(1-y_i)}{(1-\hat{y}_i)}\right) * \hat{y}_i(1-\hat{y}_i) * a_1{}^{(1)} = (\hat{y}_i - y_i) * a_1{}^{(1)}}$

Hoàn toàn tương tự, ta tính đạo hàm của L theo $w_{21}{}^{(2)}$ và $b_1{}^{(2)}$

- $\frac{\partial L}{\partial w_{21}{}^{(2)}} = \frac{\partial L}{\partial \hat{y}_i} * \frac{\partial \hat{y}_i}{\partial z_1{}^{(2)}} * \frac{\partial z_1{}^{(2)}}{\partial w_{21}{}^{(2)}} = (\hat{y}_i - y_i) * a_2{}^{(1)}$
- $\frac{\partial L}{\partial b_1{}^{(2)}} = \frac{\partial L}{\partial \hat{y}_i} * \frac{\partial \hat{y}_i}{\partial z_1{}^{(2)}} * \frac{\partial z_1{}^{(2)}}{\partial b_1{}^{(2)}} = (\hat{y}_i - y_i) * 1$
- $\rightarrow \boldsymbol{\frac{\partial L}{\partial W^{(2)}} = \left[(\hat{y}_i - y_i) \quad (\hat{y}_i - y_i) * a_1{}^{(1)} \quad (\hat{y}_i - y_i) * a_2{}^{(1)}\right]}$

Tính đạo hàm của L theo $w_{11}{}^{(1)}$ thì cần nhiều bước hơn nhưng cũng sử dụng chain rule như trên: $\frac{\partial L}{\partial w_{11}{}^{(1)}} = \frac{\partial L}{\partial \hat{y}_i} * \frac{\partial \hat{y}_i}{\partial z_1{}^{(2)}} * \frac{\partial z_1{}^{(2)}}{\partial a_1{}^{(1)}} * \frac{\partial a_1{}^{(1)}}{\partial z_1{}^{(1)}} * \frac{\partial z_1{}^{(1)}}{\partial w_{11}{}^{(1)}}$

- $\frac{\partial L}{\partial \hat{y}_i} * \frac{\partial \hat{y}_i}{\partial z_1{}^{(2)}} = (\hat{y}_i - y_i)$
- $\frac{\partial z_1{}^{(2)}}{\partial a_1{}^{(1)}} = w_{11}{}^{(2)}$
- $\frac{\partial a_1{}^{(1)}}{\partial z_1{}^{(1)}} = a_1{}^{(1)} * (1 - a_1{}^{(1)})$
- $\frac{\partial z_1{}^{(1)}}{\partial w_{11}{}^{(1)}} = x_1$
- $\rightarrow \quad \boldsymbol{\frac{\partial L}{\partial w_{11}{}^{(1)}} = (\hat{y}_i - y_i) * w_{11}{}^{(2)} * a_1{}^{(1)}.\left(1 - a_1{}^{(1)}\right) * x_1}$

Từ đó hoàn toàn tính được $\boldsymbol{\frac{\partial L}{\partial W^{(2)}}}$ và để đơn giản hơn, ta sẽ trình bày dưới dạng ma trận từ rút ra công thức tổng quát:

- $Z^{(1)} = X * W^{(1)} + b^{(1)} \rightarrow A^{(1)} = \sigma(Z^{(1)})$
- $Z^{(2)} = A^{(1)} * W^{(2)} + b^{(2)} \rightarrow \hat{Y} = A^{(2)} = \sigma(Z^{(2)})$
- $L = Loss(\hat{Y}) \rightarrow J = \sum Loss(\hat{Y})$

Khi đó:

- $\frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial \hat{Y}} * \frac{\partial \hat{Y}}{\partial Z^{(2)}} * \frac{\partial Z^{(2)}}{\partial W^{(2)}}$
- $= (\hat{\boldsymbol{Y}} - \boldsymbol{Y}) * (\boldsymbol{A^{(1)}})^{\boldsymbol{T}}$
- $\frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial \hat{Y}} * \frac{\partial \hat{Y}}{\partial Z^{(2)}} * \frac{\partial Z^{(2)}}{\partial A^{(1)}} * \frac{\partial A^{(1)}}{\partial Z^{(1)}} * \frac{\partial Z^{(1)}}{W^{(1)}}$
- $= \boldsymbol{X^T} * \left(\hat{\boldsymbol{Y}} - \boldsymbol{Y}\right) * \left(\boldsymbol{W^{(2)}}\right)^{\boldsymbol{T}} \otimes \boldsymbol{A^{(1)}} \otimes (\boldsymbol{1} - \boldsymbol{A^{(1)}})$

Neural Network càng nhiều lớp thì phải tính đạo hàm càng nhiều, và khi gần với lớp input thì việc tính toán trở nên rất phức tạp. Đây cũng chính là lý do các activation function là các hàm dễ đạo hàm.

4. Các khái niệm trong xử lí ảnh

4.1 Các loại ảnh

4.1a. Ảnh màu

Giả sử ta có một bức ảnh màu có kích thước m x n ta có thể biểu diễn dưới dạng ma trận [n x m] như sau:

$$\begin{bmatrix} w_{1,1} & \cdots & w_{1,m} \\ \vdots & \ddots & \vdots \\ w_{n,1} & \cdots & w_{n,m} \end{bmatrix}$$

Với mỗi phần $w_{i,j}$ là một pixel trong ảnh. Pixel còn gọi là điểm ảnh là một khối màu rất nhỏ và là đơn vị cơ bản nhất để tạo nên một bức ảnh kỹ thuật số.

Mỗi pixel biểu diễn một màu chứa 3 thông số trong hệ màu RGB, có thể gọi

$w_{i,j} = (r_{i,j},\ g_{i,j}, b_{i,j})$. Ta có thể biểu diễn ma trận tổng quát như sau:

$$\begin{bmatrix} (r_{1,1},\ g_{1,1}, b_{1,1}) & \cdots & (r_{1,m},\ g_{1,m},\ b_{1,m}) \\ \vdots & \ddots & \vdots \\ (r_{n,1},\ g_{n,1},\ b_{n,1}) & \cdots & (r_{n,m},\ g_{n,m},\ b_{n,m}) \end{bmatrix}$$

Để tiện cho việc lưu trữ và xử lý ảnh màu, ta tách ra thành ba ma trận tương ứng với ba kênh màu (red, green, blue) được biểu diễn như sau:

$$\begin{bmatrix} r_{1,1} & \cdots & r_{1,m} \\ \vdots & \ddots & \vdots \\ r_{n,1} & \cdots & r_{n,m} \end{bmatrix}, \begin{bmatrix} g_{1,1} & \cdots & g_{1,m} \\ \vdots & \ddots & \vdots \\ g_{n,1} & \cdots & g_{n,m} \end{bmatrix}, \begin{bmatrix} b_{1,1} & \cdots & b_{1,m} \\ \vdots & \ddots & \vdots \\ b_{n,1} & \cdots & b_{n,m} \end{bmatrix}$$

4.1.b. Ảnh đen trắng:

Tương tự như ảnh màu cũng sẽ có một bức ảnh có kích thước m x n và biểu diễn dưới dạng ma trận n x m như sau:

$$\begin{bmatrix} w_{1,1} & \cdots & w_{1,m} \\ \vdots & \ddots & \vdots \\ w_{n,1} & \cdots & w_{n,m} \end{bmatrix}$$

Với ảnh đen trắng mỗi pixel chứa một thông số là một giá trị nguyên thuộc [0, 255] ($w_{i,j} \in [0, 255]$). Chính vì thế khi biểu diễn ảnh đen trắng trên máy tính ta chỉ cần một ma trận là đủ.

Giá trị 0 là màu đen, 255 là màu trắng và giá trị pixel càng gần 0 thì càng tối và càng gần 255 thì càng sáng.

Ví dụ về ma trận biểu diễn : $\begin{bmatrix} 0 & \cdots & 255 \\ \vdots & \ddots & \vdots \\ 66 & \cdots & 128 \end{bmatrix}$

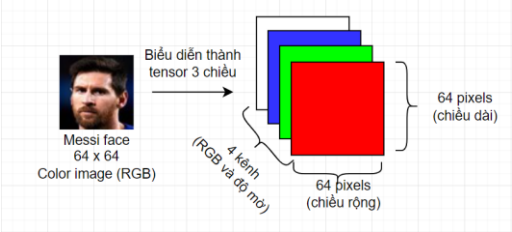
4.2 Tensor

Tensor có thể hiểu là dữ liệu có dạng nhiều hơn hai chiều bao gồm các ma trận xếp lên nhau.

Chẳng hạn, như đã giả sử ở bên ảnh màu m x n, ta sẽ biểu diễn ảnh thành dạng ma trận n x m và tách ma trận ra thành 3 ma trận tương ứng với ba kênh màu như sau:

$$\begin{bmatrix} r_{1,1} & \cdots & r_{1,m} \\ \vdots & \ddots & \vdots \\ r_{n,1} & \cdots & r_{n,m} \end{bmatrix}, \begin{bmatrix} g_{1,1} & \cdots & g_{1,m} \\ \vdots & \ddots & \vdots \\ g_{n,1} & \cdots & g_{n,m} \end{bmatrix}, \begin{bmatrix} b_{1,1} & \cdots & b_{1,m} \\ \vdots & \ddots & \vdots \\ b_{n,1} & \cdots & b_{n,m} \end{bmatrix}$$

Từ đây, ta sẽ xếp 3 ma trận này lên nhau tạo thành một dữ liệu 3 chiều với kích thước là n x m x 3. Dữ liệu 3 chiều đó được gọi là tensor.



Ảnh Messi có kích thước 64 x 64

4.3 Phép tính Convolution (Phép tích chập trong ảnh)

4.3.a. Kernel:

Là bộ lọc có thể hiểu đó là một ma trận vuông có kích thước là k x k (với k là số lẻ), giá trị ban đầu của bộ lọc được random một cách ngẫu nhiên.

Ví dụ: 1 kernel có kích thước 3 x 3.

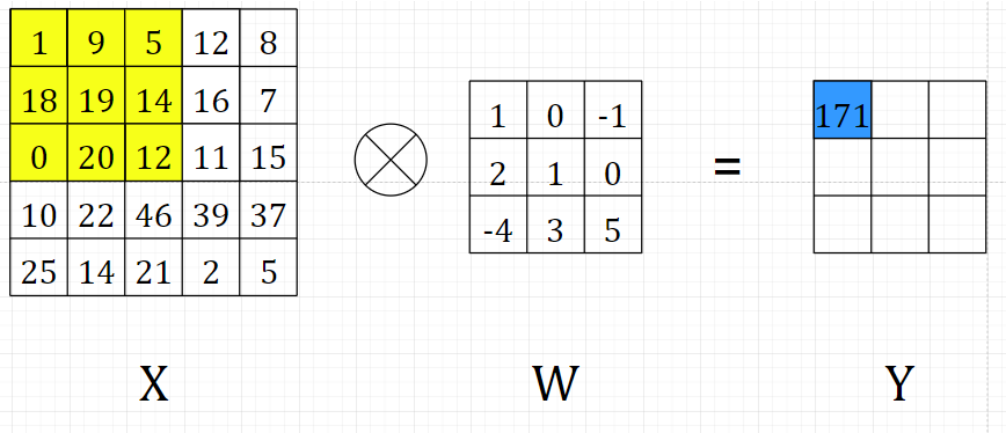
$$W = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

4.3.b. Tích chập (Convolution)

Với mỗi phần tử của ma trận gốc (ta gọi là ma trận X) ta lấy ra một ma trận con có kích thước bằng kích thước của kernel W có một phần tử khác của ma trận là trung tâm.

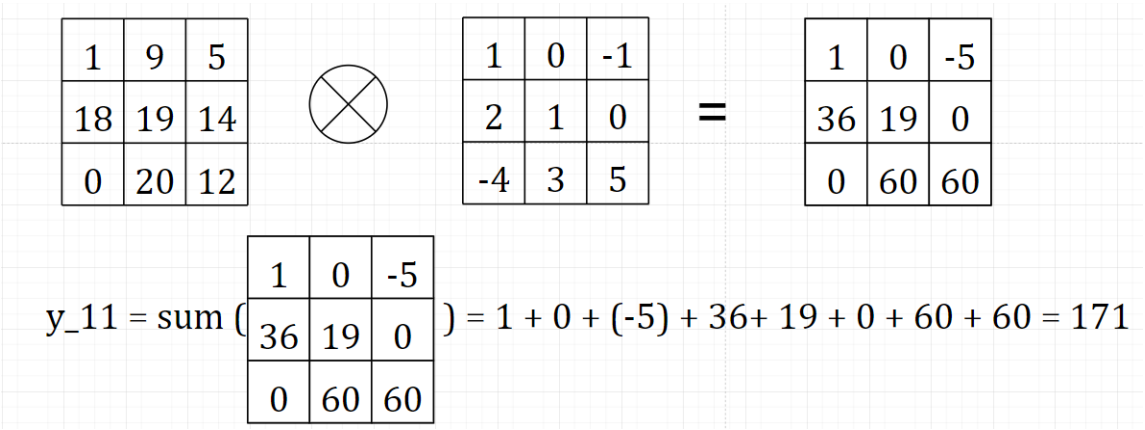
Sau đó ta tính tổng của các phần tử của ma trận sau khi thực hiện phép tính tương quan giữa ma trận X và ma trận W (gọi là element-wise) rồi ta điền vào ma trận Y. Việc thực hiện này được gọi là tích chập (convolution). Ký hiệu là X ⊗ W = Y

VD:



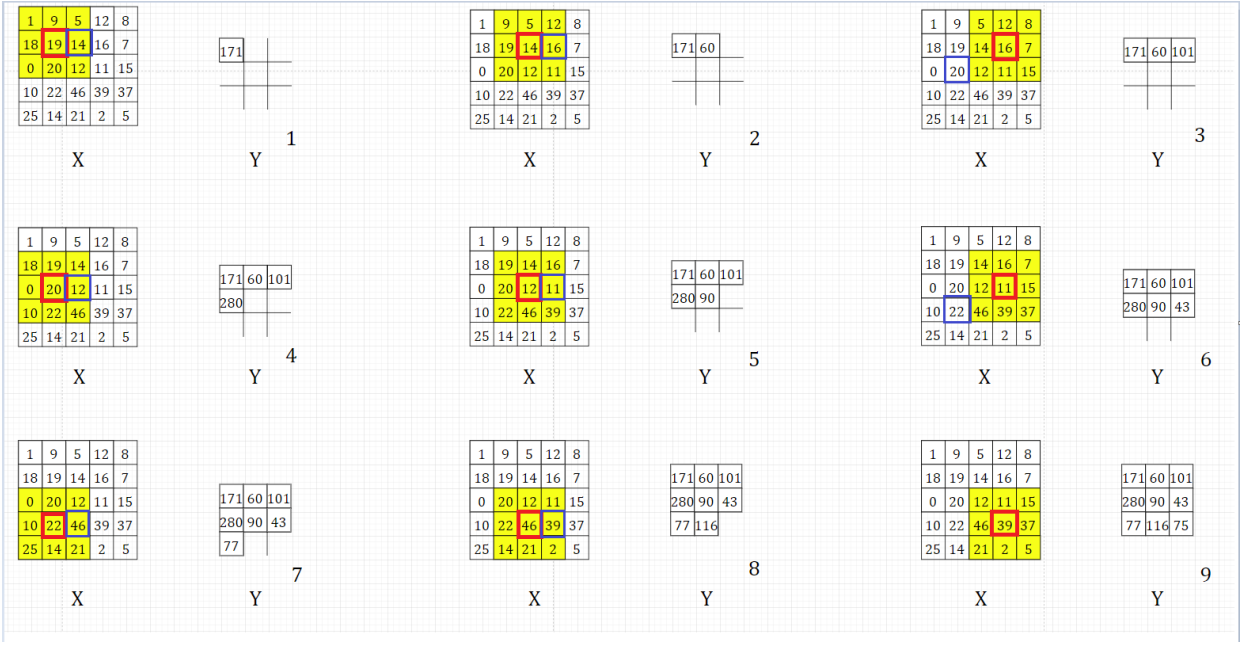
X có kích thước 5 ∗ 5, *kernel W có kích thước* 3 ∗ 3, padding = 0, stride = 1

Giá trị y_{11} sẽ được tính như sau:



Cách thực hiện phép tích chập cho phần tử y_11

Các phần tử còn lại cũng thực hiện tương tự:



Thứ tự các bước tích chập

Sau khi ra ma trận Y, chúng ta có thể thấy rằng ma trận Y có kích thước nhỏ hơn ma trận X. Kích thước của ma trận Y sẽ là $(\mathbf{m} - \mathbf{k} + \mathbf{1}) * (\mathbf{n} - \mathbf{k} + \mathbf{1})$.

4.3.c. Padding

Sau khi thực hiện phép tích chập giữa ảnh gốc (ma trận X) và bộ lọc (Kernel) W thì như chúng ta đã thấy kết quả cho ra ảnh mới (ma trận Y) có kích thước nhỏ hơn kích thước của ảnh gốc (ma trận X).

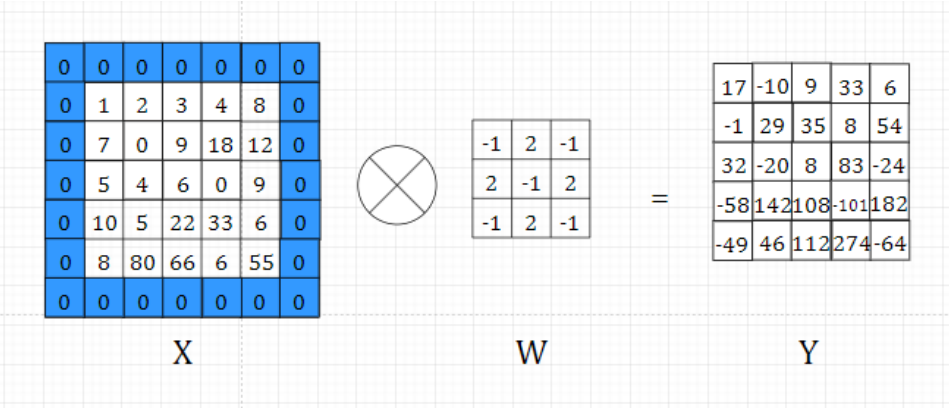
Vì vậy để khắc phục tình trạng này, ta sẽ thêm p ($p \geq 0$) đường viền có giá trị bằng 0 để sau khi tính phép tích chập sẽ làm cho ảnh mới (ma trận Y) được tạo ra sẽ giữ nguyên được đặc trưng của ảnh gốc (ma trận X), nó được gọi là padding = p

VD1:

0	0	0	0	0	0	0
0	1	2	3	4	8	0
0	7	0	9	18	12	0
0	5	4	6	0	9	0
0	10	5	22	33	6	0
0	8	80	66	6	55	0
0	0	0	0	0	0	0

Ma trận 5 ∗ 5 với padding = 1 cho ra ma trận output là 7 ∗ 7

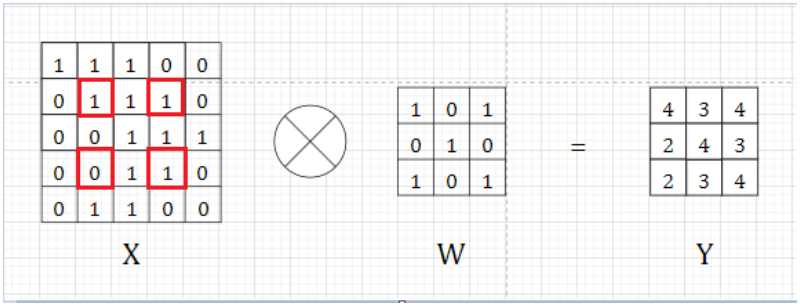
VD2:



Kết quả của phép tích chập giữa Ma trận 5 ∗ 5 với padding = 1 và kernel 3 ∗ 3

4.3.d. Sải bước (Stride)

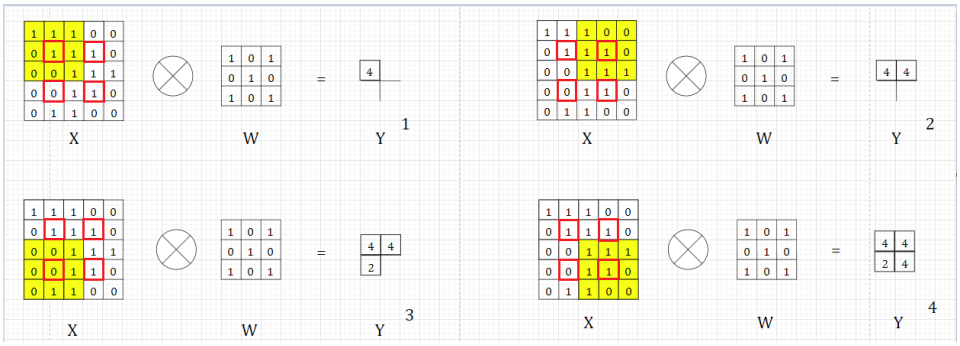
Khi thực hiện phép tích chập giữa ma trận X và kernel W, ta bắt đầu lấy ra một ma trận có kích thước bằng kích thước của kernel W từ góc trái trên rồi di chuyển sang bên phải hoặc di chuyển xuống. Ta gọi việc di chuyển này gọi là sải bước (stride). Ở các VD trước, ta di chuyển một cột hoặc di chuyển xuống một hàng khi tính tích chập, nghĩa là stride = 1.



Kết quả output Y với X có kích thước $5 * 5$, kernel W có kích thước $3 * 3$, padding = 0, stride = 2

Tuy nhiên nếu stride > 1 thì ta chỉ cần thực hiện phép tính convolution trên các phần tử $x_{1+i*stride,1+j*stride}$

VD:



*Thứ tự các bước tích chập với input 5 * 5, kernel 3 * 3, padding = 0, stride = 2*

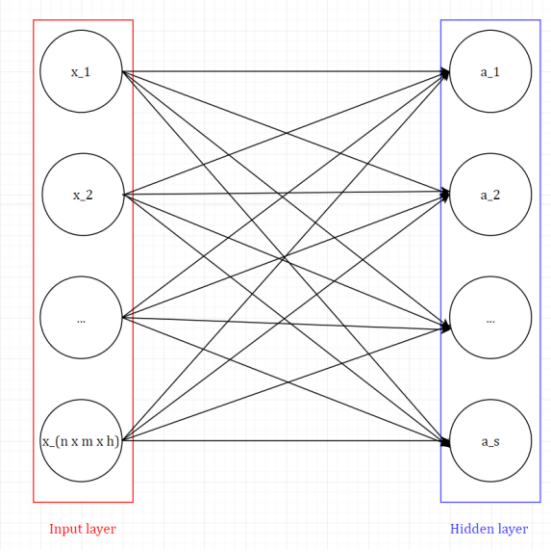
Từ đó ta thấy rằng so với stride = 1 thì kích thước của ma trận Y ở stride = 2 nhỏ hơn so với stride = 1 và ở stride = 1 ta phải tốn 9 bước mới ra được ma trận Y trong khi stride = 2 chỉ có 4.

Như vậy ta có thể kết luận rằng khi ta chọn bước sải (stride) càng cao thì khi thực hiện phép tích chập ta cho ra kết quả có kích thước càng thấp và hiệu suất tính toán cũng sẽ tăng lên (ở đây là giảm thời gian trong việc tính toán).

5. Convolution Neuron Network (CNN)

5.1. Vấn đề của fully connected neural network với xử lý ảnh

Giả sử ta có một bức ảnh kích thước 600 * 800 pixel, khi đó tổng số pixel của bức hình là 480,000 pixel. Nếu ta coi một pixel là một thuộc tính thì input layer sẽ có 480,000 nodes. Và lớp input sẽ được biểu diễn dưới dạng ma trận 480,000 * 1.



Từng node ở input layer liên kết với tất cả các node trong hidden layer

Khi đó, nếu ở lớp ẩn đầu tiên (Hidden layer) có 1000 thuộc tính thì ma trận trọng số W sẽ có kích thước 480,000 * 1000 và sẽ có 480,000,000 phần tử. Vậy vấn đề của fully connected neural network chính là:

- Số lượng node của các lớp rất lớn, dẫn đến số lượng phần tử của trọng số W tăng theo cấp số nhân. Khó để kiểm soát.
- Khi một bức ảnh $\boldsymbol{m} * \boldsymbol{n}$ được đưa về dạng vector $\boldsymbol{mn} * \boldsymbol{1}$ thì ta đã làm mất đi các đặc trưng về không gian của bức ảnh.

Nhận xét:

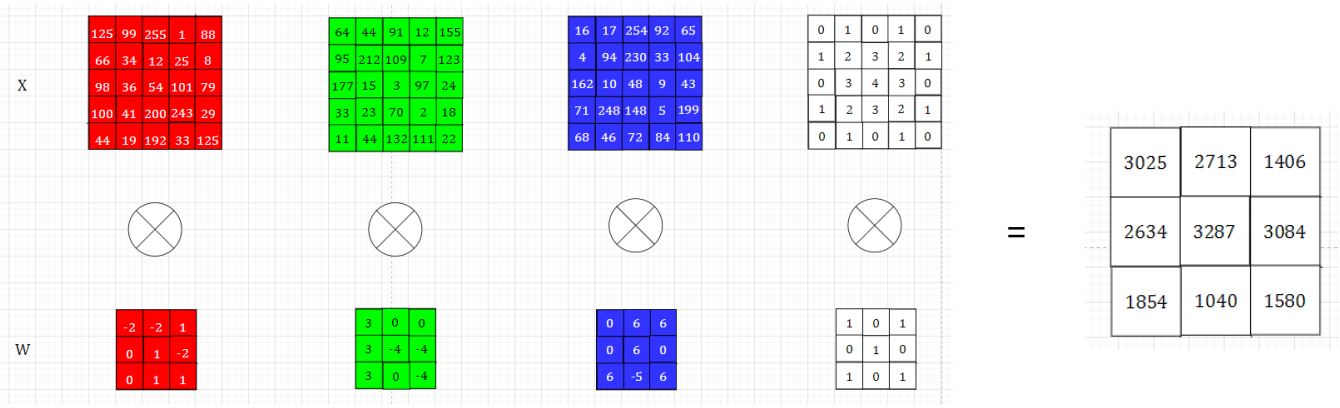
- Trong các ảnh pixel ở cạnh nhau thường có liên kết với nhau hơn là những pixel ở xa.
- Trong phép tính convolution trong ảnh, chỉ 1 kernel được dùng trên toàn bộ bức ảnh. Hay nói cách khác là các pixel ảnh chia sẻ hệ số với nhau.

Áp dụng phép tính convolution trong neural network ta có thể giải quyết được vấn đề lượng lớn parameter mà vẫn lấy ra được các đặc trưng của ảnh.

5.2. Convolution layer đầu tiên

Từ ảnh $\boldsymbol{m} * \boldsymbol{n}$ ta biểu diễn thành 1 tensor 3 chiều có kích thước là n x m x c. Từ tensor 3 chiều này ta sẽ định nghĩa kernel là một tensor 3 chiều(shape $\boldsymbol{k\_n} * \boldsymbol{k\_m} * \boldsymbol{c}$). Tổng quát hơn là ta sẽ định nghĩa kernel là một tensor có cùng độ sâu với tensor được biểu diễn của ảnh, sau đó ta sẽ di chuyển khối kernel đó theo stride tương ứng. Khi biểu diễn ở dạng tensor c chiều cần thêm chỉ số độ sâu là k' nên chỉ số của mỗi phần tử trong tensor là $x_{ijk'}$.

VD:

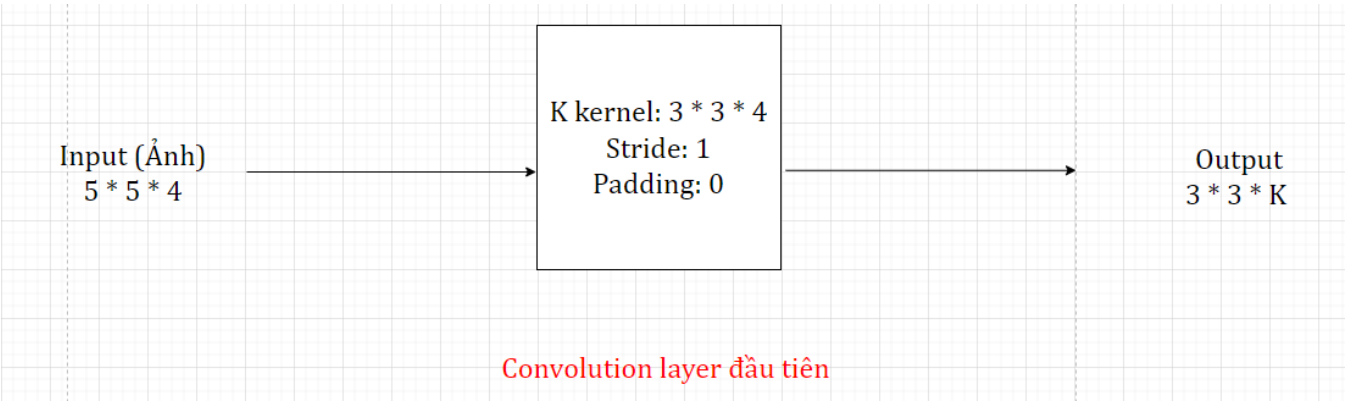


*Input 5 * 5 * 4 và kernel 3 * 3 với padding = 0, stride = 1*

Sau khi ta thực hiện phép convolution ở convolution layer đầu tiên ta thấy:

- Output Y là một ma trận .
- Có một **hệ số bias** được cộng vào sau khi tính tổng các phần tử của phép tính element-wise.

Các quy tắc padding và stride hoàn toàn tương tự như lúc thực hiện phép tích chập. Output của convolution layer đầu tiên sẽ thành input của convolution layer tiếp theo.



*Mô hình từ input (ảnh) 5 * 5 * 4 qua convolution layer đầu tiên ra output*

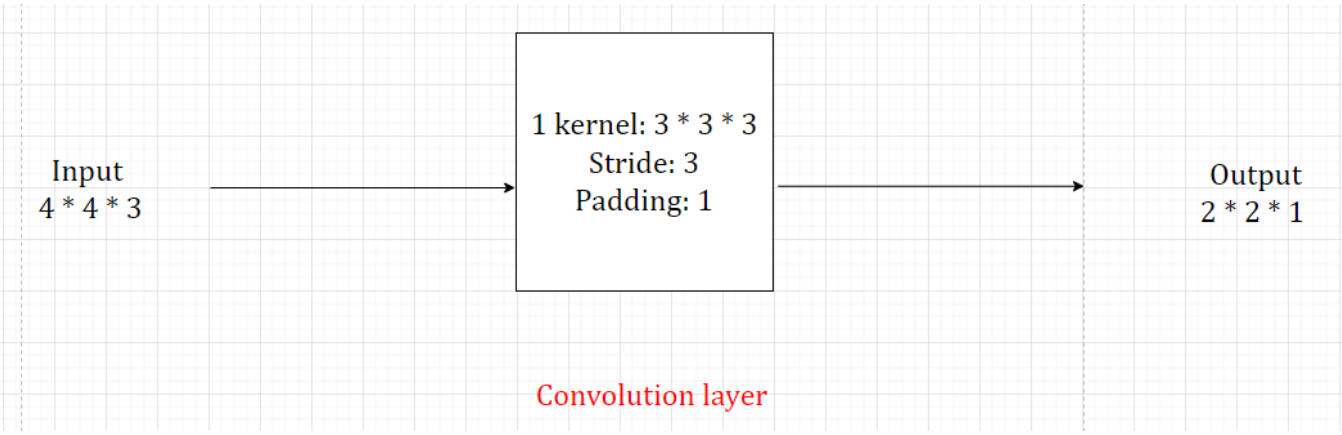
5.3. Convolution layer tổng quát

Giả sử input của 1 convolution layer tổng quát là tensor kích thước $h * w * d$.

Kernel có kích thước là $k * k * d$ (kernel luôn có depth bằng depth của input và k luôn là số lẻ), stride s, padding p.

Convolutional layer áp dụng K kernel => Output của layer là tensor 3 chiều có kích thước là: $\left(\frac{h-k+2p}{s}+1\right) \mathbf{x} \left(\frac{w-k+2p}{s}+1\right) \mathbf{x} K$.

VD:



*Mô hình từ input (ảnh) 4 * 4 * 3 qua một lớp convolution layer nào đó cho ra output*

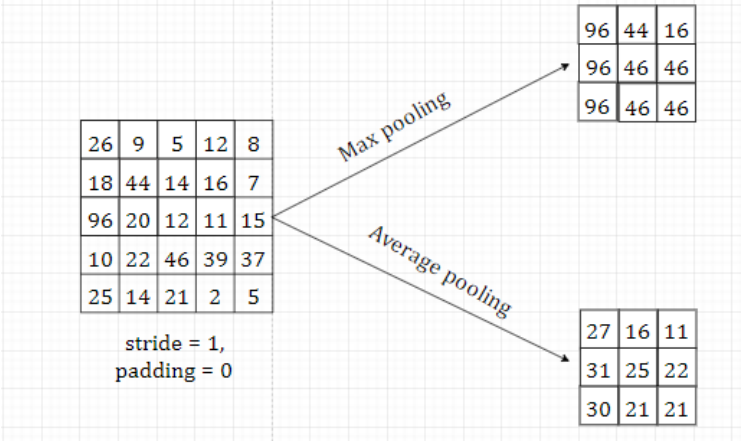
Lưu ý:

- Output của convolutional layer sẽ qua hàm activation function trước khi trở thành input của convolutional layer tiếp theo.
- Mỗi kernel có kích thước $k * k * d$ và có một hệ số bias, nên tổng parameter của 1 kernel là $k * k * d + 1$. Mà convolutional layer áp dụng K kernel => Tổng parameter trong một layer là $K * (k * k * d + 1)$.

5.4. Pooling layer (Lớp tổng hợp)

Pooling layer (lớp tổng hợp) thường được dùng giữa các convolution layer để giảm kích thước dữ liệu nhưng vẫn giữ được các thuộc tính quan trọng. Việc giảm kích thước dữ liệu sẽ làm giảm thời gian trong việc tính toán model.

Có 2 loại pooling layer phổ biến: max pooling và average pooling

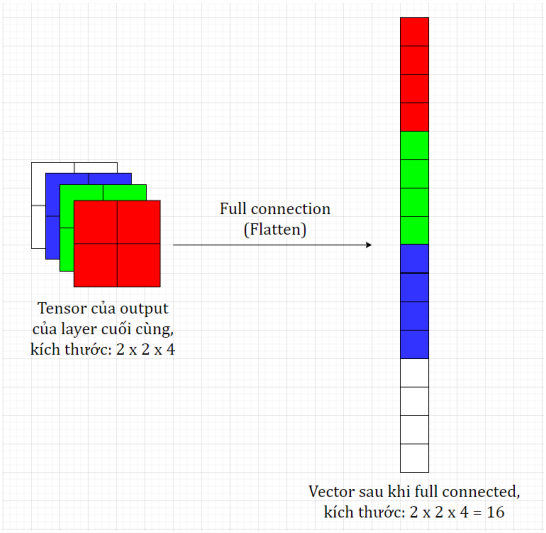


Kết quả sau khi max pooling và average pooling từ input $5 * 5$*, padding = 0, stride = 1*

Trong một số model người ta dùng convolution layer với stride > 1 để giảm kích thước dữ liệu thay cho pooling layer.

5.5. Fully connected layer

Sau khi ảnh được truyền qua nhiều convolution layer và pooling layer thì model đã học được tương đối các đặc điểm của ảnh, cho nên từ tensor của output của layer cuối cùng với kích thước là $\boldsymbol{h} * \boldsymbol{w} * \boldsymbol{d}$ sẽ chuyển về 1 vector có kích thước là $\boldsymbol{h} * \boldsymbol{w} * \boldsymbol{d}$. Sau đó ta dùng các fully connected layer để kết hợp các đặc điểm của ảnh để ra được output của model. VD:

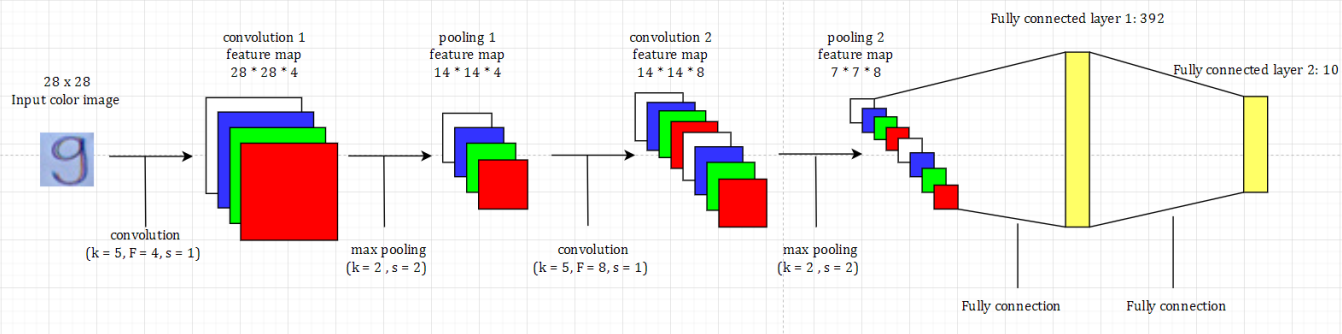


Kết quả sau khi full connected từ tensor của layer cuối cùng là một vector với kích thước 16

5.6. Visualise convolution neural network

Mô hình Convolution neural network:

Input image -> Convolution layer + Pooling layer -> Fully connected layer -> Output image



Mô hình Convolution neural network từ ảnh màu số 9 với kích thước 28 ∗ 28

6. AlexNet

6.1 Giới thiệu

Các phương pháp tiếp cận hiện đại để nhận dạng đối tượng sử dụng thiết yếu các phương pháp học máy. Để cải thiện hiệu suất của chúng, chúng ta có thể thu thập các tập dữ liệu lớn hơn, tìm hiểu các mô hình hiệu quả hơn và sử dụng các kỹ thuật tốt hơn. Cho đến gần đây, bộ dữ liệu của hình ảnh được dán nhãn là tương đối nhỏ và các tác vụ nhận dạng đơn giản có thể được giải quyết khá tốt với bộ dữ liệu có kích thước tương đối nhỏ này, đặc biệt là nếu chúng được tăng cường với các phép biến đổi bảo quản nhãn. Chẳng hạn như tỷ lệ lỗi tốt nhất hiện tại đối với tác vụ nhận dạng một chữ số MNIST (< 0.3 %) tiệm cận với hiệu suất của con người.

Ở hình trong phần 5.6, chúng ta có thể thấy được từng bước thực hiện của một mô hình Neural tích chập (CNN) từ một hình số 9 (28 ∗ 28) cho ra output là số 9 được nhận dạng ra. Nhưng trong bối cảnh thực tế hiện nay, đối tượng không phải chỉ là một con số hay một hình ảnh với kích thước tương đối nhỏ là 28 ∗ 28, đối tượng có thể lớn hơn về mặt kích thước hay số lượng hoặc hình dạng của đối tượng cũng có thể thay đổi, vì vậy để học cách nhận dạng ra chúng, ta cần phải sử dụng các tập huấn luyện lớn hơn nhiều. Và để tìm hiểu về hàng nghìn đối tượng từ hàng triệu hình ảnh, chúng ta cần một mô hình có khả năng học lớn. Tuy nhiên, độ phức tập to lớn của nhiệm vụ nhận dạng đối tượng có nghĩa là vấn đề này không thể được chỉ định ngay cả với tập dữ liệu lớn như ImageNet, vì vậy mô hình cũng cần có nhiều kiến thức trước đó để bù đắp cho tất cả dữ liệu bị thiếu.

Và để có thể thỏa mãn thực hiện nhận dạng mọi đối tượng dù lớn hay nhỏ cũng như là đáp ứng nhu cầu cho việc khắc phục những dữ liệu bị thiếu trong nhận dạng đối tượng, vào năm 2012, một mô hình mạng CNN với tên gọi là AlexNet, mô hình được đặt theo tên của tác giả chính của nhóm nghiên cứu là Alex Krizhevsky đã được giới thiệu và cũng chính mô hình này đã thắng hạng nhất trong cuộc thi ILSVRC cũng vào năm đó.

Mô hình giải quyết bài toán phân lớp một bức ảnh vào một lớp trong 1000 lớp khác nhau (ví dụ gà, chó, mèo …). Đầu ra của mô hình là một vector có 1000 phần tử. Phần tử thứ i của vector đại diện cho xác suất bức ảnh thuộc về lớp i. Do đó, tổng của các phần tử trong vector là 1. Đây là tiền đề cho sự phát triển của mô hình CNN nói riêng và Deep Learning nói chung.

6.2 Dataset

ImageNet là một bộ dữ liệu gồm hơn 15 triệu hình ảnh có độ phân giải cao được dán nhãn thuộc khoảng 22000 danh mục. Các hình ảnh được thu thập từ web và được gắn nhãn bởi người dán nhãn bằng công cụ tìm nguồn cung ứng đám đông Mechanical Turk của Amazon. Ở cuộc thi ILSVRC năm 2012, ILSVRC sử dụng một tập hợp con của ImageNet với khoảng 1000 hình ảnh trong mỗi 1000 danh mục. Tổng cộng, có khoảng 1,2 triệu hình ảnh đào tạo, 50000 hình ảnh xác thực và 150000 hình ảnh thử nghiệm. Theo nhóm tác giả AlexNet đã phân tích, trên ImageNet, thông thường sẽ báo cáo hai tỷ lệ lỗi: top-1 và top-5, trong đó tỷ lệ lỗi top-5 là tỷ lệ hình ảnh thử nghiệm mà nhãn chính xác không nằm trong số năm nhãn được mô hình coi là có thể xảy ra nhất. Hơn nữa, ImageNet bao gồm các hình ảnh có độ phân giải thay đổi nên AlexNet lúc đó không thể xử lý trước hình ảnh theo các kích thước khác nhau vì hệ thống này chỉ yêu cầu kích thước đầu vào không thay đổi.

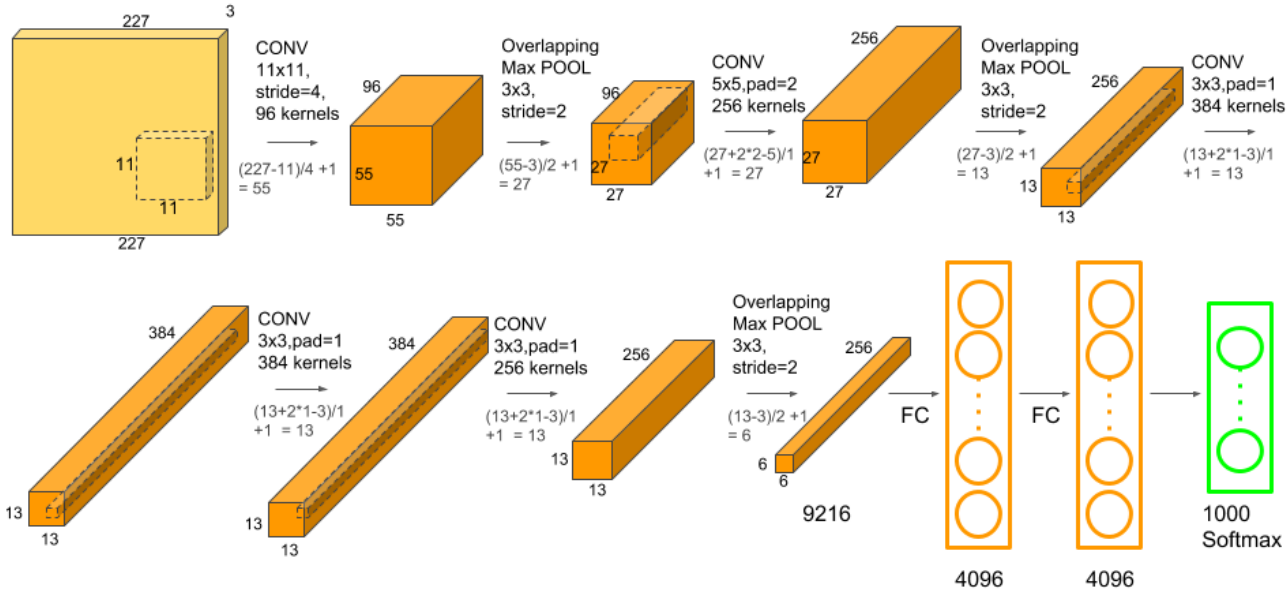
Chính vì sự thay đổi phân giải đó, nhóm tác giả chọn các mẫu ảnh bất kì (đó cũng chính là đầu vào của mạng AlexNet), sau đó thực hiện cắt đi những phần thừa của bức ảnh hoặc phóng to bức ảnh lên cho đến khi cùng kích thước là $256 * 256$ cũng như chuyển từ ảnh xám (grayscale) thành ảnh màu RGB. Điều đó có nghĩa toàn bộ các bức ảnh của tập train và tập test đều có cùng kích thước là $256 * 256$.

Ví dụ:

Input là ảnh kích thước $1024 * 500$ *và output là anh kích thước là* $256 * 256$

Sau khi chuẩn hóa hết tất cả các ảnh về dạng 256 * 256 * 3, nhóm tác giả chỉ sử dụng một phần của bức ảnh có kích thước 227 * 227 * 3 của một bước ảnh làm đầu vào cho mạng Neural network.

6.3 Kiến trúc AlexNet:



(Kiến trúc AlexNet)

6.3.1. Sơ lược về kiến trúc AlexNet:

Kiến trúc AlexNet lớn hơn nhiều so với các kiến trúc CNNs được sử dụng trong thị giác máy tính trước kia (trước năm 2010). VD: kiến trúc LeNet của Yann LeCun năm 1998 (số lượng các kênh tích chập trong AlexNet nhiều hơn gấp mười lần so với LeNet).

Có 60 triệu tham số và 650.000 neural và tốn khoảng từ năm đến sáu ngày huấn luyện trên hai GPU GTX 580 3GB.

Các layer thấp nhất có thể dùng để phát hiện biên, màu sắc và đường nét như các bộ lọc truyền thống. được tái tạo từ bài báo khoa học trên mô tả các đặc trưng cấp thấp của hình ảnh. Các layer cao hơn của mạng sẽ dựa vào các biểu diễn này để thể hiện các cấu trúc lớn hơn như mắt, mũi, ngọn cỏ,... Thậm chí các tầng cao hơn nữa có thể đại diện cho nguyên một vật thể như con người, máy bay, chó hoặc là đĩa ném. Sau cùng, tầng trạng thái ẩn cuối sẽ học cách biểu diễn cô đọng của toàn bộ hình ảnh để tổng hợp lại nội dung sao cho dữ liệu thuộc các lớp khác nhau có thể được dễ dàng phân biệt.

6.3.2. Các layer của AlexNet:

AlexNet bao gồm 5 conv layer và 3 fully connected layers. Những conv layer (filter) rút trích các thông tin hữu ích trong các bức ảnh. ReLU nonlinerity được sử dụng sau tất các các conv và fully connected layer.

Về kích thước kernel:

- Conv layer: trong conv1 của AlexNet, kích thước kernel là $11 * 11$. Vì hầu hết các ảnh trong ImageNet đều có chiều cao và chiều rộng lớn gấp hơn mười lần so với các ảnh trong MNIST, các vật thể trong dữ liệu ImageNet thường có xu hướng chiếm nhiều điểm ảnh hơn. Do đó, ta cần sử dụng một kernel lớn hơn để xác định được các vật thể này. Kích thước kernel trong các layer sau được giảm xuống còn $5 * 5$ và $3 * 3$. Ngoài ra, theo sau các conv1, 2, 5 là các overlapping max pooling với kích thước kernel là $3 * 3$ và stride bằng 2.
- Overlapping pooling layer: $3 * 3$ và stride = 2.

Các layer trong AlexNet:

- Conv1 và conv2 kết nối với nhau qua một overlapping max pooling ở giữa. Tương tự như vậy giữa conv2 và conv3.
- Conv3, conv4, conv5 kết nối trực tiếp với nhau, không thông qua trung gian.

- Conv5 kết nối fully connected layter 1 thông qua một overlapping max pooling. Tiếp theo mà một fully connected layter nữa, hai tầng FC tạo ra đến 1GB các tham số mô hình.
- Và cuối cùng là một bộ phân lớp softmax với 1000 labels.

6.3.3. Overlapping Max Pooling:

Mục đích: giảm chiều rộng và chiều dài của một tensor nhưng vẫn giữ nguyên chiều sâu.

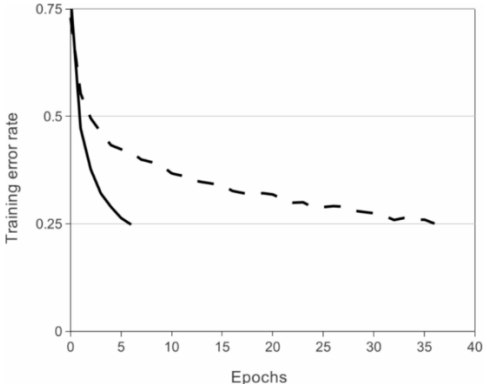
Cải tiến: overlapping max pool cũng tương tự như max pooling, ngoại trừ việc là một kernel của bước này sẽ có một phần chồng lên kernel của bước tiếp theo. Sử dụng pooling có kích thước $3 * 3$ và stride là 2 giữa các pooling. Nghĩa là giữa pooling này và pooling khác sẽ overlapping với nhau 1 pixel.

Các thí nghiệm thực tế đã chứng minh rằng việc sử dụng overlapping giữa các pooling giúp giảm độ lỗi top-1 error 0.4% và top-5 error là 0.3% khi so với việc sử dụng pooling có kích thước $2 * 2$ và stride = 2 (vector output của cả hai đều có số chiều bằng nhau).

6.3.4 ReLu Nonlinearity:

Cải tiến: Trước đây, các người ta thường sử dụng hàm kích hoạt là hàm Tanh hoặc hàm Sigmoid để huấn luyên mô hình neural network. Một cải tiến quan trọng khác của AlexNet là việc sử dụng hàm phi tuyến ReLU.

Mục đích: AlexNet chỉ ra rằng, khi sử dụng ReLU, mô hình deep CNN sẽ huấn luyện nhanh hơn so với viêc sử dụng Tanh hoặc Sigmoid. Hình bên dưới được rút ra từ bài báo chỉ ra rằng với việc sử dụng ReLU, AlexNet đạt độ lỗi 25% trên tập huấn luyện và nhanh hơn gấp 6 lần so với mô hình tương tự nhưng sử dụng Tanh. Ngoài ra, việc ReLU không bị chặn trên bởi 1 khiến cho vấn đề vanishing gradient cũng được giải quyết phần nào.



% độ lỗi của ReLU (nét liền) và Tanh (nét đứt)

Giải thích:

- Như đã đề cập ở phần 2, Tanh và Sigmoid bị bão hòa ở hai đầu (độ dốc giảm) dẫn đến gradient descent hội tụ chậm khiến lan truyền ngược không cập nhật tham số mô hình. Và một vấn đề khác hai hàm này sử dụng phép mũ làm giảm hiệu suất tính toán.
- Đối với ReLU:
  - Phần $x > 0$: độc dốc luôn là 45^o dẫn đến gradient descent luôn bằng 1 và tham số luôn được cập nhật.
  - Phần $x \leq 0$: hầu hết các neural trong mạng đều có giá trị dương nên trường hợp này hiếm xảy ra. Tuy vậy vẫn có thể có Dying ReLU.
- Về vanishing gradient: Là vấn đề xảy ra khi huấn luyện các mạng neural nhiều lớp. Khi huấn luyện, giá trị đạo hàm là thông tin phản hồi của quá trình lan truyền ngược. Khi sử dụng Sigmoid hay Tanh, giá trị này trở nên vô cùng nhỏ tại các lớp nơ-ron đầu tiên khiến cho việc cập nhật trọng số mạng không thể xảy ra.

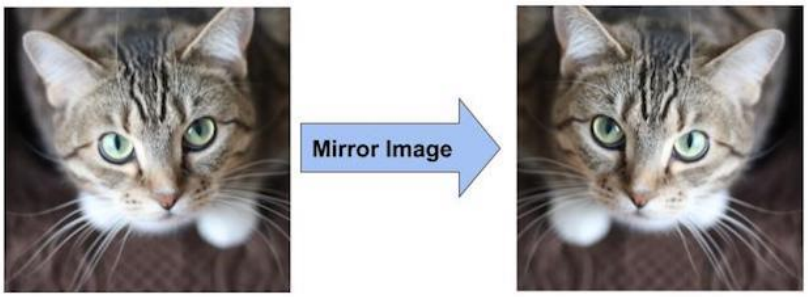
6.4 Xử lí overfitting:

Là hiện tượng neural network hoạt động tốt trên tập huấn luyện (nhưng chúng không rút ra được bản chất chính của vấn đề), và kết quả trên tập test tệ. Các phương pháp được dùng để giảm overfitting.

6.4.1 Data Augmentation:

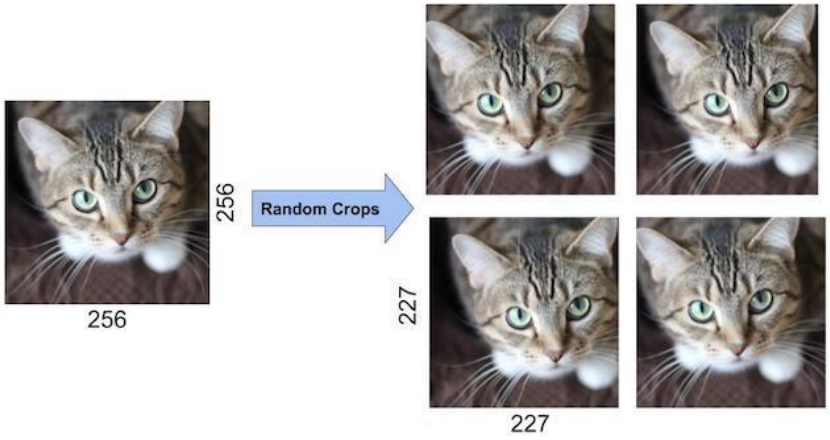
Việc sử dụng nhiều biến thể khác nhau của một bức hình có thể giúp ngăn mô hình không bị overfitting. Với việc sử dụng nhiều biến thể của 1 bức hình, bạn bắt ép mô hình không học vẹt dữ liệu. Có nhiều cách khác nhau để sinh ra dữ liệu mới dựa vào dữ liệu có sẵn. Một vài cách mà AlexNet đã sử dụng:

- *Data Augmentation by Mirroring:* Ý tưởng của việc này là lấy ảnh trong gương (đối xứng qua trục Oy) của một bức hình (ảnh ảo).



(Hình ảnh được tăng cường bằng cách đối xứng qua gương)

- *Data Augmentation by Random Crops:* Việc lựa chọn vị trí ảnh gốc một cách ngẫu nhiên cũng giúp chúng ta có thêm một ảnh khác so với ảnh gốc ban đầu.



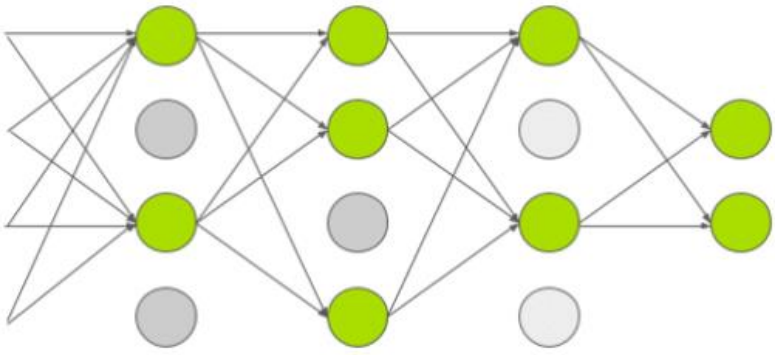
(Hình ảnh được tăng cường bằng cách lấy ở nhiều góc độ)

AlexNet rút trích ngẫu nhiên bức ảnh có kích thước $227 * 227$ từ bức ảnh $256 * 256$ ban đầu làm input dầu vào cho mô hình. Bằng cách này, chúng ta có thể tăng số lượng dữ liệu lên gấp 2048 lần.

Với việc sử dụng Data Augmentation, chúng ta đang bố gắng dạy cho mô hình rằng với việc nhìn hình con mèo qua gương, nó vẫn là con mèo, hoặc hình hình con mèo ở bất kỳ góc độ nào thì nó vẫn là nó.

6.4.2 Dropout:

Với gần 60 triệu tham số trong tập huấn luyện, việc overfitting xảy ra là điều dễ hiểu. Một kỹ thuật khác giúp giảm overfitting là dropout.



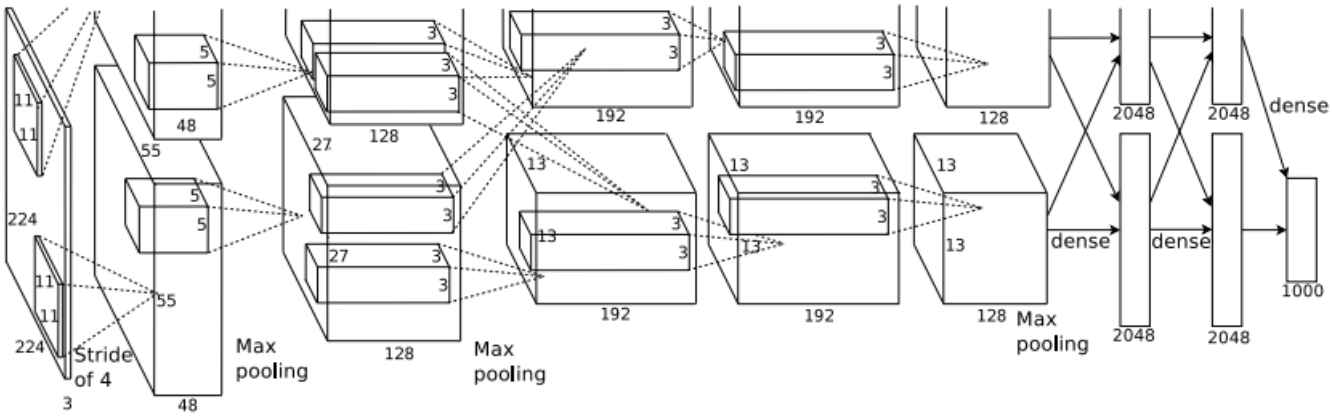
(Minh họa dropout với xác suất 0.5)

Quá trình trainning: kỹ thuật này khá đơn giản, một neural sẽ có xác suất bị loại khỏi mô hình là 0.5. Khi một neural bị loại khỏi mô hình, nó sẽ không được tham gia vào quá trình lan truyền tiến hoặc lan truyền ngược. Cho nên, mỗi giá trị input sẽ đi qua một kiến trúc mạng khác nhau. Kết quả là giá trị của tham số trọng số sẽ tốt hơn và khó bị overfitting hơn.

Quá trình testing: toàn bộ network được sử dụng, không có dropout, tuy nhiên, giá trị output sẽ scaled bởi tham số 0.5 tương ứng với những neural không sử dụng trong quá trình trainning.

Với việc sử dụng dropout, chúng ta sẽ tăng gấp đôi lần lặp cần thiết để đạt được độ hội tụ, nhưng khi không sử dụng dropout, mạng AlexNet rất dễ bị overfitting.

6.5 Quá trình học của mạng



Mô hình AlexNet sử dụng 2 GPU để huấn luyện

Mô hình AlexNet sử dụng hai GPU để huấn luyện với rất nhiều lớp. Vì vậy cần một thuật toán để đi tìm trọng số W - thuật toán Gradient Descent với kích thước là 128 ví dụ, và với độ quán tính (momentum) là 0.9, phân rã trọng lượng là 0.0005

- $\boldsymbol{v_{i+1} = 0.9 \times v_i - 0.0005 \times \epsilon \times w_i - \epsilon \times \langle \frac{\partial L}{\partial w} | w_i \rangle_{D_i}}$
- $\boldsymbol{w_{i+1} = w_i + v_{i+1}}$

Trong đó,

- $\boldsymbol{v}$ là quán tính - momentum
- $\boldsymbol{\epsilon}$ là learning rate
- $\langle \frac{\partial L}{\partial w} | w_i \rangle_{D_i}$ đạo hàm hàm lỗi theo w_i

Và để tìm được giá trị learning rate phù hợp, nhóm nghiên cứu của AlexNet đã phải điều chỉnh tỉ lệ này một cách thủ công trong suốt quá trình training. Bằng cách khởi tạo learning rate là 0.01 và sau đó đem chia cho 10 cho đến khi tỉ lệ của hàm lỗi ngừng cải thiện.

Quá trình train mạng diễn ra trong khoảng 90 chu kì với dữ liệu gồm 1,2 triệu hình ảnh và phải mất đến 5-6 ngày trên hai GPU NIVIDA GTX 580 3GB mới hoàn thành.

KẾT LUẬN

Kết quả:

Năm 2012, tại ILSVRC, Mô hình Deep Convolutional Neural Network – AlexNet đã đạt kết quả top-5 error rate 16% (nhóm tác giả đã giành hạng nhất), cách biệt khá xa so với kết quả nhóm thứ hai (error rate 26.2%). Kết quả này lãm sững sờ giới nghiên cứu trong khoảng thời gian đó.

Trong bài báo này, rất nhiều các kỹ thuật mới được giới thiệu. Trong đó, hai đóng góp nổi bật nhất chính là hàm ReLU và dropout. Hàm ReLU với cách tính đạo hàm đơn giản giúp tốc độ huấn luyện tăng lên đáng kể. Ngoài ra, việc ReLU không bị chặn trên bởi 1 khiến cho vấn đề vanishing gradient cũng được giải quyết phần nào. Dropout cũng là một kỹ thuật đơn giản và cực kì hiệu quả, giúp tránh được overfitting và giảm được số lượng phép toán khi huấn luyện.

Một trong những yếu tố quan trọng nhất giúp AlexNet thành công là việc sử dụng 2 GPU (card đồ họa) để huấn luyện mô hình. GPU được tạo ra với khả năng chạy song song nhiều lõi, đã trợ thành một công cụ cực kỳ phù hợp với các thuật toán deep learning, giúp tăng tốc thuật toán lên nhiều lần so với CPU.

AlexNet được ví như là cuộc cách mạng của Deep Convolutional Neural Network và được coi là nền tảng của các mô hình mạnh mẽ sau này (VGG, GoogleNet, ResNet). Từ năm 2012, rất nhiều các ứng dụng công nghệ đột phá đã được áp dụng vào cuộc sống hàng ngày. Cũng từ năm đó, số lượng các bài báo khoa học về deep learning tăng lên theo hàm số mũ. Các blog về deep learning cũng tăng lên từng ngày.

Công cuộc cải tiến:

Nhược điểm, cũng là ưu điểm của AlexNet chính là tỉ lệ hãm lỗi – 16%, mặc dù kết quả này là top1 năm đó nhưng vẫn chưa đạt đến ngưỡng nhận biết của con người. Và qua các năm sau năm 2012, vẫn tại ILSVRC nhiều mô hình được phát triển tới đỉnh cao và vượt qua cả khả năng nhận biết của con người. Hiện tại mô hình ResNet đang đứng vị trí số 1 với error rate 3.57%.

Về phần cứng, tại thời điểm năm 2012 thì việc huấn luyện mô hình với lượng tham số và neural lớn như vậy là một vấn đề cực kỳ khó khăn. Hiện nay, với sự tiến bộ vượt bậc của GPU, chung ta có nhiều kiến trúc CNN có cấu trúc phức tạp hơn, và hoạt động rất hiệu quả trên tập dữ liệu phức tạp.

Một số điểm cải tiến sau này:

- Mặc dù AlexNet đã chứng minh rằng các mạng neural tích chập có thể đạt được kết quả tốt, nhưng nó lại không cung cấp một khuôn mẫu chung để định hướng nghiên cứu sau này trong việc thiết kế các mạng mới. Giải pháp là sử dụng các khối tích chập chứa các tầng lặp lại theo khuôn mẫu (vd: VGG).
- Không có các kernel với đa dạng kích thước và sử dụng các nhánh song song, điều này về sau đã được áp dụng vào mạng nối song song GoogleNet.
- Và còn các kỹ thuật mà sau này mới được tìm ra và phổ biến như: batch normalization,…

TÀI LIỆU THAM KHẢO

[1] ImageNet Classification with Deep Convolutional Neural Networks (neurips.cc) URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[2] Các hàm kích hoạt (activation function) trong neural network (vietanh.dev).

URL: https://vietanh.dev/blog/2019-09-23-cac-ham-kich-hoat-activation-function-trong-neural-networks

[3] Machine Learning cơ bản (machinelearningcoban.com)

URL: https://machinelearningcoban.com/2018/06/22/deeplearning/

[4] Bài số 1 – số 6 | Deep Learning cơ bản (nttuan8.com)

URL: https://nttuan8.com/gioi-thieu-ve-deep-learning/

[5] 7.1. Mạng Nơ-ron Tích chập Sâu (AlexNet) — Đắm mình vào Học Sâu 0.14.4 documentation (aivivn.com).

URL: https://d2l.aivivn.com/chapter_convolutional-modern/alexnet_vn.html?

[6] Tìm hiểu về mạng neural network AlexNet - Phạm Duy Tùng Machine Learning Blog (phamduytung.com).

URL: https://www.phamduytung.com/blog/2018-06-15-understanding-alexnet/

---------- Hết ----------