Bảng 1: Ký Hiệu

Ký hiệu	Ý nghĩa
a	Một số thực
$\mathbf{a}$	Một vector
$\mathbf{A}$	Một ma trận
a_i	Phần tử thứ i của vector $\mathbf{a}$
a_{ij}	Phần tử ở dòng i cột j của ma trận trận $\mathbf{A}$
A_i	Dòng thứ i của ma trận $\mathbf{A}$
$\mathbf{A}^T$	Ma trận chuyển vị của ma trận $\mathbf{A}$
$\mathbb{K}$	Tập K
$f(x)$	Hàm số f với biến là x
$f'(x)$	Đạo hàm của hàm f(x)
f'_x hoặc $\frac{\partial f}{\partial x}$	Đạo hàm riêng của hàm f với x
$\nabla_{\mathbf{x}} f$	Gradient của hàm f theo vector $\mathbf{x}$

Bảng 2: Ký hiệu riêng tại phần mạng nơron

Ký hiệu	Ý nghĩa
$\mathbf{X}$	Tập dữ liệu (quan sát) đầu vào
$\mathbf{x}_{(i)}$	Dữ liệu thứ i của đầu vào
$\mathbf{y}$	Vector label
y_i	Label của dữ liệu thứ i
$\mathbf{w}$	Vector trọng số (weights)
$\mathbf{W}$	Ma trận trọng số
L	Số lượng layer
l_i	Layer thứ i
J	Hàm mất mát
$\widehat{\mathbf{Y}}$	Đầu ra dự đoán cho tập dữ liệu $\mathbf{X}$
$\widehat{\mathbf{y}}_i$	Đầu ra dự đoán cho dữ liệu $\mathbf{x}_i$
ReLu	Rectified linear unit
CNNs	Convolutional neural networks
Conv	Convolutional layer
FC	Full-connected layer

Chương 1

Nhắc lại kiến thức

Trong chương này, tôi xin trình bày về các kiến thức cơ bản cần có để có thể thực hiện được bài toán đề ra.

1.1 Ma trận

1.1.1 Định nghĩa

Một ma trận **A** loại (cấp) m_xn trên trường $\mathbb{K}$ ($\mathbb{K}$ – là trường thực $\mathbb{R}$ hoặc phức $\mathbb{C}$) là một bảng chữ nhật gồm mxn phần tử trong $\mathbb{K}$ được viết thành m dòng và n cột như sau:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix}$$

Trong đó:

a_{ij} là phần tử của ma trận A nằm ở giao điểm của dòng i và cột j

m : số dòng của ma trận **A**

n: số cột của ma trận **A**

$\begin{bmatrix} a_{i1} & a_{i2} & a_{i3} & \dots & a_{in} \end{bmatrix}$: dòng thứ i của ma trận **A**

$\begin{bmatrix} a_{j1} \\ a_{j2} \\ a_{j3} \\ \vdots \\ a_{jm} \end{bmatrix}$: cột thứ j của ma trận **A**

1.1.2 Phép cộng ma trận

Cho hai ma trận **A**, **B** cùng cỡ m_xn, ta có tổng $\mathbf{A} + \mathbf{B}$ là ma trận có cùng kích thước (m_xn) với phần tử trong vị trí tương ứng bằng tổng của hai phần tử tương ứng của mỗi ma trận:

$$(\mathbf{A}+\mathbf{B}) = a_{ij} + b_{ij} \text{ với } 1 \leq i \leq m \text{ và } 1 \leq j \leq n$$

Ví dụ:

$$\begin{bmatrix} 1 & 2 \\ 6 & 3 \end{bmatrix} + \begin{bmatrix} 3 & -2 \\ -4 & 1 \end{bmatrix} = \begin{bmatrix} 1+3 & 2+-2 \\ 6+-4 & 3+1 \end{bmatrix} = \begin{bmatrix} 4 & 0 \\ 2 & 4 \end{bmatrix}$$

1.1.3 Phép nhân ma trận với ma trận

Xét ma trận $\mathbf{A}_{mxp}$ và ma trận $\mathbf{B}_{pxn}$, trong đó số cột của ma trận **A** bằng số hàng của ma trận **B**. Tích **AB** là ma trận **C** có m hàng và n cột, phần tử c_{ij} được xác định theo tích vô hướng của hàng tương ứng trong **A** với cột tương ứng trong **B**:

$$c_{ij} = a_{i1}b_{j1} + a_{i2}b_{j2} + \ldots + a_{ip}b_{jp} = \sum_{k=1}^{p}(a_{ik}b_{jk})$$

Ví dụ:

$$\begin{bmatrix} 1 & 2 \\ 6 & 3 \end{bmatrix} + \begin{bmatrix} 3 & -2 \\ -4 & 1 \end{bmatrix} = \begin{bmatrix} 1*3+2*-4 & 1*-2+2*1 \\ 6*3+3*-4 & 6*-2+3*1 \end{bmatrix} = \begin{bmatrix} -5 & 0 \\ 6 & -9 \end{bmatrix}$$

Ngoài ra có một phép nhân khác được gọi là *Hadamard* (hay *element-wise*) được sử dụng khá nhiều trong học máy. Tích Hadamard của hai ma trận **cùng kích thước A, B** được kí hiệu là $\mathbf{C} = \mathbf{A} \odot \mathbf{B}$, trong đó:

$$c_{ij} = a_{ij}b_{ij}$$

1.1.4 Ma trận chuyển vị

Ma trận chuyển vị là một ma trận ở đó các hàng được thay thế bằng các cột, và ngược lại hay nói cách khác nếu ma trận **B** là ma trận chuyển vị của ma trận **A** thì: $b_{ij} = a_{ji}$

Ma trận chuyển vị của ma trận **A** được ký hiệu là $\mathbf{A}^T$.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^T = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$$

1.2 Đạo hàm

1.2.1 Định nghĩa

Cho hàm số $y = f(x)$ xác định trên khoảng $(a;b)$ (khoảng $(a;b) = \{x \in \mathbb{R} | a < x < b\}$. Xét giá trị x_0 và giá trị $x \in (a;b), x \neq x_0$.

Đặt $\Delta x = x - x_0$ thì $x = x_0 + \Delta x$ và Δx được gọi là số gia đối số.

Đặt $\Delta y = f(x) - f(x_0)$ và Δy được gọi là số gia hàm số.

Xét tỷ số $\dfrac{\Delta y}{\Delta x}$. Nếu khi $\Delta x \to 0$, tỷ số đó dần tới một giới hạn thì giới hạn đó được gọi là đạo hàm của hàm số $y = f(x)$ tại điểm x_0 ký hiệu là $f'(x)$

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$$

1.2.2 Đạo hàm riêng

Đạo hàm riêng của một hàm số đa biến là đạo hàm theo một biến, các biến khác được xem như là hằng số.

Đạo hàm riêng của f đối với biến x được ký hiệu khác nhau bởi : f'_x, $\dfrac{\partial f}{\partial x}$.

Ví dụ: Hàm số $f(x, y) = ax^2 + bxy + cy^5$ thì ta có:

- Đạo hàm theo x: $f'_x = 2ax + by$
- Đạo hàm theo y: $f'_y = bx + 5cy^4$

Vector gradient: Cho một hàm số $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$. Trong trường hợp này f có các đạo hàm riêng $\dfrac{\partial f}{\partial x_j}$ đối với mỗi biến x_j $(1 \leq j \leq n)$ thì vector chứa các đạo hàm riêng này là vector gradient.

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \begin{bmatrix} \dfrac{\partial f(\mathbf{x})}{\partial x_1} \\ \dfrac{\partial f(\mathbf{x})}{\partial x_2} \\ \vdots \\ \dfrac{\partial f(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

1.2.3 Đạo hàm hàm hợp (Chain rule)

Đạo hàm hàm hợp là công thức để tính đạo hàm của hàm số gồm nhiều hàm số kết hợp với nhau. Đó là, nếu f, g là hai hàm số và hàm $h(x) = f(g(x))$ thì ta có

$$h'(x) = f(g(x))' = f'(g(x)).g'(x)$$

hay chúng ta có công thức quen thuộc hơn với cách đặt $z = f(y), y = g(x)$:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dz} = f'(y)g'(x) = f'(g(x)).g'(x)$$

Ví dụ: Tính đạo hàm của hàm số $h = (x^2 + 5)^2$
Đặt $g(x) = x^2 + 5$ thì ta có $f(g) = g^2$, do đó $h'(x) = f'(g(x)).g'(x) = 2g2x = 4(x^2 + 5)x$

Đạo hàm riêng của hàm hợp: Giả sử ta có hàm số $f(u, v)$ trong đó $u(x, y)$, $v(x, y)$ là các hàm số theo biến x, y và f có đạo hàm riêng theo u, v và u, v có đạo hàm riêng theo x, y thì ta có đạo hàm riêng của f theo x, y như sau:

$$\begin{cases} f'_x = f'_u u'_x + f'_v v'_x \\ f'_y = f'_u u'_y + f'_v v'_y \end{cases}$$

một cách biểu diễn khác

$$\begin{cases} \dfrac{\partial f}{\partial x} = \dfrac{\partial f}{\partial u}\dfrac{\partial u}{\partial x} + \dfrac{\partial f}{\partial v}\dfrac{\partial v}{\partial x} \\ \dfrac{\partial f}{\partial y} = \dfrac{\partial f}{\partial u}\dfrac{\partial u}{\partial y} + \dfrac{\partial f}{\partial v}\dfrac{\partial v}{\partial y} \end{cases} \tag{1.1}$$

1.3 Ước lượng tham số bằng cực đại khả dĩ

Ước lượng hợp lý cực đại (có người gọi là khả năng cực đại, tiếng Anh thường được viết là MLE, gọi tắt từ Maximum-Likelihood Estimation) là một kỹ thuật trong thống kê dùng để ước lượng giá trị tham số của một mô hình xác suất dựa trên những dữ liệu có được. Phương pháp này được đĩnh nghĩa như sau:

Giả sử $X = x_1, x_2, ..., x_n$ là tập n quan sát và $Y = y_1, y_2, ..., y_n$ là số nhãn của quan sát; x, y là hai biến độc lập ngẫu nhiên. Ta cần phải tìm tham số θ để biểu thức sau đây đạt được giá trị lớn nhất

$$h_\theta = P(Y|X;\theta) \tag{1.2}$$

hay biểu thức 1.2 được viết lại như sau:

$$\widehat{\theta} = \arg\max_\theta P(Y|X;\theta) \tag{1.3}$$

Do các quan sát là biến độc lập ngẫu nhiên nên ta có thể viết lại thành:

$$P(Y|X;\theta) = \prod_{i=1}^{N} P(y_i|x_i,\ \theta) \tag{1.4}$$

Nhưng trực tiếp hàm số trên không hề đơn giản, hơn nữa khi N lớn thì tích của N số nhỏ hơn một có thể dẫn đến sai số trong tính toán. Một phương pháp thường được sử dụng đó là lấy logarit tự nhiên (cơ số e) của hàm khả dĩ ta được:

$$l(P(Y|X;\theta)) = log \prod_{i=1}^{N} P(y_i|x_i;\theta) = \sum_{i=1}^{N} logP(y_i|x_i;\theta) \tag{1.5}$$

1.4 Một vài phương pháp tối ưu

Mục tiêu của bài toán tối ưu là tìm ra nghiệm *global minimum*(điểm mà tại đó hàm số đạt giá trị nhỏ nhất) của hàm số. Tuy nhiên, việc tìm global minimum của các hàm số là rất phức tạp, thậm chí là bất khả thi. Thay vào đó, người ta thường cố gắng tìm các điểm *local minimum* (điểm cực tiểu), và ở một mức độ nào đó, coi đó là nghiệm cần tìm của bài toán.

Giả sử ta có N quan sát $(\mathbf{X}, \mathbf{y})$ có ánh xạ $\mathbf{X} \rightarrow \mathbf{Y}$ với $f(\theta, \mathbf{X}) = \mathbf{y}$. Ta cần tìm global minimum cho hàm $f(\theta; \mathbf{X}, \mathbf{y})$ trong đó θ là một vector, $\theta = [\theta_1, \theta_2, ..., \theta_m]$. Đạo hàm của hàm số đó tại một điểm θ bất kỳ được ký hiệu là $\nabla_\theta f(\theta; \mathbf{x}, y)$

1.4.1 Gradient descent

Thuật toán gradient descent giúp ta tìm θ sao cho $f(\theta, \mathbf{X})$ gần $\mathbf{y}$ nhất.

Algorithm 1 Gradient desent

Require: Tập N quan sát ($\mathbf{X}$,$\mathbf{y}$)
Require: $\theta = [\theta_1, \theta_2, ..., \theta_m]$
repeat
  for all $i = 1; i \leq N; i++$ **do**
    $\nabla\theta_i := -\nabla_\theta f(\theta; \mathbf{x}_i, y_i)$
  end for
  $\nabla\theta = \frac{1}{N}\sum_{i=1}^{N} \nabla\theta_i$
  Chọn bước nhảy η
  Cập nhật $\theta := \theta - \eta\nabla\theta$
until thuật toán hội tụ

Trong đó η (đọc là eta) là một số dương được gọi là learning rate (tốc độ học) và giá trị của learning rate thường là 0.001. Việc lựa chọn learning rate rất quan trọng trong các bài toán thực tế. Việc lựa chọn giá trị này phụ thuộc nhiều vào từng bài toán và phải làm một vài thí nghiệm để chọn ra giá trị tốt nhất. Và dấu trừ tại $\nabla\theta := -\nabla_\theta f(\theta)$ thể hiện việc chúng ta phải đi *ngược với đạo hàm* (Đây cũng chính là lý do phương pháp này được gọi là Gradient Descent - descent nghĩa là đi ngược).

Nếu dữ liệu có kích thước N lớn thì mỗi lần cập nhật θ đòi hỏi chúng ta sử dụng tất cả các quan sát $\mathbf{x}_i$ do đó khối lượng tính toán lớn do phải tính đạo hàm trên toàn bộ dữ liệu, thuật toán chạy chậm. Do vậy để tiết kiệm khối lượng tính toán, chúng ta sẽ cập nhật tính toán sau mỗi dữ liệu quan sát, phương pháp này gọi là *sochastic gradient descent (SGD)*

Algorithm 2 Stochastic Gradient descent

Require: Tập N quan sát ($\mathbf{X}$,$\mathbf{y}$)
Require: $\theta = [\theta_1, \theta_2, ..., \theta_m]$
repeat
  Xáo trộn dữ liệu
  for all $i = 1; i \leq N; i++$ **do**
    $\nabla\theta_i := -\nabla_\theta f(\theta; \mathbf{x}_i, y_i)$
    Chọn bước nhảy η
    Cập nhật $\theta := \theta + \eta\nabla\theta$
    if hội tụ **then**
      break
    end if
  end for
until thuật toán hội tụ

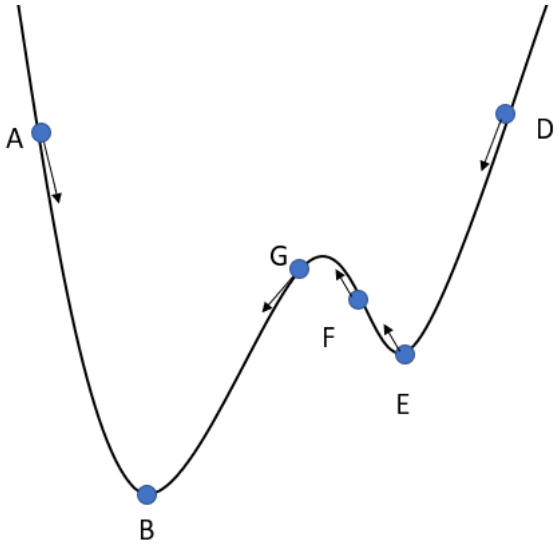
Khác với SGD, thay vì mỗi *iteration* ta tính toán trên một quan sát thì ta sẽ tính toán với k quan sát ($1 < k \ll N$). Phương pháp này được gọi là *mini-batch gradient descent.*

Algorithm 3 Mini-batch Gradient descent

Require: Tập n quan sát (**X**,**y**)
Require: $\theta = [\theta_1, \theta_2, ..., \theta_m]$
repeat
  Xáo trộn dữ liệu
  for all $i = 1; i \leq N; i = i + k$ **do**
    $\nabla\theta_i := -\sum_{j=i}^{i+k} \nabla_\theta f(\theta; \mathbf{x}_j, y_j)$
    Chọn bước nhảy η
    Cập nhật $\theta := \theta - \eta\nabla\theta_i$
    if hội tụ **then**
      break
    end if
  end for
until thuật toán hội tụ

1.4.2 Gradient descents với Momentum

Giả sử ta vẽ được một thung lũng như Hình **??**, ta thả một viên bi vào trong đó và mong muốn nó lăn đến điểm B, điểm sâu nhất của thung lũng. Nếu ta may mắn thả viên bi ở điểm A hoặc điểm G thì viên bi dễ dàng đến tiến điểm B. Nhưng nếu ta thả viên vi tại điểm D thì viên bi có thể sẽ giao động xung quanh điểm E và dừng tại đó do chưa đủ lực để đẩy viên bi qua điểm F rồi đến điểm G. Khi đó E chính là một điểm local minimum mà chúng ta không muốn. Do đó nếu ta tác động thêm một lực vào viên bi giúp nó có thể từ điểm E vượt qua F và tiến đến G.



Hình 1.1: Đồ thị

Thuật toán gradient descent được ví như trọng lực tác dụng vào viên bi giúp nó di chuyển, B được coi điểm global minimum và E là một điểm local minimun. Để tránh hiện tượng nghiệm của gradient descent rơi vào một điểm local minimum không mong muốn thì ta tác động thêm một lực giúp gradient descent có thể bật ra khỏi vị trí của local minimum, lực này gọi là đà (*momentum*). Vì thế cách cập nhật θ sẽ thay được thay đổi một chút như sau:

$$v_t := \beta v_{t-1} + \eta \nabla \theta_t$$

$$\theta_t := \theta_{t-1} - v_t$$

Với t là bước lặp thứ t, biến v được khởi tạo bằng 0 và trong tối ưu β được gọi là đà (*momentum*) với giá trị thường là 0.9.

1.4.3 RMSprop

RMSprop là một cách giúp cho learning rate có thể thích nghi, điều chỉnh dựa trên độ lớn của gradient, được đưa ra bởi Geoff Hinton. Thuật toán này là một cách khắc phục việc dừng huấn luyện quá sớm khi áp dụng thuật toán Adagrad bằng cách chia learning rate cho . Khi áp dụng RMSprop, learning rate sẽ được thay đổi như sau:

$$m_t := \beta m_{t-1} + (1 - \beta) \nabla \theta_t^2$$

$$\theta_t := \theta_{t-1} - \frac{\eta}{\sqrt{m_{t-1}} + \epsilon} \nabla \theta_t$$

Ta có biến m được khởi tạo bằng 0 và β là tốc độ giảm của learning rate, thường có giá trị là 0.9, 0.95 hoặc 0.99. Và ϵ giúp chúng ta tránh trường hợp chia cho 0, vì thế giá trị của ϵ thường là 10^{-8}

1.4.4 Adam

Adam (Adaptive Moment Estimation) là một bản cập nhật được đưa ra gần đây, nó khá giống với RMSprop và momentum. Cách cập nhật của thuật toán Adam như sau:

$$m_t := \beta_1 m_{t-1} + (1 - \beta_1) \nabla \theta_t$$

$$v_t := \beta_2 v_{t-1} + (1 - \beta_2) \nabla \theta_t^2$$

$$\widehat{m_t} := \frac{m_t}{1 - \beta_1^t}$$

$$\widehat{v_t} := \frac{v_t}{1 - \beta_1^t}$$

$$\theta_t := \theta_1 + \frac{\eta}{\sqrt{\widehat{v_t}} + \epsilon} \widehat{m_t}$$

$m_t,\ v_t$ là giá trị ước lượng giữa thời điểm trước và thời điểm sau của các gradient tương ứng. Các giá trị của $m_t,\ v_t$ đều được khởi tạo bằng 0. Các tác giả của Adam thấy rằng chúng bị lệch về 0, đặc biệt là các bước lặp đầu. Do vậy họ tạo ra $\widehat{m_t}$ và $\widehat{v_t}$ để chống lại việc giá trị của m_t và v_t lệch về 0. Các giá trị của $\beta_1,\ \beta_2$ được tác giả đề xuất là 0.9 cho β_1, 0.999 cho $\beta 2$ và 10^{-8} cho ϵ như ở RMSprop.

Chương 2

Cơ Sở Lý Thuyết

Trong chương này tôi sẽ trình bày về một số lý thuyết cơ bản để tiếp cận với mạng nerual dễ dàng hơn.

2.1 Các khái niệm cơ bản

- **Quan sát**: kí hiệu là $\mathbf{x}$, input trong các bài toán. Quan sát thường có dạng là một vector $\mathbf{x} = (x_1, x_2, x_3, \ldots, x_n)$, gọi là **feature vector**. Mỗi x_i gọi là một feature. Ví dụ bạn muốn đoán xem hôm nay có mưa không dựa vào observation gồm các feature (nhiệt độ, độ ẩm, tốc độ gió).

- **Label**: kí hiệu là y, output của bài toán. Mỗi quan sát sẽ có một label tương ứng. Ở ví dụ về mưa ở trên label chỉ là "mưa" hoặc "không mưa"; hay về điểm thì là các số thực từ 0 đến 10. Label có thể mang nhiều dạng nhưng đều có thể chuyển đổi thành một số thực hoặc một vector.

- **Model**: trong chương này các bạn hiểu là nó là một hàm số $f(x)$, nhận vào một đầu vào $\mathbf{x}$ và trả về một đầu ra dự đoán (predict) $y = f(\mathbf{x})$.

- **Parameter**: mọi thứ của model được sử dụng để tính toán ra output. Ví dụ model là một hàm đa thức bậc hai: $f(x) = ax_1^2 + bx_2 + c$ thì parameter của nó là bộ ba (a, b, c). Để ngắn gọn, người ta thường gom tất cả parameter của một model lại thành một vector, thường được kí hiệu là $\mathbf{w}$ và biểu diễn thông qua hàm $f(\mathbf{x}, \mathbf{w}) = \mathbf{xw}$.

2.2 Huấn luyện mạng

Một mạng nơron được huyấn luyện sao cho với một tập các vector đầu vào X, mạng có khả năng tạo ra tập các vector đầu ra mong muốn Y của nó. Tập $\mathbf{X}$ được sử dụng cho huấn luyện mạng được gọi là tập huấn luyện (training set). Các phần tử $\mathbf{x}$ thuộc $\mathbf{X}$ được gọi là các mẫu huấn luyện (training example). Quá trình huấn luyện bản chất là sự thay đổi các trọng số liên kết của mạng. Trong quá trình này, các trọng số của mạng sẽ hội tụ dần tới các giá trị sao cho với mỗi vector đầu vào $\mathbf{x}$ từ tập huấn luyện, mạng sẽ cho ra vector đầu ra y như mong muốn.

Có hai phương pháp học phổ biến là học có giám sát (supervised learning), học không giám sát (unsupervised learning):

- **Học có giám sát**: Là quá trình học có sự tham gia giám sát của một "thầy giáo". Giống như ta dạy trẻ nhận diện các loại phương tiện. Ta đưa ra hình ô tô và bảo với trẻ đó rằng đây là chiếc ô tô. Việc này được thực hiện trên các loại phương tiện khác nhau như xe

máy, máy bay, xe đạp.... Sau đó khi kiểm tra ta sẽ đưa ra một hình phương tiện bất kì, các hình này hơi khác so với các hình đã dạy trẻ, và cho trẻ đoán xem xe này thuộc loại phương tiện nào?
Như vậy với học có giám sát, số lớp cần phân loại đã được biết trước. Nhiệm vụ của thuật toán là phải xác định được một cách thức phân lớp sao cho với mỗi vector đầu vào sẽ được phân loại chính xác vào lớp của nó

- **Học không giám sát**: Là việc học không cần có bất kỳ một sự giám sát nào.Trong bài toán học không giám sát, chúng ta không biết câu trả lời chính xác cho mỗi dữ liệu đầu vào. Nhiệm vụ của thuật toán là phải phân chia tập dữ liệu đầu vào thành các nhóm con, mỗi nhóm chứa các đặc trưng giống nhau. Ví dụ như phân nhóm loại khách hàng dựa trên hành vi mua hàng: số lượng hàng hóa mua, loại hàng hóa mua, khoảng thời gian cách nhau giữa mỗi lần mua,....
Như vậy với học không giám sát, số lớp phân loại chưa được biết trước, và tùy theo tiêu chuẩn đánh giá độ tương tự giữa các mẫu mà ta có thể có các lớp phân loại khác nhau.

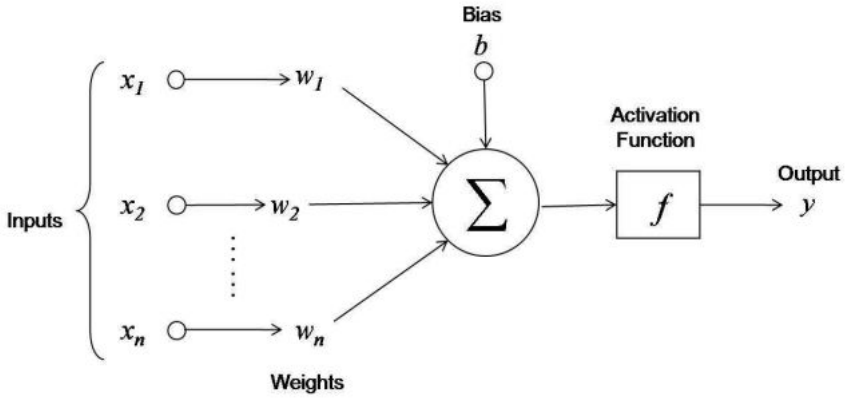
Tài liệu này tôi sẽ trình bày về phần học có giám sát thông qua mạng nơ-ron nhân tạo và mạng tích chập.

2.3 Định nghĩa

Mạng nơron nhân tạo, Artificial Neural Network (ANN) là một mô hình xử lý thông tin phỏng theo cách thức xử lý thông tin của các hệ nơ-ron sinh học. Nó được tạo nên từ một số lượng lớn các phần tử (nơ-ron) kết nối với nhau thông qua các liên kết (trọng số liên kết) làm việc như một thể thống nhất để giải quyết một vấn đề cụ thể nào đó. Một mạng nơ-ron nhân tạo được cấu hình cho một ứng dụng cụ thể (nhận dạng mẫu, phân loại dữ liệu,...) thông qua một quá trình học từ tập các mẫu huấn luyện. Về bản chất học chính là quá trình hiệu chỉnh trọng số liên kết giữa các nơ-ron.

2.4 Cấu tạo của một nơ-ron

Trong phần này tôi sẽ trình bày chi tiết cấu trúc của một nơ-ron 2.1 trong mạng lưới nơ-ron.



Hình 2.1: Cấu trúc của một nơ-ron

Các thành phần cơ bản của nơ-ron:

1. **Đầu vào** (**x**): các tín hiệu vào của nơ-ron, các tín hiệu này thường đưa vào dưới dạng một vector N chiều và được kí hiệu là **x** và mỗi phần tử trong vector được kí hiệu là $x_i, i \in 0, n$

2. **Trọng số** (**w**): mỗi liên kết được thể hiện bởi một trọng số liên kết và được gọi là weight. Trọng số liên kết giữa tín hiệu vào thứ j với nơ-ron k thường được kí hiệu là w_{kj}, và. Thông thường, các trọng số này được khởi tạo một cách ngẫu nhiên theo phân phối chuẩn ở thời điểm khởi tạo mạng và được cập nhật liên tục trong quá trình học mạng.

3. **Bias** (b): là tham số nhằm tăng khả năng thích ứng của mạng nơron trong quá trình học. Bias gần giống như trọng số, trừ một điều là nó luôn có tín hiệu vào không đổi bằng 1 (được trình bày rõ hơn ở phần 2.7.2). Tham số này có thể bỏ đi nếu không cần thiết.

4. **Hàm kết hợp** (z): Mỗi một đơn vị trong một mạng kết hợp các giá trị đưa vào nó thông qua các liên kết với các đơn vị khác, sinh ra một giá trị gọi là net input. Hàm thực hiện nhiệm vụ này gọi là hàm kết hợp (combination function), được định nghĩa bởi một luật lan truyền cụ thể. Thông thường hàm này sẽ là hàm tổng của tích các trọng số với đầu vào và độ lệch. Và được biể diễn thông qua biểu thức $z = \sum_{i=1}^{n}(x_i w_i) + b$.

5. **Hàm truyền (activiation function hoặc transfer function)**: Hàm này được dùng để giới hạn phạm vi đầu ra của mỗi nơ-ron và nhận đầu vào là kết quả của hàm kết hợp. Tôi sẽ trình bày phần này rõ hơn ở phần 2.6.

6. **Đầu ra (output)**: Là tín hiệu đầu ra của một nơ-ron, với mỗi nơ-ron sẽ có tối đa là một đầu ra. Nếu như nơ-ron đó ở các hidden layer thì đầu ra của nó được gọi là activation và được kí hiệu là a.

Khi quy về toán học thì một nơ-ron sẽ được thể hiện thông qua hai hàm sau:

$$z = \sum_{i=1}^{n}(x_i w_i) + b = \mathbf{xw} + b$$

$$a = f(z)$$

Trong đó:

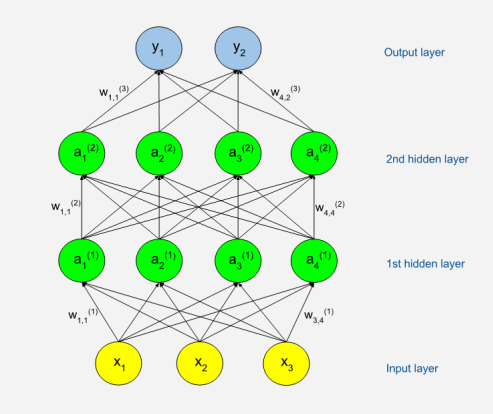
- x_i, w_i là giá trị thứ i của đầu vào và trọng số tương ứng
- Hàm f là hàm truyền và có đầu vào là giá trị của bộ tổng z
- a là giá trị được tính bởi hàm truyền và là đầu ra của nơ-ron

Như vậy một nơ-ron nhận các đầu vào sau đó kết hợp với các trọng số rồi đưa kết quả vào hàm truyền và cho ra một giá trị đầu ra.

2.5 Cấu trúc của mạng nơ-ron

2.5.1 Layer

Mạng neural được chia làm 3 lớp chính: lớp đầu vào (*input layer*, màu vàng), lớp ẩn (*hidden layer*, màu xanh lá), lớp đầu ra (*output layer*, màu xanh dương) và trong mỗi lớp có số lượng *unit* khác nhau. Cấu trúc mạng được minh họa trong Hình 2.2.



Hình 2.2: Cấu trúc mạng nơ-ron

- *Input layer*: Biểu diễn tổng quát của mỗi quan sát
- *Output layer*: Thể hiện đầu ra dự đoán của model
- *Hidden layer*: Là lớp thể hiện cấu trúc của mạng nơ-ron. Các *hidden layer* theo thứ tự từ input layer đến output layer được đánh số thứ tự là *hidden layer 1, hidden layer 2,....*

Số lượng layer trong một mạng nơ-ron được ký hiệu là L và được tính bằng số hidden layer cộng thêm một. Ví dụ trong Hình 2.2, $L = 3$.

2.5.2 Units

Mỗi *node* hình trong trong Hình 2.2 được gọi là một *unit* hoặc một nơ-ron. Như tôi đã trình bày cấu trúc của một nơ-ron ở phần 2.4, đầu vào của một nơ-ron là một hàm kết hợp và đầu ra thông qua một hàm gọi là activiation. Ở mạng nơ-ron thì đầu vào của hidden layer thứ l được ký hiệu là $\mathbf{z}^{(l)}$ với $z_i^{(l)} = \mathbf{x}\mathbf{w}_i^{(l)}$ và đầu ra của mỗi unit được ký hiệu là $\mathbf{a}^{(l)}$ với $\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)})$.

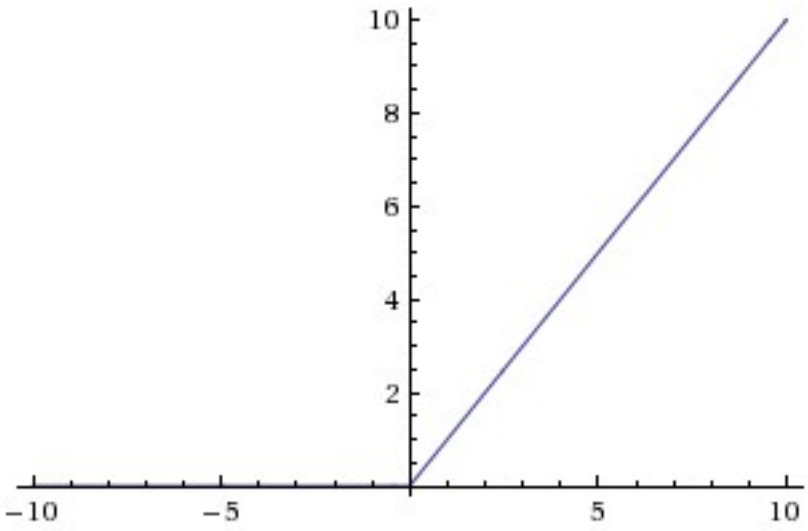
Số lượng unit trong mỗi lớp, số lượng lớp trong mỗi cấu trúc mạng là không xác định. Nó được xây dựng dựa vào kinh nghiệm của người thiết kế mạng hoặc theo các tài liệu đã được công bố. Nếu số lượng lớp quá lớn thì tốc độ tính toán chậm, còn số lượng lớp ít thì độ tin cậy của kết quả không cao. Nếu số node trong các lớp lớn thì sẽ bị overfitting, ngược lại sẽ bị underfitting. Tôi sẽ trình bày phần overfitting, underfitting ở phần sau.

2.6 Hàm truyền (activation function)

Phần lớn các đơn vị trong mạng nơron chuyển net input bằng cách sử dụng một hàm vô hướng (scalar-to-scalar function) gọi là hàm kích hoạt, kết quả của hàm này là một giá trị gọi là mức độ kích hoạt của đơn vị (unit's activation). Loại trừ khả năng đơn vị đó thuộc lớp ra, giá trị kích hoạt được đưa vào một hay nhiều đơn vị khác. Các hàm kích hoạt thường bị ép vào một khoảng giá trị xác định, do đó thường được gọi là các hàm bẹp (squashing). Một số hàm thường được sử dụng là ReLU, sigmoid, softmax.

2.6.1 Hàm ReLU

Hàm ReLU (Rectified Linear Unit) được sử dụng rộng rãi gần đây vì tính đơn giản của nó. Nó có công thức toán học $f(s) = \max(0, s)$. Đồ thị hàm ReLU được thể hiện ở Hình 2.3



Hình 2.3: Hàm ReLU

Từ biểu đồ ta có thể dễ dàng thấy độ lệch của hàm ReLU luôn bằng 1 khi $x > 0$, bằng 0 khi $x < 0$ và ta quy ước tại $x = 0$ thì độ lệch bằng 0 nhưng trường hợp này rất ít khi xảy ra.

$$\frac{d}{dx}ReLU(x) = \begin{cases} 0, \text{ if } x < 0, \\ 1, otherwise. \end{cases}$$

Hàm ReLU thường được sử dụng làm hàm truyền trong các hidden layer, còn ở layer cuối cùng thì ta sẽ sử dụng hàm khác để có thể tính toán được sắc xuất dự đoán vào vùng phân loại.

2.6.2 Hàm sigmoid

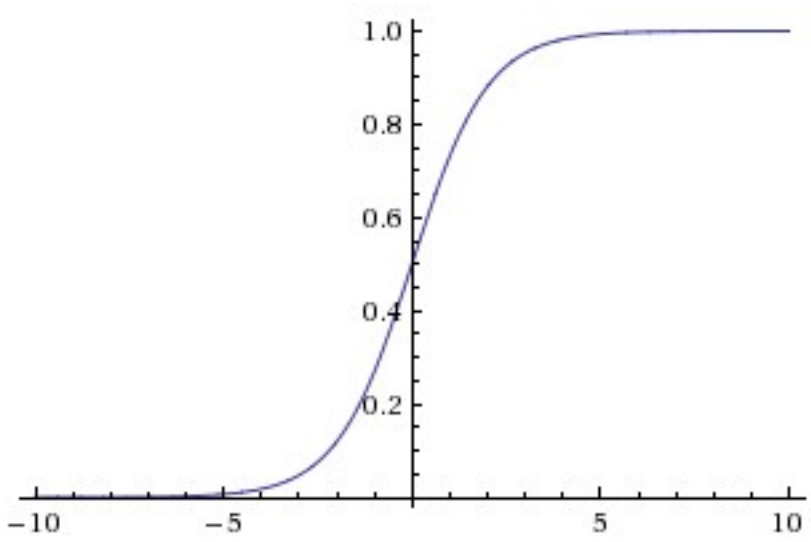
Hàm logistic là mô hình một dạng đường cong-S của sự tăng trưởng của một tập P nào đó. Sự tăng trưởng được mô hình như sau: giai đoạn tăng trưởng ban đầu được xấp xỉ hàm mũ và khi quá trình bão hòa bắt đầu, sự phát triển sẽ chậm lại, và tới giai đoạn trưởng thành thì dừng hẳn.

Hàm logistic có công thức toán học là:

$$P(x, a, m, n, r) = a\frac{1 + m\exp(-x/r)}{1 + n\exp(-x/r)} \qquad \text{với } a, m, n, r \text{ là các tham số thực.}$$

Hàm sigmoid là một trường hợp đặc biệt của hàm logistic và có công thức toán học là $\sigma(x) = \dfrac{1}{1 + \exp(-t)}$, đồ thị được thể hiện ở Hình 2.4

Nhìn vào đồ thị ta có thể thấy giá trị của hàm nằm trong khoảng (0,1]. Cụ thể, với đầu vào lớn hàm số sẽ cho đầu ra gần với 1 còn với đầu vào nhỏ hàm số sẽ cho đầu ra gần với 0. Hàm số này được sử dụng nhiều trong quá khứ vì có đạo hàm $\dfrac{d\sigma(x)}{d(x)} = \sigma(x).(1 - \sigma(x)$ rất đẹp. Những năm gần đây, hàm số này ít khi được sử dụng do khi có đầu vào là những số cực lớn hoặc cực bé thì đạo hàm của hàm số xấp xỉ bằng 0, điều này gây ảnh hưởng đến việc cập nhập các trọng số trong quá trình học và làm cho thời gian tính toán lâu hơn. Tôi sẽ trình bày tại sao lại như vậy ở phần backpropagation.



Hình 2.4: Hàm sigmoid

2.6.3 Hàm softmax

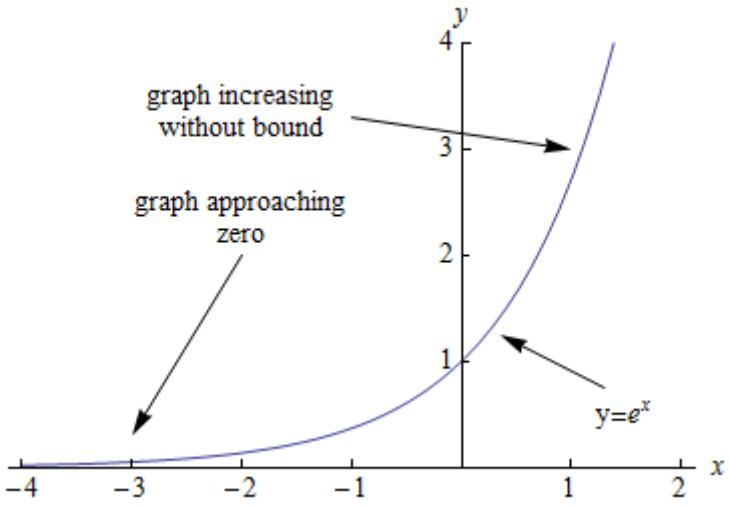
Hàm softmax hay hàm trung bình mũ là sự khái quát hóa của hàm logistic biến không gian K-chiều véc tơ với giá trị thực bất kỳ đến không gian K-chiều véc tơ mang giá trị trong phạm vi (0, 1).

Hàm softmax có phương trình toán học như sau:

$$y_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}; \forall i = 1, ..., n$$

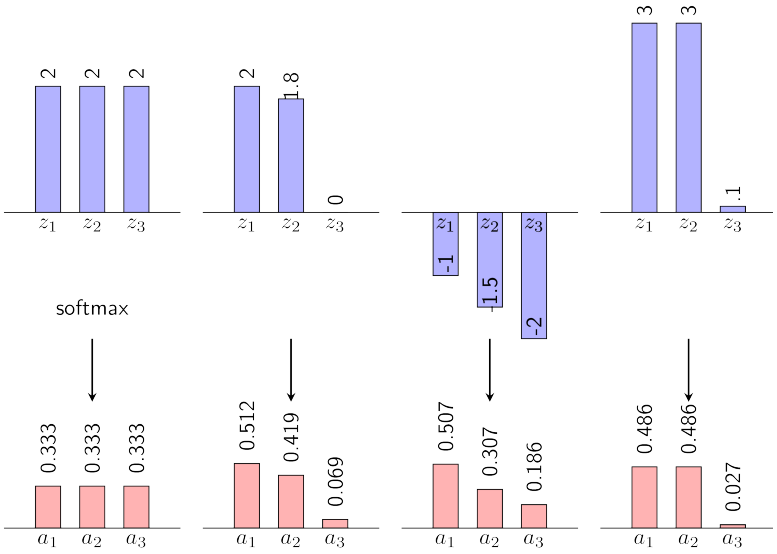
với n là số lượng phần tử trong vector $\mathbf{x}$.

Hiện nay hàm này được sử dụng rộng rãi trong việc phân loại trong lớp đầu ra. Vì nếu tại giá trị phần tử x_i lớn vượt trội so với toàn bộ dữ liệu ở vector $\mathbf{x}$ thì giá trị đầu ra y_i cũng sẽ lớn vượt trội so với đầu ra ở các phần tử khác. Điều tiếp theo chúng ta có thể thấy rằng tổng đầu ra sẽ luôn bằng môt, softmax đã sử dụng hàm $\exp(x) = e^x$ (Hình 2.5) giúp cho giá trị của đầu ra luôn dương và thứ tự các phần tử đầu ra tương ứng với thứ tự đầu vào.



Hình 2.5: Đồ thị hàm $f(x) = \exp(x)$

Một vài ví dụ về hàm softmax, với dữ liệu đầu vào là vector **z** và giá trị đầu ra là vector **a**, giá trị các vector được thể hiện qua Hình 2.6.



Hình 2.6: Ví dụ hàm softmax

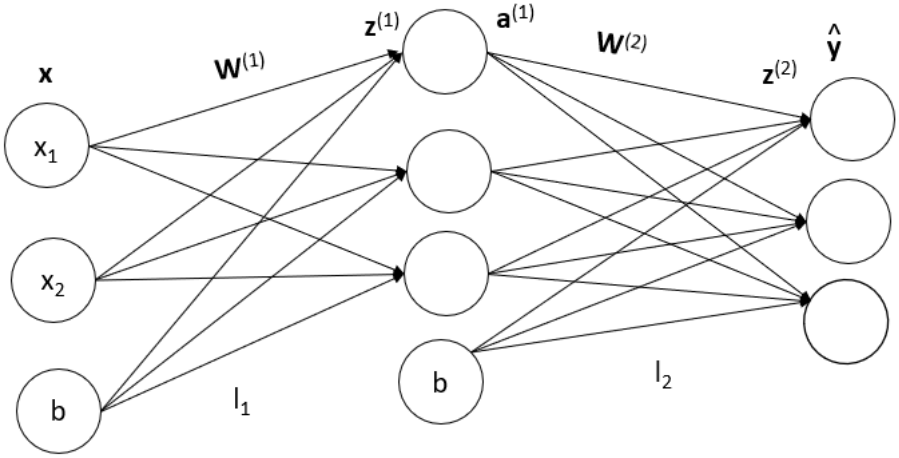
Tại đây chúng ta có thể thấy giá trị của đầu ra luôn dương, giá trị đầu ra lớn nhất tại giá trị ở phần tử đầu vào lớn nhất, tổng các giá trị đầu ra luông bằng 1.

Như vậy với hàm softmax chúng ta có với mỗi đầu ra y_i luôn phụ thuộc vào tất cả x_i và đầu ra luôn dương, tổng đầu ra bằng một, thứ tự đầu vào tương ứng với đầu ra. Vì thế đầu ra y_i có thể coi là xác suất để đầu vào x_i rơi vào lớp thứ i.

2.7 Mạng lan truyền thẳng

2.7.1 Ví dụ

Giả sử ta có mạng neural như Hình 2.7 với tập dữ liệu đầu vào là ma trận $\mathbf{X}_{2\times N}$ (N là số lượng quan sát, 2 là số lượng feature), và label tương ứng là vector $\mathbf{y}_{N\times 1}$ (2 là số lượng nhãn), $\mathbf{y}_i \in (1, 2, 3, \ldots, C)$ với C là số lớp cần phân loại .



Hình 2.7: Mạng neural cho bài toán

Xét một cặp dữ liệu $(\mathbf{x}_i, \mathbf{y}_i)$ với hàm activation ở hidden layer là hàm relu, hàm activation ở output layer là hàm softmax. Mạng lan truyền thẳng sẽ được tính như sau:

$$\mathbf{x}_{1\times 2} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \quad \mathbf{W}^{(1)T}_{3\times 2} = \begin{bmatrix} w^{(1)}_{11} & w^{(1)}_{12} \\ w^{(1)}_{21} & w^{(1)}_{22} \\ w^{(1)}_{31} & w^{(1)}_{32} \end{bmatrix}^T \tag{2.1}$$

$$\begin{aligned} \mathbf{z}^{(1)}_{1\times 3} &= \mathbf{x}_{1\times 2}\mathbf{W}^{(1)T}_{3\times 2} + b^{(1)} \\ &= \begin{bmatrix} z^{(1)}_1 & z^{(1)}_2 & z^{(1)}_3 \end{bmatrix} \end{aligned} \tag{2.2}$$

$$\begin{aligned} \mathbf{a}^{(1)}_{1\times 3} &= \max(0, \mathbf{z}^{(1)}) \\ &= \begin{bmatrix} a^{(1)}_1 & a^{(1)}_2 & a^{(1)}_3 \end{bmatrix} \end{aligned} \tag{2.3}$$

$$\mathbf{W}^{(2)T}_{3\times 3} = \begin{bmatrix} w^{(2)}_{11} & w^{(2)}_{11} & w^{(2)}_{13} \\ w^{(2)}_{21} & w^{(2)}_{22} & w^{(2)}_{23} \\ w^{(2)}_{31} & w^{(2)}_{32} & w^{(2)}_{33} \end{bmatrix}^T \tag{2.4}$$

$$\begin{aligned} \mathbf{z}^{(2)}_{1\times 3} &= \mathbf{a}^{(1)}_{1\times 3}\mathbf{W}^{(2)T}_{3\times 3} + b^{(2)} \\ &= \begin{bmatrix} z^{(2)}_1 & z^{(2)}_2 & z^{(2)}_3 \end{bmatrix} \end{aligned} \tag{2.5}$$

$$\mathbf{a}^{(2)}_{1\times 3} = \frac{\exp(\mathbf{z}^{(2)})}{\sum_{j=1}^{2}\exp(\mathbf{z}^{(3)}_j)} \tag{2.6}$$

$$\widehat{\mathbf{y}} = \mathbf{a}^{(2)}_{1\times 3}$$

Tại các biểu thức số 2.1,2.4 là giá trị đầu vào và khởi tạo ma trận trọng số cho layer thứ 1,2; tại biểu thức số 2.2,2.5 ta tính giá trị hàm kết hợp (bộ tổng) làm input đầu vào cho nơ-ron; tại biểu thức số 2.3,2.6 ta tính đầu ra cho mỗi nơ-ron. Ở layer thứ nhất đầu ra nơ-ron sử dụng *hàm ReLU* 2.3 làm hàm activation vì thế tại biểu thứ số 2.3 ta có $\mathbf{a}^{(1)}_{1\times 3} = \max(0, \mathbf{z}^{(1)})$ còn tại layer thứ hai sử dụng *hàm softmax* 2.6.3 làm hàm đầu ra cho nơ-ron nên ta có $\mathbf{a}^{(2)}_{1\times 3} = \frac{\exp(\mathbf{z}^{(2)})}{\sum_{j=1}^{3}\exp(\mathbf{z}^{(2)}_j)}$.

Với tập dữ liệu $(\mathbf{X}, \mathbf{y})$ thì ta có đầu ra dự đoán như sau:

$$\widehat{\mathbf{Y}} = \frac{\exp(\mathbf{z}^{(2)}_i)}{\sum_{j=1}^{2}\exp(\mathbf{z}^{(2)}_{ij})}; \text{ với } i = 1, 2, ..., N$$

Việc tính toán tuần tự như vậy được được gọi là *feedforward*.

2.7.2 Thuật toán

Nếu ta thêm một phần tử vào $\mathbf{x}_i$ với giá trị bằng 1 và ta coi $b^{(l)}$ là một phần tử của ma trận trọng số $\mathbf{W}^{(l)}$ tương ứng với l là layer thứ l. Khi đó ta sẽ có công thức tổng quát hơn.

$$\mathbf{a}^{(0)} = \mathbf{x}_i = \begin{bmatrix} 1, x_1, x_2, \ldots, x_n \end{bmatrix}$$

$$\mathbf{w}^{(l)T}_j = \begin{bmatrix} w^l_{j0} := b^{(l)}, w^l_{j1}, w^l_{j2}, \ldots, w^l_{jn} \end{bmatrix}^T$$

$$z_j^{(l)} = \mathbf{a}^{(l-1)}\mathbf{w}_j^{(l)}$$
$$\mathbf{a}^{(l)} = \left[1, f(\mathbf{z}^{(l)})\right]^T$$

Algorithm 4 Forward propagation

Require: Network depth, L
Require: $\mathbf{W}^{(i)}, i \in \{1, \ldots, L\}$, ma trận trọng số của layer thứ i của model
Require: $\mathbf{X}, \mathbf{x}_i, i \in \{1, \ldots, N\}$, tập dữ liệu đầu vào với n là số lượng dữ liệu.

```
for j = 1, ..., N do
    a^(0) = x_j
    for i = 1, ..., L do
        z^i = a^(i-1) W^(i)T
        a^(i) = f(z^(i))
    end for
    ŷ = a^(L)
end for
```

2.7.3 Tối ưu hàm mất mát

Để giải quyết bài toán $\mathbf{W}=\arg\min_{\mathbf{W}} J$ chúng ta có một thuật toán là lan truyền ngược (back propagation) giúp ta điều chỉnh trọng số để giá trị của hàm mất mát giảm đi.

2.8 Hàm mất mát và Cross-entropy

2.8.1 One-hot encoding

One-hot encoding là biến mỗi nhãn thành một vector có kích thước bằng với tập nhãn và vị trí nhãn sẽ bằng 1 còn các vị trí còn lại sẽ bằng 0. One-hot encoding được mô tả ở Hình 2.8. Số lớp cần phân loại gồm: red, yellow, green, blue; các nhãn sẽ được biểu diễn lại như bảng bên; với cột là tập nhãn, dòng là dữ liệu được biểu diễn dưới dạng one-hot encoding.

	Các lớp : red, yellow, grenn, blue			
Nhãn	red	yellow	green	blue
red	1	0	0	0
green	0	0	1	0
red	1	0	0	0
yellow	0	1	0	0
blue	0	0	0	1
green	0	0	1	0
blue	0	0	0	1

Hình 2.8: Ví dụ one-hot encoding

2.8.2 Hàm mất mát

Hàm mất mát hay còn gọi là *loss function*, là sự chêch lệch, khác biệt giữa đầu ra dự đoán và đầu ra thực tế, có chức năng là đo độ chính xác của đầu ra dự đoán, thường được ký hiệu là J. Giả sử ta cần ánh xạ: $\mathbf{X} \rightarrow \mathbf{Y}$, trong đó $\mathbf{X}$ là tập các vấn đề và Y là tập các lời giải tương ứng cho vấn đề đó thì ta cần xây dựng. Ta cần xây dựng hàm sao cho $\widehat{\mathbf{Y}} = f(\mathbf{X}, \mathbf{W}) \approx \mathbf{Y}$. Như vậy hàm mất mát là độ chênh lệch, sự khác biệt giữa $\widehat{\mathbf{Y}}$ và $\mathbf{Y}$. Nếu như giá trị mất mát càng lớn thì điều đó có nghĩa rằng đầu ra dự đoán càng sai, các tham số truyền vào chưa chính xác vì thế chúng ta cần điều chỉnh tham số sao cho hàm $\widehat{\mathbf{Y}}$ càng gần $\mathbf{Y}$ càng tốt. Có nghĩa rằng chúng ta cần tìm: $\mathbf{W} = \arg\min_{\mathbf{W}} J$.

Trong nhiều bài toán, các mô hình tham số được định nghĩa như một phân phối $p(\mathbf{y}|\mathbf{x}, \mathbf{W})$ và sử dụng nguyên lý Maximum-Likelihood, có nghĩa là chúng ta sử dụng cross-entropy giữa $\widehat{\mathbf{y}}$ và $\mathbf{y}$ như là hàm mất mát.

2.8.3 Cross-entropy

Cross entropy là một đại lượng phản ánh mức độ khác biệt giữa hai phân phối xắc xuất. Sự khác biệt giữa phân bố p và q càng lớn, thì cross-entropy của p đối với q sẽ càng lớn hơn entropy của p.

Cross entropy giữa hai vector phân phối $\mathbf{p}$ và $\mathbf{q}$ được định nghĩa là:

$$H(\mathbf{p}, \mathbf{q}) = \mathbf{E}_{\mathbf{p}}[-\log(\mathbf{q})] \tag{2.7}$$

Với $\mathbf{p}, \mathbf{q}$ là rời rạc, công thức 2.7 được viết dưới dạng:

$$H(\mathbf{p}, \mathbf{q}) = -\sum_{i=1}^{C} p_i \log q_i \tag{2.8}$$

Chú ý: Hàm cross entropy không có tính đối xứng $H(\mathbf{p}, \mathbf{q}) \neq H(\mathbf{q}, \mathbf{p})$. Theo công thức 2.7 và 2.8 chúng ta có thể thấy giá trị của $\mathbf{q}$ không thể nhận giá trị là 0. Vì thế khi sử dụng cross entropy trong các bài toán học có giám sát, chúng ta phải để $\mathbf{p}$ là đầu ra thực tế vì chỉ có vị trí nhãn là được đánh dấu 1, các vị trí còn lại được đánh dấu 0 (sử dụng one-hot encoding), $\mathbf{q}$ là đầu ra dự đoán vì không có xắc suất nào bằng 0 tuyệt đối cả.

2.8.4 Xây dựng hàm mất mát

Giả sử số lớp chúng ta cần phân loại là C, xét một cặp dữ liệu $(\mathbf{x}_i, \mathbf{y}_i)$ với $\mathbf{y}_i$ là dạng one-hot tại đầu ra thực tế thứ i và đầu ra dự đoán $\widehat{\mathbf{y}}_i = softmax(\mathbf{a}_i^{(L)} \mathbf{W}_i^{(L)T})$. Như đã trình bày ở phần 2.6.3, giá trị $\widehat{\mathbf{y}_{ij}}$ *với j=1,...,C* thể hiện xắc suất để $\mathbf{x}_i$ rơi vào lớp thứ j. Chúng ta coi hai vector xắc suất $\widehat{\mathbf{y}}_i$, $\mathbf{y}_i$ lần lượt là $\mathbf{q}, \mathbf{p}$ ở trong biểu thức 2.8 do đó *giá trị mất mát* được tính như sau:

$$\begin{aligned}
J_i(\mathbf{W}) = J_i(\mathbf{W}; \mathbf{x}_i, \mathbf{y}_i) &= -\sum_{j=1}^{C} y_{ij} \log \widehat{y}_{ij} \\
&= -\sum_{j=1}^{C} y_{ij} \log \left(\frac{\exp(\mathbf{a}_i^{(L-1)} \mathbf{w}_{ij}^{(L)T})}{\sum_{k=1}^{C} \exp(\mathbf{a}_i^{(L-1)} \mathbf{w}_{ik}^{(L)T})} \right) \\
&= -\sum_{j=1}^{C} \left(y_{ji} \mathbf{a}_i^{(L-1)} \mathbf{w}_{ij}^{(L)T} \log \left(\sum_{k=1}^{C} \exp(\mathbf{a}_i^{(L-1)} \mathbf{w}_{ik}^{(L)T}) \right) \right) \\
&= -\sum_{j=1}^{C} y_{ji} \mathbf{a}_i^{(L-1)} \mathbf{w}_{ij}^{(L)T} + \log \left(\sum_{k=1}^{C} \exp(\mathbf{a}_i^{(L-1)} \mathbf{w}_{ik}^{(L)T}) \right)
\end{aligned} \tag{2.9}$$

với $y_{ij}, \widehat{y}_{ij}$ lần lượt là phần tử thứ j của vector đầu ra thực tế (*one-hot*) $\mathbf{y}_i$ và vector đầu ra dự đoán $\widehat{\mathbf{y}}_i$.

Khi dữ liệu là một tập hợp $(\mathbf{x}_i, \mathbf{y}_i)$, $i = 1, 2, ..., N$ thì hàm mất mát cho *softmax* được tính như sau

$$J(\mathbf{W}; \mathbf{X}, \mathbf{Y}) = -\frac{1}{N} \sum_{i=1}^{N} \left(\sum_{j=1}^{C} y_{ji} \mathbf{a}_i^{(L-1)} \mathbf{w}_{ij}^{(L)T} + \log \left(\sum_{k=1}^{C} \exp(\mathbf{a}_i^{(L-1)} \mathbf{w}_{ik}^{(L)T}) \right) \right) \tag{2.10}$$

2.9 Back-propagation và tối ưu hàm mất mát

2.9.1 Khái niệm

Truyền ngược là Backpropagation, là một từ viết tắt cho "backward propagation of errors" tức là "truyền ngược của sai số", là một phương pháp phổ biến để *huấn luyện* các mạng thần kinh nhân tạo được sử dụng *kết hợp* với một *phương pháp tối ưu hóa* như gradient descent, momentum, nesterov accelerated gradient, adam. Nhìn chung các phương pháp này tính toán gradient của hàm tổn thất với tất cả các trọng số có liên quan trong mạng nơ-ron đó và sử dụng nó để cập nhật các trọng số, để cực tiểu hóa hàm tổn thất. Tùy theo mỗi phương pháp sẽ có các cách cập nhật trọng số khác nhau.

2.9.2 Thuật toán

Algorithm 5 Back-propagation

Require: Network depth, L
Require: $\mathbf{W}^{(i)}, i \in \{1, \ldots, L\}$, ma trận trọng số của layer thứ i của model
Require: $\mathbf{b}^{(i)}, i \in \{1, \ldots, L\}$, bias layer thứ i của model
Require: $\mathbf{X}, \mathbf{x}_i, i \in \{1, \ldots, n\}$, tập dữ liệu đầu vào với n là số lượng dữ liệu
Require: $\mathbf{Y}$, tập đầu ra tương ứng dạng one-hot; $\mathbf{y}_i, i \in \{1, \ldots, N\}$ đầu ra tương ứng với x_i

for $i = 1; i \leq N; i = i + 1$ **do**
  $\widehat{\mathbf{y}} = \mathbf{a}^{(l)} = f(\mathbf{x}_i, \mathbf{W}_i, \mathbf{b}_i)$
  $\mathbf{e}^{(L)} = \dfrac{\partial J(\widehat{\mathbf{y}}, \mathbf{y}_i)}{\partial \widehat{\mathbf{y}}} \dfrac{\partial \widehat{\mathbf{y}}}{\partial \mathbf{z}^{(L)}}$
  $\dfrac{\partial J(\mathbf{W})}{\partial \mathbf{w}_j^L} = \mathbf{e}^{(L)} \dfrac{\partial \mathbf{z}^{(L)}}{\partial \mathbf{w}_j^{(L)}} = \mathbf{a}_j^{(L-1)} \mathbf{e}^{(L)}$, *với mỗi* j; $j = 1, \ldots, m$; m là số units trong layer L
  for $k = L - 1; k \geq 1; k + +$ **do**
    $\mathbf{e}^{(k)} = \mathbf{e}^{(k+1)} \dfrac{\partial \mathbf{z}^{(k+1)}}{\partial \mathbf{a}^k} \dfrac{\partial \mathbf{a}^k}{\partial \mathbf{z}^k}$
    $\dfrac{\partial J(\mathbf{W})}{\partial \mathbf{w}_j^k} = \mathbf{e}^{(k)} \dfrac{\partial \mathbf{z}^{(k)}}{\partial \mathbf{w}_j^{(k)}} = \mathbf{a}_j^{(k-1)} \mathbf{e}^{(k)}$, *với mỗi* j; $j = 1, \ldots, m$; m là số units trong layer k
  end for
  for $k = L; k \geq 1; k + +$ **do**
    cập nhật $\boldsymbol{w}_j^{(k)}$ *theo gradient* $\dfrac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{w}_j^k}$ *bằng một trong các cách tối ưu như: gradient descent, adam, momentum,... với mỗi* j; $j = 1, \ldots, m$; m *là số units trong layer* k
  end for
end for

2.9.3 Ví dụ

Tiếp tục ví dụ với cấu trúc mạng nơ-ron như Hình 2.7, xét một cặp dữ liệu $(\mathbf{x}_i, \mathbf{y}_i)$ với $\mathbf{y}_i$ là dạng *one-hot* tại đầu ra thực tế thứ i. Vì ta hiện ta chỉ xét một cặp dữ liệu vì thế tôi xin được lược bỏ chỉ số i tại các công thức dưới đây. Theo như ta tính bằng *feedforward* chúng ta đã có:

$$\mathbf{a}^{(0)} = \mathbf{x}_{1\times 2}$$

$$\mathbf{z}^{(1)}_{1\times 3} = \mathbf{a}\mathbf{W}^{(1)T}_{2\times 3}$$

$$\mathbf{a}^{(1)}_{1\times 3} = \max(0, \mathbf{z}^{(1)})$$

$$\mathbf{z}^{(2)}_{1\times 3} = \mathbf{a}^{(1)}_{1\times 3}\mathbf{W}^{(2)T}_{3\times 3}$$

$$\widehat{\mathbf{y}}_{1\times 3} = \frac{\exp(\mathbf{z}^{(2)})}{\sum_{j=1}^{3}\exp(\mathbf{z}^{(2)}_j)}.$$

Để thực hiện thuật toán backpropagation có nghĩa là chúng ta đi tìm các giá trị sau: $\frac{\partial J_i(\mathbf{W})}{\partial \mathbf{W}^{(2)}}, \frac{\partial J_i(\mathbf{W})}{\partial \mathbf{W}^{(1)}}$.

Vì ta sử dụng hàm activation ở layer cuối là softmax vì thế ta có hàm mất mát là biểu thức 2.9. Tính $\frac{\partial J(\mathbf{W})}{\partial \mathbf{w}^L}$

$$\begin{aligned}
\frac{\partial J(\mathbf{W})}{\partial \mathbf{w}^L_j} &= \frac{\partial\left(-\sum_{j=1}^{C} y_j \mathbf{a}^{(L-1)}\mathbf{w}^{(L)T}_j + \log\left(\sum_{k=1}^{C}\exp(\mathbf{a}^{(L-1)}_i \mathbf{w}^{(L)T}_j)\right)\right)}{\partial \mathbf{w}^{(L)}_j} \\
&= -y_j\mathbf{a}^{(L-1)} + \frac{\exp(\mathbf{w}^{(L-1)T}_j \mathbf{a}^{(L-1)})}{\sum_{k=1}^{C}\exp(\mathbf{w}^{(L-1)T}_k \mathbf{a}^{(L-1)})}\mathbf{a}^{(L-1)} \\
&= -y_j\mathbf{a}^{(L-1)} + a^{(L)}_j\mathbf{a}^{(L-1)} = \mathbf{a}^{(L-1)}(a^{(L)}_j - y_j) \text{ , với } a^L_j = \widehat{y}_j \\
&= e^{(L)}_j \mathbf{a}^{(L-1)} \text{ (với } e^L_j = a^{(L)}_j - y_j)
\end{aligned} \tag{2.11}$$

Tổng quát ta có:

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}^L} = \mathbf{a}^{L-1}[e_1, e_2, \ldots, e_C] = \mathbf{a}^{L-1}\mathbf{e}^T \tag{2.12}$$

Vẫn với hàm mất mát như trên, nếu ta tính theo trình tự thuật toán ta có:

$$\begin{aligned}
e^{(L)}_j &= \frac{\partial J(\mathbf{W})}{\partial \widehat{y}_j}\frac{\partial \widehat{y}_j}{\partial \mathbf{z}^{(L)}_j} \\
&= -\sum_{j=1}^{C}\frac{y_j \log \widehat{y}_{ij}}{\partial \widehat{y}_j}\frac{\partial \widehat{y}_j}{\partial \mathbf{z}^{(L)}_j} \\
&= -\sum_{s=1}^{C} y_s \frac{\sum_{k=1}^{C}\exp\left(z^{(L)}_k\right)}{\exp\left(z^{(L)}_j\right)} \times \frac{\exp(z^{(L)}_j)\sum_{k=1}^{C}\exp(z^{(L)}_k) - (\exp(z^{(L)}_j))^2}{\left(\sum_{k=1}^{C}\exp(z^{(L)}_k)\right)^2} \\
&= -\sum_{s=1}^{C} y_s\left(1 - \frac{\exp(z^{(L)}_j)}{\sum_{k=1}^{C}\exp(z^{(L)}_k)}\right) \\
&= -\sum_{s=1}^{C} y_s\left(1 - \widehat{y}_j\right) \\
&= -\sum_{s=1}^{C}(y_s - y_s\widehat{y}_j) = \widehat{y}_j - y_s, \text{ do đầu ra thực tế là dạng one-hot}
\end{aligned} \tag{2.13}$$

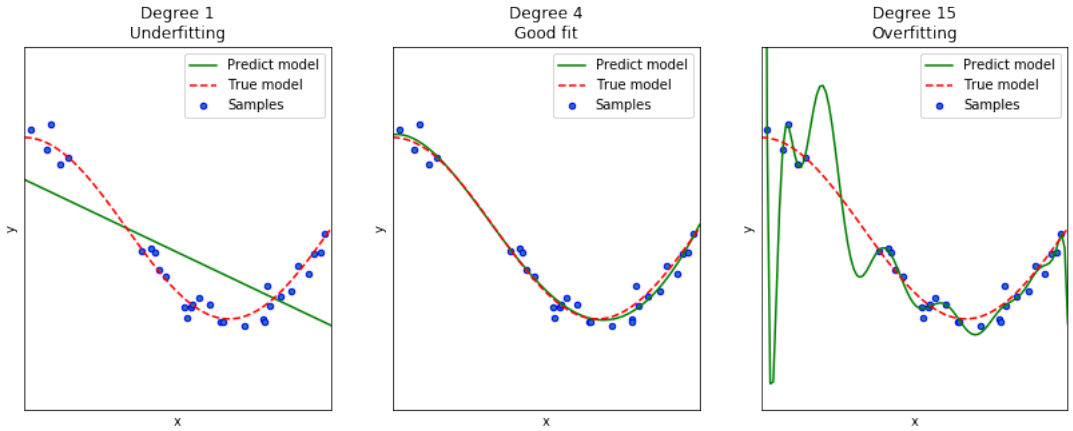
$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{w}_j^L} = e^{(L)} \frac{\partial \widehat{\mathbf{y}}}{\partial \mathbf{z}^{(L)}}$$
$$= -\sum_{s=1}^{C} y_s \left(1 - \widehat{y}_j\right) \mathbf{a}^{L-1}$$

2.10 Overfitting

Overfitting là một hiệnn tượng không mong muốn thường gặp trong quá trình xây dựng mô hình, người xây dựng mô hình cần nắm được kỹ thuật để tránh hiện tượng này xảy ra.

2.10.1 Tổng quan

Để hiểu vấn đề này chúng ta sẽ cùng nhau xem qua ví dụ sau Hình 2.9.



Hình 2.9: overfitting và underfitting

Đường nét liền thể hiện *mô hình dự đoán (predicted model)* , đường nét đứt thể hiện *mô hình thực (true model)*, các chấm hình tròn là các điểm dữ liệu. Mô hình được xây dựng bằng hồi quy tuyến tính (*linear regression*) với các feature là bậc mũ.

Ở hình thứ nhất chúng ta có thể thấy mô hình dự đoán là một hàm tuyến tính (bậc bằng 1) rất khác với mô hình thực, xa với các điểm dữ liệu. Hiện tượng này ta nói mô hình bị *underfitting*. Với mô hình dự đoán là đa thức bậc 4 chúng ta có thể thấy mô hình dự đoán xấp xỉ như mô hình thực (hình thứ 2). Trường hợp này ta nói mô hình phù hợp (*good fit*). Ở hình thứ 3, khi ta tăng bậc đa thức lên thì mô hình dự đoán quá khớp với các điểm dữ liệu, gần như mọi điểm dữ liệu đều nằm trên mô hình. Tuy nhiên việc khớp hoàn toàn dữ liệu lại không hề tốt vì dữ liệu thường bị nhiễu và có thể khiến mô hình dự đoán bị nhiễu hơn. Trường hợp này ta nói mô hình bị *overfitting*.

Một vài khái niệm về mô hình

1. *Underfitting* là hiện tượng mô hình chưa được phù hợp với tập dữ liệu huấn luyện và cả các mẫu mới khi dự đoán. Nguyên nhân có thể là do mô hình chưa đủ độ phức tạp cần thiết để bao quát được tập dữ liệu.

2. *Overfitting* là hiện tượng mô hình quá khớp với *tập dữ liệu huấn luyện (training set)*, việc này sẽ gây ra hậu quả vô cùng nghiêm trọng nếu tập dữ liệu huấn luyện xuất hiện nhiễu. Mô hình sẽ chỉ chú trọng vào việc xấp xỉ với tập dữ liệu huấn luyện mà quên đi mục đích ban đầu là tổng quát hóa, làm cho mô hình sẽ không thật sự tốt dối với dữ liệu

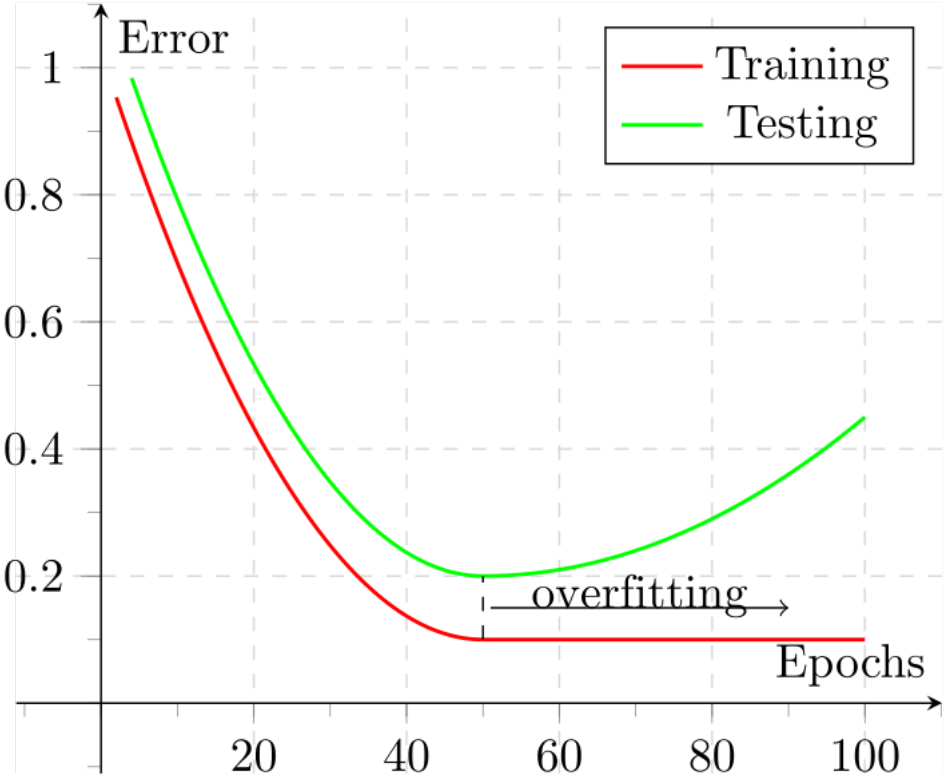
nằm ngoài dữ liệu huấn luyện (dữ liệu test và dữ liệu thực tế). Overfitting xảy ra khi *độ phức tạp của mô hình quá lớn* hoặc *quá ít dữ liệu.*

3. *Good fitting* là mô hình nằm giữa 2 mô hình chưa khớp (*underfitting*) và quá khớp (*overfitting*) cho ra kết quả hợp lý với cả tập dữ liệu huấn luyện và các giá trị mới, tức là nó mang được tính tổng quát như hình 1 ở giữa phía trên. Lý tưởng nhất là khớp được với nhiều dữ liệu mẫu và cả các dữ liệu mới. Tuy nhiên trên thực tế được mô hình như vậy rất hiếm.

.

2.10.2 Mối liên hệ giữa overfitting và giá trị hàm mát mát

Như tôi đã trình bày phần trên, overfitting là hiện tượng mô hình quá khớp với tập dữ liệu huấn luyện có nghĩa là giá trị của hàm mất mát trên tập dữ liệu huấn luyện (J_{train}) rất nhỏ. Nhưng khi đó giá trị của hàm mất trên tập dữ liệu test (J_{test}) lại tăng lên gây ra mô hình mất đi sự tổng quát.



Hình 2.10: Overfitting xảy ra khi giá trị mất mát giữa tập train giảm còn tập test tăng

2.10.3 Regularization

Regularization là thay đổi mô hình một chút, chấp nhận hy sinh độ chính xác trong trainning set nhưng giảm độ phức tạp của mô hình, do đó giúp tránh được được overfitting mà vẫn giữ được tính tổng quát của mô hình.

2.10.3.1 Early stopping

Khi ta dùng một phương pháp tối ưu hàm số để giảm thiểu giá trị mất mát thì J_{train}, J_{test} sẽ cùng giảm theo thời gian nhưng nếu sau một thời gian J_{test} tăng lên còn J_{train} tiếp tục giảm thì đó là lúc bắt đầu dẫn đến overfitting. Cách đơn giản nhất để giảm thiểu overfitting đó là dừng huấn luyện tại ngay thời điểm bắt đầu overfitting và phương pháp này được gọi là *early stopping*. Nếu ta có biểu đồ về sự thay đổi giá trị mất mát của trainning và testing như Hình 2.10 thì ta có thể thấy thời điểm sử dụng early stopping là vào khoảng epochs 50.

2.10.3.2 Thêm số hạng vào hàm mất mát

Một kỹ thuật regulazation phổ biến là thêm một số hạng vào hàm mất mát như sau:

$$J_{reg}(\mathbf{W}) = J(\mathbf{W}) + \lambda R(\mathbf{W}) \tag{2.14}$$

$J(\mathbf{W})$ là hàm mất mát ban đầu và cụm $\lambda R(\mathbf{W})$ mới thêm vào là số hạng chính quy hoá (hay số hạng regularization) đóng vai trò như một biện pháp phạt lỗi (penalization).

Trong đó, tham số chính quy hoá (*regularizaton parameter*) λ được chọn từ trước để cân bằng giữa $J(\mathbf{W})$ và $R(\mathbf{W})$. λ càng lớn thì ta càng coi trọng $R(\mathbf{W})$, ít coi trọng tham số cho hàm mất mát ban đầu hơn, dẫn tới việc các trọng số $\mathbf{W}$ ít có ảnh hưởng tới mô hình hơn. Hay nói cách khác là mô hình bớt phức tạp đi giúp ta đỡ việc lỗi quá khớp.

$R(\mathbf{W})$ thường có dạng như sau:

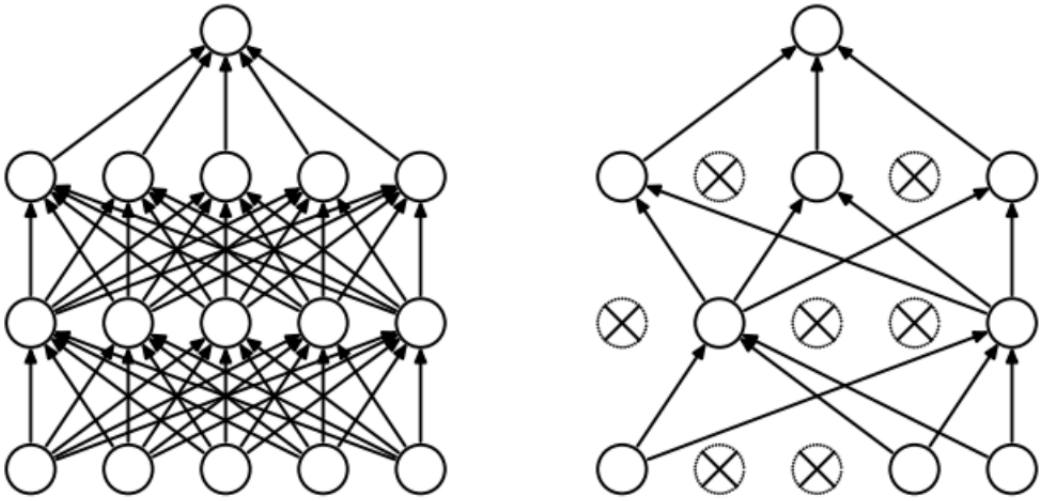
$$R(\mathbf{W}) = \frac{1}{p}\|\mathbf{W}\|_p^p = \frac{1}{p}\sum_i^n |\mathbf{W}|^p \tag{2.15}$$

p thường được chọn là 2 ($l_2 normregularization$) và 1 ($l_1 normregularization$)

Phương pháp chính quy hoá này còn có tên là cắt trọng số (weight decay) vì nó khiến các hệ số trong $\mathbf{W}$ không quá lớn, giúp tránh việc đầu ra phụ thuộc quá nhiều vào một đặc trưng nào đó.

2.10.3.3 Drop-out

Drop-out là một kĩ thuật Regularization để chống lại vấn đề overfitting. Cách dropout thực hiện là xoá bỏ một số unit trong các step training ứng với một giá trị xác suất $\mathbf{p}$ cho trước. Các mạng mới sau khi áp dụng dropout được gọi là subsample. Thông thường xắc suất ở layer input bằng 0.8 hay ta loại bỏ khoảng 20% số unit, ở hidden layer thì xắc suất là 0.5 có nghĩa là ta loại bỏ 50% số unit ở layer sử dụng dropout.



Hình 2.11: Dropout với p =0.5 và (b) là một subsample

Cách hoạt động của dropout

- Dropout được áp dụng trên một layer của mạng neural networks với một xác suất **p** cho trước (có thể sử dụng nhiều Drop-Out khác nhau cho những layer khác nhau, nhưng trên 1 layer sẽ chỉ có 1 dropout)
- Tại mỗi step trong quá trình training, khi thực hiện feedforward đến layer sử dụng dropout, thay vì tính toán tất cả unit có trên layer, tại mỗi unit ta "gieo xúc xắc" với xắc suất p xem unit đó được tính (active) hay không được tính (deactive). Những unit active ta tính toán bình thường còn với những unit deactive thì ta set giá trị tại unit đó bằng 0
- Trong quá trình test thì tất cả các unit đều được active và chúng ta mong muốn đầu ra của các units giống với đầu ra mong đợi trong quá trình trainning. Ví dụ đầu ra của một unit (trước khi drôput) là **a**, khi áp dụng dropout thì đầu ra mong đợi của unit đó sẽ là $\mathbf{pa} + (1 - \mathbf{p})0$, vì unit bị deactive thì giá trị của unit đó là 0. Do đó trong quá trình test chúng ta điều chỉnh đầu ra $\mathbf{a} \rightarrow \mathbf{pa}$ để giống với đầu ra mong đợi.

Thời gian test khá là quan trọng nên nếu chúng ta điều chỉnh đầu ra ở các layer áp dụng dropout thì hiệu suất test sẽ bị giảm đi. Vì thế thay vì chỉnh trong quá trình test thì chúng ta sẽ thực hiện việc này trong quá trình trainning. Ta sẽ lấy *dropout mask* (vector xắc suất được khởi tạo ngẫu nhiên, tại vị trí có giá trị nhỏ hơn p sẽ được giữ nguyên còn lớn hơn p sẽ set lại giá trị vị trí đó là 0) chia cho p trong quá trình trainning. Trường hợp này được gọi là *inverted dropout*.

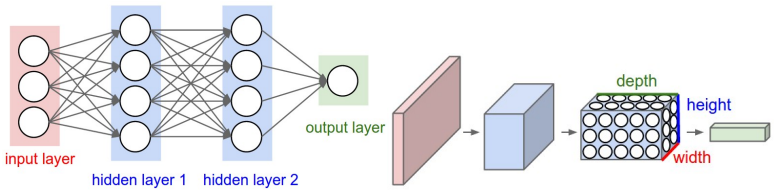
Chương 3

Convolutional Nerual Networks

3.1 Tổng quan

Convolutional Neural Network (CNNs – Mạng nơ-ron tích chập) là một trong những mô hình Deep Learning tiên tiến giúp cho chúng ta xây dựng được những hệ thống thông minh với độ chính xác cao. Trong khóa luận này, tôi sẽ trình bày về Convolution (tích chập) cũng như xây dựng mô hình CNNs cho bài toán nhận dạng chữ viết tay trong tiếng Nhật (Image Classification).

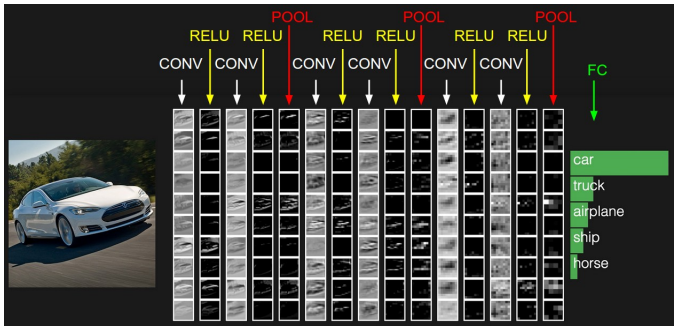
Lấy cảm hứng từ xử lý ảnh nên đầu vào và các lớp của CNNs có dạng như một bức ảnh chứ không có dạng vector như Neural Networks thông thường. Cụ thể, một bức ảnh sau khi số hoá có dạng *width* × *height* × *depth* (width: số lượng điểm ảnh trên chiều rộng, height: số lượng điểm ảnh trên chiều cao, depth: số lượng kênh chẳng hạn như RGB có 3 kênh đại diện cho mức độ của 3 màu Đỏ, Lục, Lam). Mô hình được mô tả ở Hình **??**



(a) mạng nơron thông thường (b) mạng nơron tích chập

Hình 3.1: Kiến trúc hai mạng

Một mạng CNNs đơn giản có ba lớp chính: lớp tích chập (*convolutional layer*), lớp giảm số chiều (*pooling layer*), lớp fully-connected. Các lớp convolutional và lớp pooling được xếp xen kẽ nhau Hình 3.2.



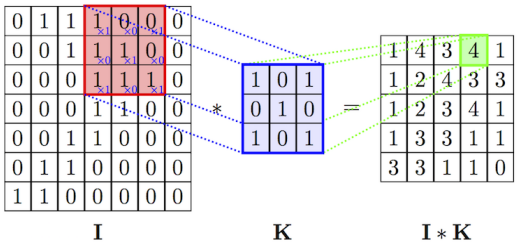
Hình 3.2: ví dụ về kiến trúc của ConvNets

3.2 Bộ lọc và cửa sổ trượt

Trước khi đi đến chi tiết các lớp, chúng ta cần hiểu *bộ lọc (filter)* là gì, *cửa sổ trượt (sliding window)* là gì.

3.2.1 Bộ lọc - filter

Như tôi đã trình bày ở phần trên, kích thước của đầu vào đã thay đổi thành dạng khối. Vì thế kích thước của trọng số cũng được thay đổi thành dạng khối để phù hợp với hình dạng của đầu vào và nó còn được gọi là *filter* hoặc *kennel*. Ví dụ như Hình 3.3, ma trận I là ma trận đầu vào, ma trận K là filter và ma trận I*K là ma trận kết quả (được trình bày ở phần sau).

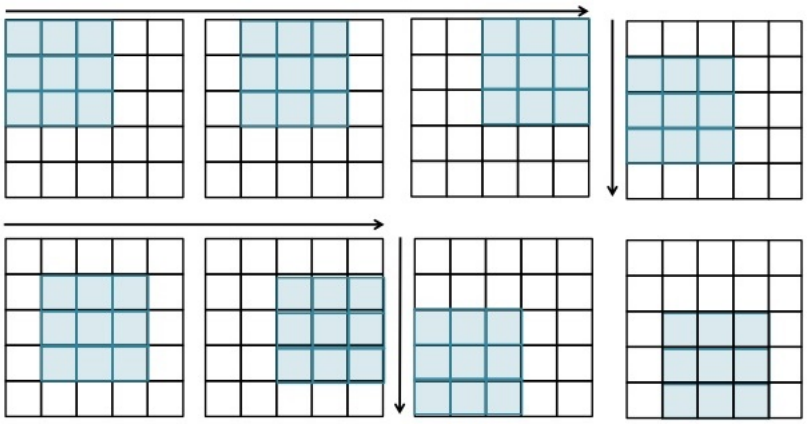


Hình 3.3: Bộ lọc - filter

Kích thước của các filter thường là $d \times 3 \times 3$, $d \times 5 \times 5$, $d \times 7 \times 7$ hoặc $d \times 11 \times 11$, trong đó d là chiều sâu của filter, hai tham số tiếp theo thể hiện chiều rộng và chiều cao của filter. Chú ý rằng chiều sâu của filter luôn luôn bằng chiều sâu của đầu vào.

3.2.2 Cửa sổ trượt - sliding window

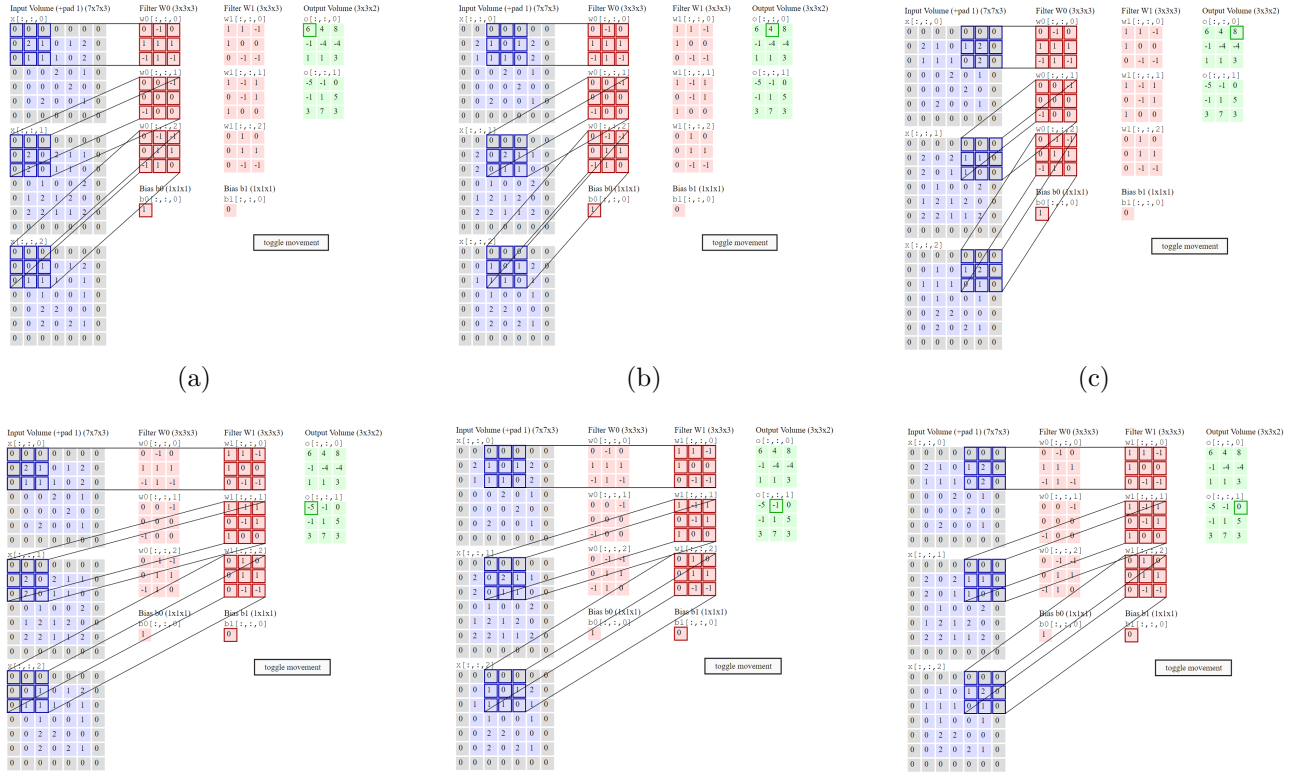
Cửa sổ trượt (sliding window) là ta chọn một "cửa sổ" có kích thước nhỏ hơn kích thước đầu vào và trượt cửa sổ đó trên đầu vào theo chiều ngang và chiều dọc. Các bước trượt là khoảng cách mà ta dịch chuyển filter trên đầu vào và đó gọi là *stride*, ký hiệu là s. Ví dụ như Hình 3.4, ta có đầu vào kích thước là 5×5, cửa sổ có kích thước là 3×3 và tại mỗi bước trượt cửa sổ dịch chuyển đi một đơn vị, do đó stride bằng 1.



Hình 3.4: Sliding window

3.3 Lớp tích chập - Convolutional layer

Convolution layer là khối cốt lõi, cơ bản của ConvNets, lớp này chủ yếu nặng về việc tính toán. Lớp này chính là nơi thể hiện tư tưởng ban đầu của mạng nơron tích chập. Thay vì kết nối toàn bộ điểm ảnh, lớp này sẽ sử dụng bộ lọc (*filter*) áp vào một vùng trong ảnh và tiến hành tính tích chập giữa bộ filter và giá trị điểm ảnh trong vùng cục bộ đó, vùng này ta gọi là *receptive filter*. Sau đó ta dùng sliding window để trượt filter và tiến hành tích chập tại mỗi vùng ta trượt đến. Tôi sẽ trình bày chi tiết thông qua Hình 3.5.



Hình 3.5: Tính toán trong lớp tích chập

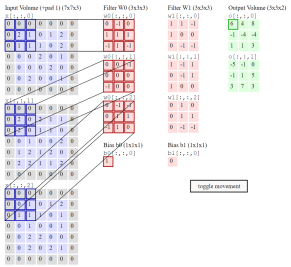
Theo Hình 3.5 ta có kích thước đầu vào là $7 \times 7 \times 3$, stride bằng 2, kích thước filter là $2 \times 3 \times 3 \times 3$ với 2 là số *channel* (số chanel sẽ là chiều sâu output chúng ta mong muốn). Tại Hình 3.5a ta thực hiện phép tính... tương ứng giữa receptive filter, $filter\ W0$ tại tất cả các độ sâu, kết quả được cộng thêm *bias* rồi lưu vào ma trận output vị trí đầu tiên của đầu ra. Sau đó ta thực hiện cách tính toán trên kết hợp sliding window với stride bằng 2 và filter là $filterW0$, kết quả ta thu được là ma trận output thứ nhất, các Hình 3.5a, 3.5b, 3.5c mô tả giai đoạn này. Và tương tự như trên ta thực hiện với $filter\ W1$ theo Hình 3.5d, 3.5e, 3.5f ta thu được kết quả là ma trận thứ hai tại output. Như vậy nếu ta có càng nhiều channel thì chiều sâu output của chúng ta càng lớn.

Nếu như đầu ta chỉ tính trên kích thước đầu vào thì với filter 3×3 và stride bằng 2 thì ta chỉ trượt được hai lần và đến hàng, cột thứ 5 của đầu vào còn giá trị tại hàng, cột thứ 6, 7 sẽ không được tính. Vì thế chúng ta thêm vào ma trận đầu vào các hàng và cột đối xứng với giá trị bằng 0 để không ảnh hưởng đến đầu vào mà khi đó ta sẽ quét được hết các giá trị của đầu vào. Cách này được gọi là *zero padding*. Như ở ví dụ trên ta thấy đầu vào được bao quanh bởi các số 0 và kích thước tăng thành 8×8, đó là ta áp dụng zero padding với p=1.

Ta có công thức để tính toán kích thước đầu ra như sau:

- Kích thước đầu vào W_1 x H_1 x D_1 (rộng x cao x sâu)

- Kích thước của filter F
- Số channel K
- Tốc độ trượt S
- Giá trị của zero-padding P
- Kích thước đầu ra W_2 x H_2 x D_2:
  + $W_2 = (W_1 - F + 2P)/S + 1$
  + $H_2 = (H_1 - F + 2P)/S + 1$
  + $D_2 = K$



Hình 3.6: Ví dụ về tính toán kích thước đầu ra

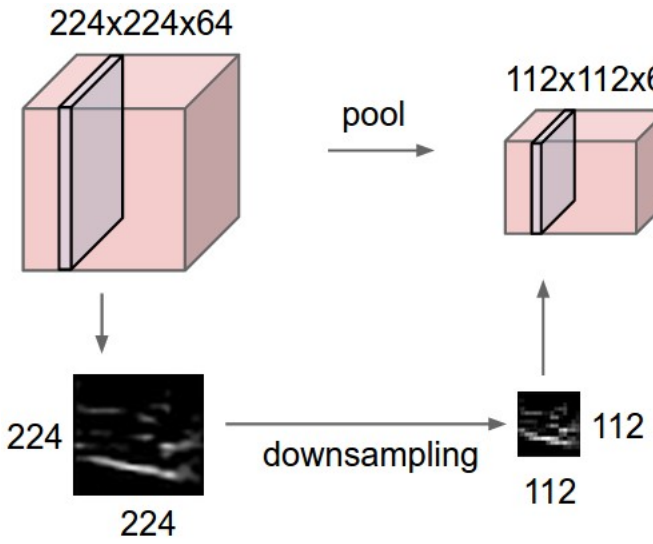
Giả sử ta có thông số như hình 3.6 với kích thước đầu vào là $7 \times 7 \times 3$ tương đương $W = H = 7$, $D_1 = 3$;stride $S = 3$; kích thước filter là $2 \times 3 \times 3 \times 3$ hay $F = 3$ và số channel $K = 2$. Theo công thức ta tính được $(W_1 - F + 2P)/S$ không phải là số nguyên do đó ta cần thêm padding để giá trị này là số nguyên. Ta chọn $P = 1$ vì $(7 - 3 + 2 \times 1)/3 + 1$ là số nguyên. Như vậy ta tính được kích thước đầu ra sẽ là $3 \times 3 \times 2$.

Một phần quan trọng của layer này đó là hàm activation, hàm này được tính với đầu vào là kết quả sau khi thực hiện tích chập. Và ReLU thường được chọn là hàm activation do tính đơn giản của nó. Nhiệm vụ của nó là chuyển toàn bộ giá trị âm trong kết quả sau khi tích chập thành 0. Ý nghĩa của cách cài đặt này chính là tạo nên tính phi tuyến cho mô hình.

3.4 Lớp giảm số chiều - pooling layer

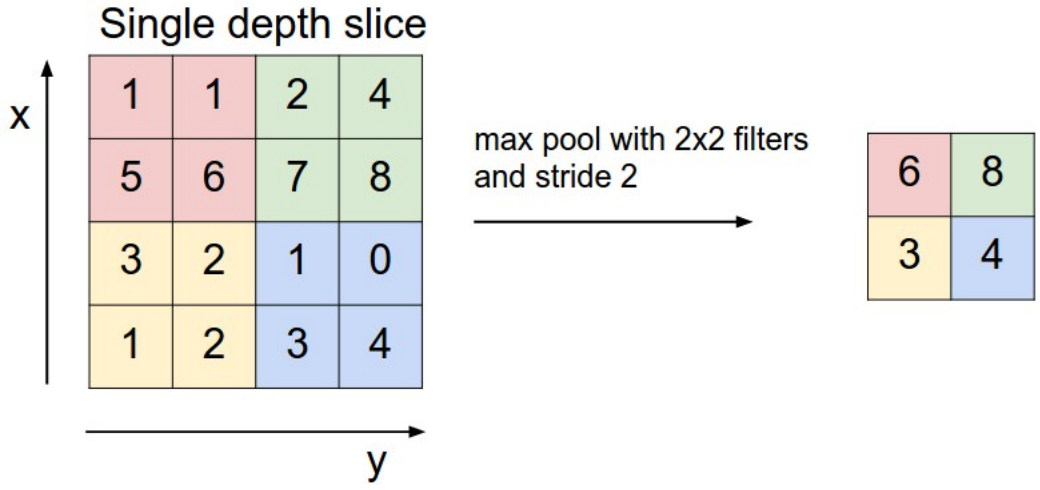
Lớp giảm số chiều (pooling layer) trong mạng CNNs thực hiện công việc loại bỏ bớt những thông tin không cần thiết sau khi thực hiện tích chập và được chèn giữa các lớp convolutional với nhau hoặc sau một tập lớp convolutional. Nó có vai trò giảm kích thước dữ liệu. Với một bức ảnh kích thước lớn qua nhiều lớp pooling sẽ được thu nhỏ lại tuy nhiên vẫn giữ được những đặc trưng cần cho việc nhận dạng (thông qua cách lấy mẫu). Việc giảm kích thước dữ liệu sẽ làm giảm lượng tham số, tăng hiệu quả tính toán và góp phần kiểm soát hiện tượng quá khớp (overfitting). Tuy nhiên nếu lạm dụng loại layer này cũng có thể khiến data đi qua bị mất dữ liệu.

Cách thức hoạt động: Pooling layer sử dụng sliding window và tại mỗi cửa sổ trượt trên đầu vào chỉ có một giá trị được xem là giá trị đại diện cho thông tin ảnh tại vùng đó (giá trị mẫu) được giữ lại và đó gọi là tiến hành lấy mẫu (*subsampling*). Các phương thức lấy phổ biến trong pooling layer là *Max pooling* (lấy giá trị lớn nhất), *Min pooling* (lấy giá trị nhỏ nhất) và *Average Pooling* (lấy giá trị trung bình). Hình 3.7 mô tả tiến hành lấy mẫu bằng max pooling.



Hình 3.7: Minh họa giảm số chiều

Ví dụ ta có ma trận đầu vào 4×4 như hình 3.8, với kích thước cửa sổ áp dụng cho slidiing window là 2×2 và stride $= 2$, điều này tương đương với việc ta chia ma trận đầu vào thành 4 ma trận con với kích thước là 2×2. Phương thức ta áp dụng tại đây là max pooling, do đó tại mỗi cửa sổ khi ta áp vào đầu vào thì ta sẽ chọn giá trị lớn nhất làm đại diện cho cửa sổ đó và ta thu được kết quả là ma trận 2×2 bên tay phải.



Hình 3.8: Max pooling

Ta rút ra được công thức tính kích thước đầu ra cho lớp này như sau:

- Kích thước đầu vào W_1 x H_1 x D_1 (rộng x cao x sâu),
- Kích thước cửa sổ F,
- Tốc độ trượt S
- Kích thước đầu ra W_2 x H_2 x D_2:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$

- $D_2 = D_1$

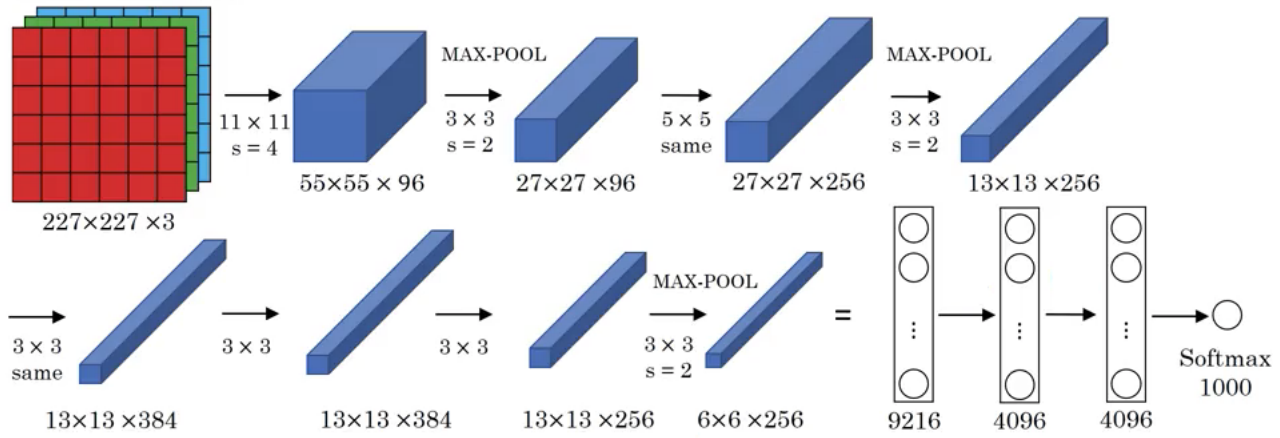
Chú ý rằng chúng ta không thường xuyên sử dụng *zero padding* cho lớp này.

3.5 Lớp Fully-Connected (Fully-Connected layer)

Sau khi ảnh được xử lý và trích xuất đặc trưng bằng các lớp convolutional, pooling thì ta sẽ làm phẳng (flatten) output cuối cùng của giai đoạn trước đó và áp dụng mạng nơron truyền thẳng với input là dữ liệu ta vừa làm phẳng. Hay nói cách khác Fully-Connected chính là một mạng nơron được gắn vào phần cuối của CNNs với input là đầu ra của các lớp trước đó. Nó đóng vai trò như một mô hình phân lớp và tiến hành dựa trên dữ liệu đã được xử lý ở các lớp trước đó.

3.6 Tổng kết

Dưới đây là một mô hình mạng CNNs kết hợp giữa các layer với nhau với cấu trúc như sau: Conv → Max-pool → Conv (giữ nguyên kích thước) → Max-pool → Conv (giữ nguyên kích thước) → Conv (giữ nguyên kích thước) → Conv (giữ nguyên kích thước) → Max-pool → Flatten → FC . Kích thước các lớp được thể hiện tại Hình 3.9.



Hình 3.9: Ví dụ cấu trúc mạng CNNs