MỞ ĐẦU

Deep Learning là một thuật toán dựa trên một số ý tưởng từ não bộ tới việc tiếp thu nhiều tầng biểu đạt, cả cụ thể lẫn trừu tượng, qua đó làm rõ nghĩa của các loại dữ liệu. Deep Learning được ứng dụng trong nhận diện hình ảnh, nhận diện giọng nói, xử lý ngôn ngữ tự nhiên. Với dữ liệu khổng lồ hiện nay thì deep learning được ứng dụng vào rất nhiều các bài toán nhận dạng và cho thấy tính hiệu quả, độ chính xác cao so với các phương pháp truyền thống.

Những năm gần đây, ta đã chứng kiến được nhiều thành tựu vượt bậc trong ngành Thị giác máy tính (Computer Vision). Các hệ thống xử lý ảnh lớn như Facebook, Google hay Amazon đã đưa vào sản phẩm của mình những chức năng thông minh như nhận diện khuôn mặt người dùng, phát triển xe hơi tự lái hay drone giao hàng tự động.

Mạng nơ-ron tích chập (Convolutional Neural Network - CNN) là một trong những mô hình deep learning tiên tiến giúp chúng ta xây dựng được những hệ thống thông minh với độ chính xác cao như hiện nay. Trong khóa luận này, tôi đi vào nghiên cứu về mạng nơ-ron cũng như mạng nơ-ron tích chập, ý tưởng của mô hình CNN trong phân lớp ảnh (Image Classification), và áp dụng trong việc xây dựng hệ thống phân loại biển báo giao thông.

Nội dung khóa luận gồm 4 chương:

Chương 1: Tổng quan về bài toán phân loại biển báo giao thông. Tại chương này tôi sẽ trình bày khái quát về bài toán phân loại biển báo và ứng dụng của nó trong tương lai.

Chương 2: Cở sở toán học. Ở chương này tôi nhắc lại một số kiến thức cơ bản để tiếp cận với cách hoạt động của mạng nơ-ron dễ dàng hơn.

Chương 3: Mạng nơ-ron và mạng nơ-ron tích chập. Chương này sẽ trình bày cấu trúc, thành phần và cách hoạt động của mạng nơ-ron và mạng nơ-ron tích chập.

Chương 4: Ứng dụng mạng nơ-ron tích chập vào bài toán phân loại biển báo giao thông. Tại chương này, tôi sẽ trình bài cách áp dụng mạng nơ-ron cho bài toán phân loại biển báo giao thông.

Mục lục

Danh sách hình vẽ

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

Cơ Sở Toán Học

1.1 Ma trận

1.1.1 Định nghĩa

Một ma trận **A** loại (cấp) $m_x n$ 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 $m_x n$, ta có tổng $\mathbf{A}+\mathbf{B}$ là ma trận có cùng kích thước $(m_x n)$ 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**:

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

Ngoài ra có một phép nhân khác được gọi là *element-wise* (hay *hadamard*) được sử dụng khá nhiều trong học máy. Tích element-wise 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 $z = f(y), y = g(x)$:

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

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

$$h_\theta = P(Y|X;\theta) \tag{1.1}$$

hay biểu thức 1.1 được viết lại như sau:

$$\widehat{\theta} = \arg\max_\theta P(Y|X;\theta) \tag{1.2}$$

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

$$P(Y|X;\theta) = \prod_{i=1}^{N} P(y_i|x_i,\ \theta) \tag{1.3}$$

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:

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

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: $\theta = [\theta_1, \theta_2, ..., \theta_m]$
1: **repeat**
2: **for all** $i = 1; i \leq N; i++$ **do**
3: $\nabla\theta_i := -\nabla_\theta f(\theta; \mathbf{x}_i, y_i)$
4: **end for**
5: $\nabla\theta = \frac{1}{N} \sum_{i=1}^{N} \nabla\theta_i$
6: Chọn bước nhảy η
7: Cập nhật $\theta := \theta - \eta\nabla\theta$
8: **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

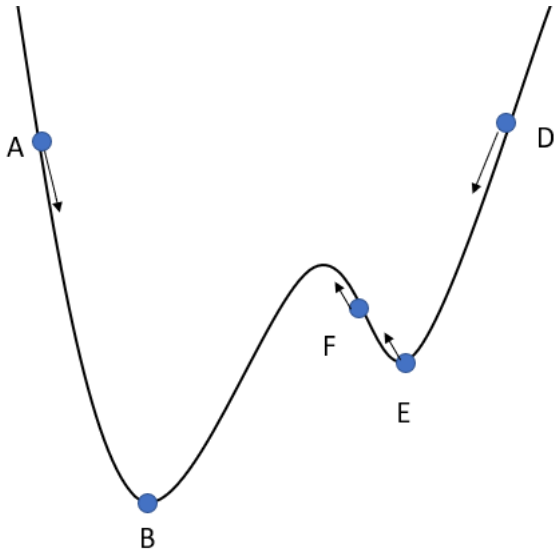
Khác với SGD, thay vì mỗi *iteration* ta tính toán trên một quan sát thì ta sẽ tính toán với k quan sát ($1 < k \ll N$). Phương pháp này được gọi là *mini-batch gradient descent.*

Algorithm 3 Mini-batch Gradient descent

Require: Tập n quan sát ($\mathbf{X}$,$\mathbf{y}$)
Require: $\theta = [\theta_1, \theta_2, ..., \theta_m]$
repeat
  Xáo trộn dữ liệu
  for all $i = 1; i \leq N; i = i + k$ **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

Mạng Nơ-ron

2.1 Tổng quan

2.1.1 Định nghĩa

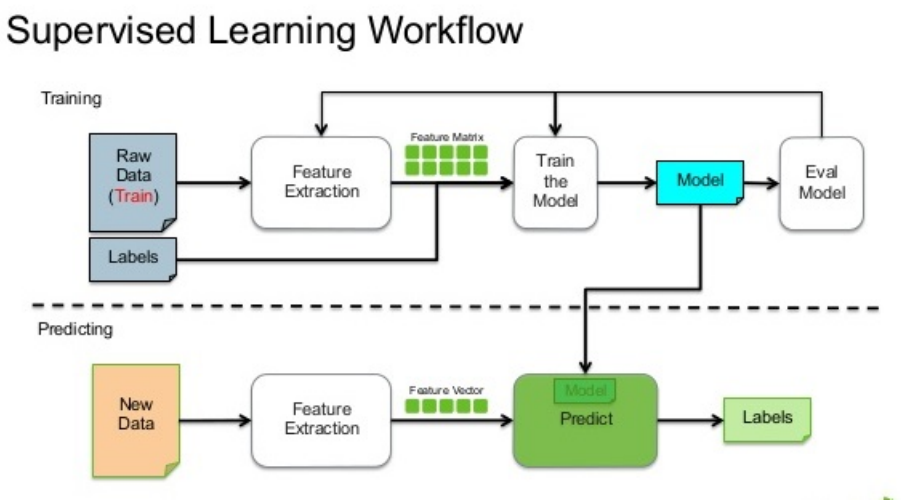
Mạng nơron nhân tạo, Artificial Neural Network (ANN) là một mô hình xử lý thông tin phỏng theo cách thức xử lý thông tin của các hệ nơ-ron sinh học. Nó được tạo nên từ một số lượng lớn các phần tử (nơ-ron) kết nối với nhau thông qua các liên kết (trọng số liên kết) làm việc như một thể thống nhất để 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.

2.1.2 Các phương pháp học

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.

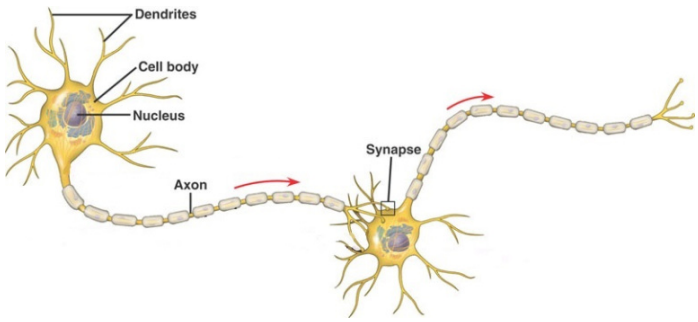


Hình 2.1: Học có giám sát

2.2 Cấu tạo của nơ-ron

2.2.1 Nơ-ron sinh học

Cách thức hoạt động của bộ não nói riêng và của hệ thần kinh nói chung đã được con người quân tâm nghiên cứu từ rất lâu nhưng cho đến nay các nhà khoa học vẫm chưa thực sự hiểu rõ chi tiết về hoạt động của bộ não và hệ thần kinh. Đặc biệt là trong các hoạt động liên quan đến trí óc như suy nghĩ, học tập, tư duy, trí nhớ, sáng tạo....Tuy nhiên, các nhà khoa học cũng có một số thông tin căn bản về bộ não con người. Theo đó, một bộ não con người trung bình cân nặng khoảng 1,5kg và có thể tích là 235 cm^3 , cấu tạo bộ não được chia ra làm nhiều vùng khác nhau, mỗi vùng kiểm soát một hay nhiều hoạt động của con người. Hoạt động của cả hệ thống thần kinh bao gồm não bộ và các giác quan như sau: đầu tiên con người nhận được kích thích bởi các giác quan từ bên ngoài hoặc trong cơ thể. Các kích thích này được biến thành các xung điện bởi chính các giác quan tiếp nhận kích thích. Những tín hiệu này được chuyển về trung ương thần kinh là bộ não để xử lý. Tại bộ não các thông tin sẽ được xử lý, đánh giá và so sánh với các thông tin đã được lưu trữ để đưa ra các quyết định dưới dạng các xung điện. Từ những quyết định từ bộ não sẽ sinh ra các mệnh lệnh cần thiết và gửi đến những bộ phận thi hành thích hợp như các cơ tay, chân, giác quan. . . .



Hình 2.2: Minh họa cấu tạo nơ-ron sinh học

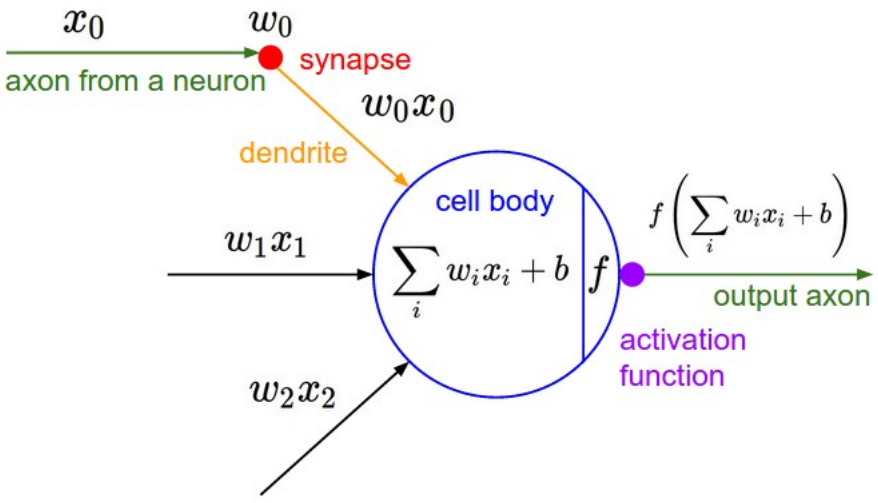
Khi xem xét ở mức độ tế bào thì bộ não được hình thành từ 10^11 phần tử gọi là nơ-ron (hay neural sinh học). Mỗi neural được liên kết chặt chẽ với 10^4 neural khác. Các neural này có nhiều đặc điểm chung với các tế bào khác trong cơ thể, ngoài ra chúng có những khả năng mà

các tế bào khác không có được đó là khả năng nhận, xử lý và truyền các tín hiệu điện hóa làm cơ sở hình cách thức xử lý thông tin của bộ não. Hình 2.2 mô tả cấu tạo và cách thức hoạt động của neural sinh học, trong đó mỗi nơ-ron sinh học có 4 thành phần cơ bản:

- Các nhánh tín hiệu vào (denrites) đây chính là các mạng dạng cây của các dây thần kinh truyền tín hiệu vào đến thân nơ-ron.
- Thân nơ-ron (cell body) chứa nhân (nucleus) hay một số tài liệu gọi là soma có nhiệm vụ chính là tổng hợp và xử lý các tín hiệu điện nhận vào từ các đầu vào. Bản chất của quá trình này chính là việc lấy tổng tất cả các tín hiệu neural nhận được.
- Sợi trục ra (axon) có chức năng truyền tín hiệu từ thân tế bào này sang nơ-ron khác. Phần cuối của axon được chia thành nhiều nhánh nhỏ (cả của denrites và axon) kết thúc tại khớp nối (Synapse).
- Khớp nối (Synapse) là điểm liên kết giữa sợi trục ra của nơ-ron này với các nhánh denrites của neural khác. Liên kết giữa các nơ-ron và độ nhạy của mỗi synapse được xác định bởi quá trình học phức tạp. Khi điện thế của synapses tăng lên do xung điện phát ra từ axon thì synapses sẽ tiết ra một loại hóa chất để kết nối mở ra cho các ion đi qua nó. Các ion này làm thay đổi tín hiệu điện thế trên các điểm tiếp xúc tạo ra các xung điện lan truyền tới các neural khác.

2.2.2 Nơ-ron nhân tạo

Cũng giống như nơ-ron sinh học, nơ-ron nhân tạo cũng nhận đầu vào và thông qua quá trình xử lý đầu vào đó để thu được kết quả là đầu ra. Các nhánh tín hiệu vào sẽ được nhận từ các sợi trục ra của các nơ-ron khác tại khớp nối và chuyển thông tin vào trong nhân, nhân nơ-ron sẽ là với hàm tổng, hàm kích hoạt và sợi trục ra của nơ-ron tương đương với đầu ra. Các thành phần trên được trình bày qua Hình 2.3.



Hình 2.3: Cấu trúc của một nơ-ron, nguồn: https://cs231n.github.io/

– *Đầu vào (input)*: là các tín hiệu vào của nơ-ron, các tín hiệu này thường đưa vào dưới dạng một vector, kí hiệu là $\mathbf{x}$.

– *Trọng số (weight)*: mỗi liên kết được thể hiện bởi một trọng số liên kết và thường được kí hiệu w. 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.

– *Ngưỡng (bias)*: là tham số nhằm tăng khả năng thích ứng của mạng nơron trong quá trình học và thường được kí hiệu là b. Bias gần giống như trọng số, trừ một điều là nó luôn có tín hiệu vào không đổi bằng 1.

– *Hàm kết hợp (combination function)*: 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 *netinput*. Thông thường hàm này sẽ là hàm tổng của các tích giữa trọng số với đầu vào sau đó cộng thêm bias, được biểu diễn thông qua biểu thức $z = \sum_{i=1}^{n}(x_i w_i) + b$.

– *Hàm kích hoạt (activiation function hoặc transfer function)*: Hàm này được dùng để giới hạn phạm vi đầu ra của mỗi nơ-ron. Nó nhận đầu vào là kết quả của hàm kết hợp và ngưỡng đã cho. Thông thường hàm kích hoạt sẽ là các hàm phi tuyến tính.

– *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 (2.3.1) 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:

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

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

Một số hàm kích hoạt

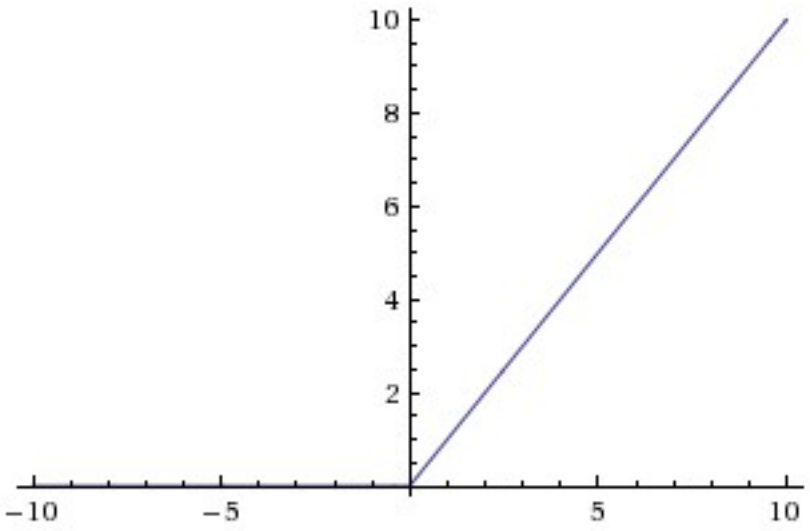
1. *Hàm ReLU* (rectified linear unit function) là hàm phi tuyến tính đơn giản nhất, có công thức toán học $f(s) = \max(0, s)$. Đồ thị hàm ReLU được thể hiện ở Hình 2.4

   Ta thấy hàm ReLU có đầu ra là 0 nếu đầu vào nhỏ hơn hoặc bằng 0, và đầu ra bằng đầu vào trong trường hợp ngược lại. Hàm ReLU có đạo hàm như sau:

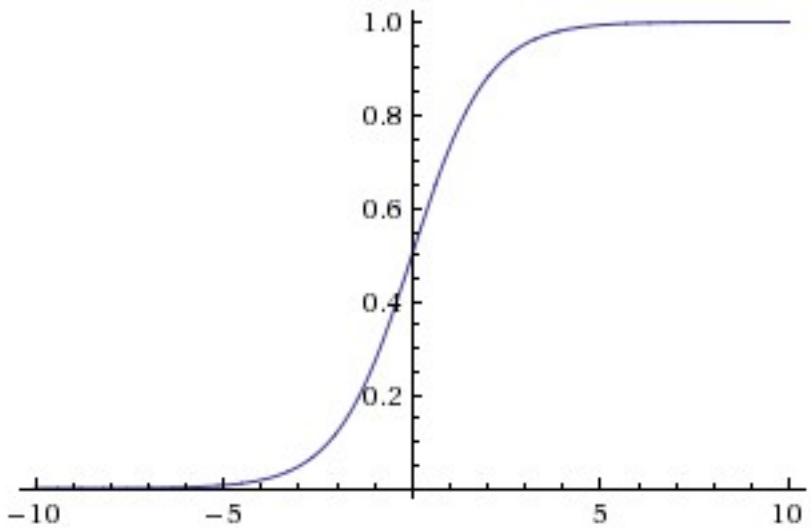
   $$\frac{d}{dx}ReLU(x) = \begin{cases} 0, \text{ if } x < 0, \\ 1, otherwise. \end{cases}$$

   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. *Hàm sigmoid* là hàm phi tuyến tính có đồ thị một dạng đường cong hình dạng chữ "S" (Hình 2.5) và có công thức toán học là $f(x) = \frac{1}{1+e^x}$. Sự tăng trưởng của đồ thị gồm 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.



Hình 2.4: Hàm ReLU



Hình 2.5: Hàm sigmoid

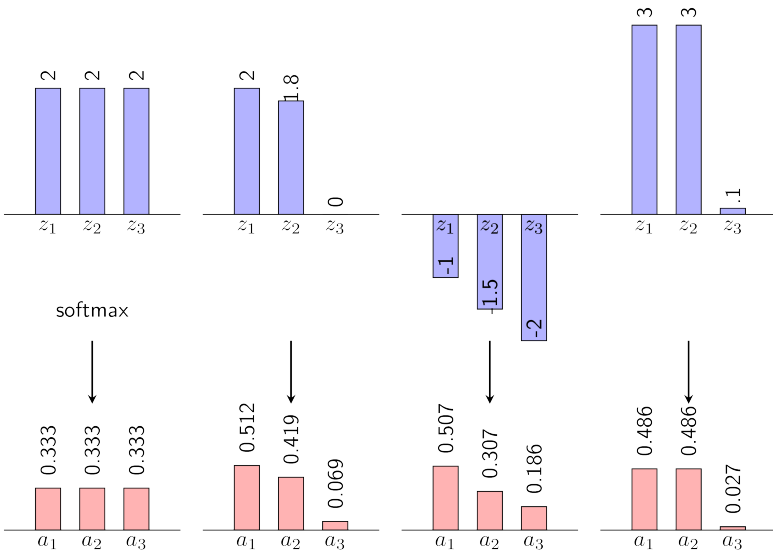
Giá trị của hàm số luôn nằm trong khoảng (0,1], cụ thể hơn là với đầu vào lớn hàm số sẽ cho đầu ra gần với 1 còn với đầu vào nhỏ hàm số sẽ cho đầu ra gần với 0. Đạo hàm của hàm sigmoid này khá là đẹp $f'(x) = f(x)(1 - f(x))$.

3. *Hàm softmax* hay hàm trung bình mũ là sự khái quát hóa của hàm sigmoid 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:

$$y_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}; \forall i = 1, ..., n, \text{ với } n \text{ là số lượng phần tử trong vector } \mathbf{x}$$

Hàm softmax có công dụng trong việc phân loại tập dữ liệu. 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. Do tổng giá trị đầu ra của hàm softmax luôn bằng một, luôn dương và mỗi đầu ra đều phụ thuộc vào tất cả các đầu vào nên ta có thể coi giá trị đầu ra thể hiện xác suất của dữ liệu rơi vào từng tập dữ liệu tương ứng.

Ví dụ dữ liệu đầu vào là vector **z**, kết quả giá trị đầu ra tương ứng là vector **a** (Hình 2.6). Chúng ta có thể thấy các giá trị a_1, a_2, a_3 thể hiện xác suất của dữ liệu **z** rơi vào.

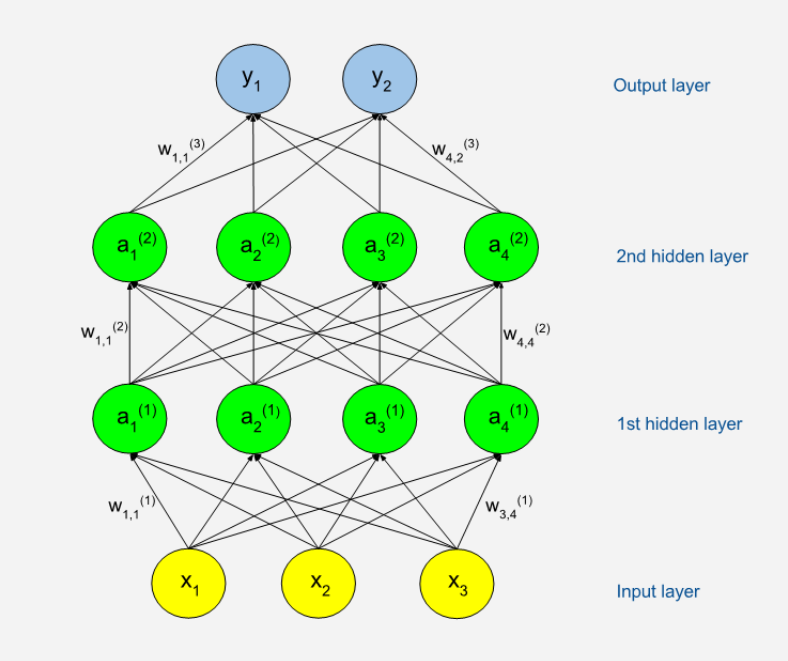


Hình 2.6: Ví dụ hàm softmax (nguồn: machinelearningcoban.com)

2.3 Mạng nơ-ron nhiều lớp

2.3.1 Kiến trúc chung

Mạng nơ-ron là sự kết hợp của các lớp perceptron hay còn gọi là perceptron nhiều lớp (multilayer perceptron) sẽ có nhiều lớp liên kết với nhau và được chia làm 3 lớp chính: lớp đầu vào (*input layer*), lớp ẩn (*hidden layer*), lớp đầu ra (*output layer*). Mỗi lớp sẽ có một số lượng perceptron khác nhau, không có quy định chung và tùy theo người thiết kế mạng. Kiến trúc chung của mạng nơ-ron được minh họa trong Hình 2.7.



Hình 2.7: Cấu trúc chung của mạng nơ-ron

- Lớp đầu vào hay còn gọi là *input layer*: Biểu diễn tổng quát của mỗi quan sát

- Lớp đầu ra hay còn gọi là *output layer*: Thể hiện đầu ra dự đoán của model

- Lớp ẩn hay còn gọi là *hidden layer*: Là lớp thể hiện cấu trúc, suy luận logic của mạng nơ-ron. Hidden layer là các lớp nằm giữa lớp đầu ra và lớp đầu vào.

Số lượng layer trong một mạng nơ-ron được ký hiệu là l và được tính bằng số hidden layer cộng thêm một, ví như mạng nơ-ron trong Hình 2.7 là 3. Người ta gọi mỗi một nơ-ron trong mạng là một *unit*.

2.3.2 Mạng lan truyền thẳng

2.3.2.1 Mô tả

Một mạng lan truyền thẳng (*feed forward*) nhiều lớp bao gồm một lớp vào, một lớp ra và một hoặc nhiều lớp ẩn. Các nơron đầu vào thực chất không phải các nơron theo đúng nghĩa, bởi lẽ chúng không thực hiện bất kỳ một tính toán nào trên dữ liệu vào, đơn giản nó chỉ tiếp nhận các dữ liệu vào và chuyển cho các lớp kế tiếp. Các nơron ở lớp ẩn và lớp ra mới thực sự thực hiện các tính toán, kết quả được định dạng bởi hàm đầu ra. Cụm từ "truyền thẳng" (feed forward) (không phải là trái nghĩa của lan truyền ngược) liên quan đến một thực tế là tất cả các nơ-ron chỉ có thể được kết nối với nhau theo một hướng: tới một hay nhiều các nơron khác trong lớp kế tiếp (loại trừ các nơron ở lớp ra). Cụ thể hơn là không có các liên kết từ các nơ-ron ở lớp đầu ra đến các nơ-ron ở các lớp đầu và hay các nơ-ron trong cùng một lớp cũng không có liên kết với nhau. Các lớp lớp liên kết với nhau theo quy luật đầu vào lớp sau là đầu ra của lớp trước.

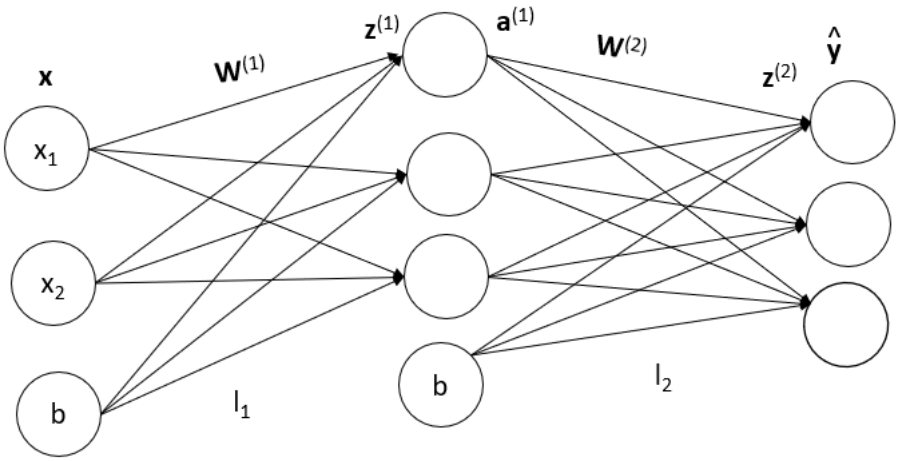
2.3.2.2 Thuật toán

Algorithm 4 Forward propagation

Require: Network depth, l
Require: $\mathbf{W}^{(i)}, i \in \{1, \ldots, l\}$, ma trận trọng số của model
Require: $\mathbf{b}^{(i)}, i \in \{1, \ldots, l\}$tham số bias 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 $j = 1, \ldots, N$ **do**
    $\mathbf{a}^{(0)} = \mathbf{x}_j$
    for $i = 1, \ldots, l$ **do**
      $\mathbf{z}^{(i)} = \mathbf{b}^{(i)} + \mathbf{W}^{(i)}\mathbf{a}^{(i-1)}$
      $\mathbf{a}^{(i)} = f(\mathbf{z}^{(i)})$
    end for
    $\widehat{\mathbf{y}} = \mathbf{a}^{(L)}$
  end for

2.3.2.3 Ví dụ

Giả sử ta có mạng neural như Hình 2.8 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 2.8: 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:

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

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

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

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

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

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

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* 2.4 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 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*.

2.3.3 Hàm mất mát

Hàm mất mát hay còn gọi là *loss function* hoặc *cost function*, là sự chêch lệch, khác biệt giữa đầu ra dự đoán và đầu ra thực tế bằng một số thực không âm, có chức năng là đo độ chính xác của đầu ra dự đoán, thường được ký hiệu là J. Ví dụ ta cần ánh xạ: $\mathbf{x} \rightarrow \mathbf{y}$, trong đó $\mathbf{x}$ là tập các dữ liệu và $\mathbf{y}$ là tập các nhãn tương ứng cho từng dữ liệu thì ta cần *ước lượng các tham số* $\mathbf{w}$ sao cho thỏa mãn $f(\mathbf{x}, \mathbf{w}) \approx \mathbf{y}$, giá trị của hàm f được kí hiệu là $\widehat{\mathbf{y}}$. Khi đó 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 lại các tham số sao cho tập giá trị của $\widehat{\mathbf{y}}$ càng gần $\mathbf{y}$ càng tốt. Có nghĩa rằng ta cần tìm $\mathbf{w}$ sao cho giá trị mất mát là nhỏ nhất: $\mathbf{w}$=$\arg\min_{\mathbf{w}} J$ hay nói cách khác là tìm $\mathbf{w}$ sao cho tỉ lệ rơi vào đúng tập nhãn là cao nhất: $\mathbf{w}$=$\arg\max_{\mathbf{w}} p(\mathbf{y}|\mathbf{x}, \mathbf{w})$. Có nhiều hàm có thể làm hàm mất mát như: sai số bình phương trung bình, perceptron, hinge,cross-entropy,...

2.3.3.1 Hàm sai số bình phương trung bình

Hàm sai số toàn phương trung bình hay mean squared error (MSE), là hàm tiếp cận dễ nhất để hiểu hàm mất mát. Nó được định nghĩa là trung bình của bình phương các sai số giữa giá trị ước lượng được và thực tế.

Giả sử $\widehat{\mathbf{y}}$ là một vector đầu ra dự đoán, và $\mathbf{y}$ là vector đầu ra thực tế quan sát được, tương ứng với các dữ liệu đầu vào, thì MSE của phép dự báo có thể ước lượng theo công thức:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{y}_i - \widehat{\mathbf{y}})^2$$

2.3.3.2 Hàm cross-entropy

1. *Cross entropy*
   Cross entropy được dùng để đo sự giống nhau của hai phân phối xác suất, giá trị của hàm số càng nhỏ thì hai xác suất càng gần nhau. Cho một phân bố xác suất $\mathbf{p} = [p_1, p_2, \ldots, p_n]$ với $q_i \in [0, 1]$ và $\sum_{i=1}^{n} q_i = 1$. Nếu ta có một phân bố xác suất bất kỳ $\mathbf{q} = [q_1, q_2, \ldots, q_n]$ và $q_i \neq 0, \forall i$ thì hàm số cross entropy giữa hai phân bố $\mathbf{p}$ và $\mathbf{q}$ được định nghĩa là:

   $$H(\mathbf{p}, \mathbf{q}) = -\sum_{i=1}^{C} p_i \log q_i \tag{2.7}$$

   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 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 xác suất nào bằng 0 tuyệt đối.

2. *One-hot encoding*
   One-hot encoding là mã hóa dữ liệu thành các vector ở dạng phân loại tức là chỉ có duy nhất một phần tử bằng 1 còn các phần tử khác bằng 0. Ví dụ ta có tập dữ liệu $y = [1, 2, 4, 1, 1]$ và cần phân loại ra 4 lớp là 1, 2, 3, 4, khi đó mã hóa các dữ liệu về dạng one-hot ta thu được tập dữ liệu tương ứng $y = [1000, 0100, 0001, 1000, 1000]$.

3. *Ví dụ*

Tại đây tôi sẽ tiếp tục với ví dụ ở phần mạng lan truyền thẳng, các đầu ra đều được chuyển về dạng one-hot và lấy hàm cross-entropy làm hàm mất mát. Ta viết được hàm mất mát cho dữ liệu thứ i như sau:

$$\begin{aligned} J_i(\mathbf{x}_i, \mathbf{y}_i) = J_i &= -\sum_{j=1}^{C} y_{ij} \log \widehat{y}_{ij} \\ &= -\sum_{j=1}^{C} y_{ij} \log \frac{e^{z_{ij}^{(l)}}}{\sum_{k=1}^{C} e^{z_{ik}^{(l)}}} \\ &= -\sum_{j=1}^{C} \left(y_{ij} z_{ij}^{(l)} - \log(\sum_{k=1}^{C} e^{z_{ik}^{(l)}}) \right) \\ &= -\sum_{j=1}^{C} y_{ij} z_{ij}^{(l)} + \sum_{j=1}^{C} y_{ji} \log(\sum_{k=1}^{C} e^{z_{ik}^{(l)}}) \end{aligned}$$

vì đầu ra thực tế ở dạng onehot nên ta có $\sum_{j=1}^{C} y_{ij=1}$ mà $\log(\sum_{k=1}^{C} e^{z_{ik}^{(l)}})$ là hằng số nên ta có $\sum_{j=1}^{C} y_{ji} \log(\sum_{k=1}^{C} e^{z_{ik}^{(l)}}) = \log(\sum_{k=1}^{C} e^{z_{ik}^{(l)}})$. Hàm mất mát được rút gọn lại như sau:

$$J_i(\mathbf{x}_i, \mathbf{y}_i) = -\sum_{j=1}^{C} y_{ij} z_{ij}^{(l)} + \log(\sum_{k=1}^{C} e^{z_{ik}^{(l)}}) \tag{2.8}$$

Như vậy đốivới tập dữ liệu $(\mathbf{x}_i, y_i)$, $i = 1, 2, ..., N$ thì hàm mất mát cho *softmax* được tính theo biểu thức sau đây:

$$J(\mathbf{X}, \mathbf{Y}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} \log \widehat{y}_{ij} = -\frac{1}{N} \sum_{i=1}^{N} \left(\sum_{j=1}^{C} y_{ij} z_{ij}^{(l)} + \log \sum_{k=1}^{C} (e^{z_{ik}^{(l)}}) \right) \tag{2.9}$$

2.3.4 Thuật toán lan truyền ngược và cập nhật tham số

Trong tiếng Anh, Lan 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. Phương pháp này tính toán gradient của hàm tổn thất với tất cả các trọng số có liên quan trong mạng nơ ron đó. Gradient này được đưa vào phương pháp tối ưu hóa, sử dụng nó để cập nhật các trọng số, để cực tiểu hóa hàm tổn thất.

2.3.4.1 Mô tả thuật toán

Truyền ngược yêu cầu một đầu ra mong muốn, đã biết cho mỗi giá trị đầu vào để tính toán gradient hàm tổn thất. Do đó, nó thường được xem là một phương pháp học có giám sát, tuy nhiên nó cũng được sử dụng trong một số mạng không có giám sát. Nó là tổng quát hóa của gradient descent cho các mạng lan truyền thẳng nhiều lớp, thực hiện bằng cách sử dụng quy tắc chain rule để tính toán lặp đi lặp lại các gradient cho mỗi lớp. Thuật toán lan truyền ngược yêu cầu các hàm kích hoạt được sử dụng bởi các nơ-ron nhân tạo khả vi.

2.3.4.2 Thuật toán

Algorithm 5 Backpropagation

Require: Network depth, l
Require: $\mathbf{W}^{(i)}, i \in \{1, \ldots, l\}$, ma trận trọng số của model
Require: $\mathbf{b}^{(i)}, i \in \{1, \ldots, l\}$tham số bias của model
Require: $\mathbf{x}_i, i \in \{1, \ldots, N\}$, tập dữ liệu đầu vào
Require: $\mathbf{y}_i, i \in \{1, \ldots, N\}$, đầu ra thực tế
Require: $\widehat{\mathbf{y}}$, đầu ra dự đoán
Require: $\mathbf{z}, \mathbf{a}$, đầu ra của hàm kết hợp và hàm kích hoạt
Require: J, giá trị của hàm mất mát
Sau khi thực hiện thuật toán lan truyền thẳng, tính gradient ở lớp đầu ra:
$\mathbf{g} \leftarrow \dfrac{\partial J}{\partial \widehat{y}}$
for $k = 1, \ldots, l$ **do**
Từ lớp đầu ra, tính gradient quay lui với hàm kích hoạt ở lớp thứ k theo chain rule
$\mathbf{g} \leftarrow \dfrac{\partial J}{\partial \mathbf{z}^{(k)}} = \mathbf{g} \odot f'(\mathbf{z}^{(k)})$
Tính gradient so với trọng số, ngưỡng
$\dfrac{\partial J}{\partial \mathbf{b}^{(k)}} = \mathbf{g}$
$\dfrac{\partial J}{\partial \mathbf{W}^{(k)}} = \mathbf{g}\mathbf{a}^{(k-1)T}$
Gradient được đưa xuống lớp đầu ra thấp hơn
$\mathbf{g} \leftarrow \dfrac{\partial J}{\partial \mathbf{a}^{(k-1)}} = \mathbf{W}^{(k)T}\mathbf{g}$
end for

2.3.4.3 Ví dụ

Ta sử dụng mạng nơ-ron ở Hình 2.8, xét một cặp dữ liệu thứ i ta có $(\mathbf{x}_i, \mathbf{y}_i)$ với $\mathbf{x}_i$ là đầu vào và $\mathbf{y}_i$ là đầu ra thực tế ở dạng *one-hot*. Để thực hiện được thuật toán chúng ta cần tìm đầu ra dự đoán thông qua lan truyền thẳng (refer), tính giá trị mất mát (refer). Tôi xin được viết lại như sau:

$$\mathbf{a}_i^{(0)} = \mathbf{x}_i$$

$$\mathbf{z}_i^{(1)} = \mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{a}_i^{(0)}$$

$$\mathbf{a}_i^{(1)} = \max(0, \mathbf{z}_i^{(1)})$$

$$\mathbf{z}_i^{(2)} = \mathbf{b}^{(2)} + \mathbf{W}^{(2)}\mathbf{a}_i^{(1)}$$

$$\mathbf{a}_i^{(2)} = \widehat{\mathbf{y}}_i = softmax(\mathbf{z}_i^{(2)}) = \frac{\exp(\mathbf{z}_i^{(2)})}{\sum_{j=1}^{3} \exp(\mathbf{z}_j^{(2)})}$$

và hàm mất mát: $J_i = -\sum_{j=1}^{3} y_{ij} \log \widehat{y}_{ij}$
Vì chỉ xét trên một cặp dữ liệu và để dễ nhìn hơn, ta lược bỏ chỉ số i ở các biểu thức tính toán

dưới đây. Theo như thuật toán lan truyền ngược, trước tiên ta cần tính đạo hàm của đầu ra dự đoán đối với hàm mất mát. Xét đầu ra dự đoán thứ k của tập dữ liệu thứ i.

$$g_k^{(l)} = \frac{\partial J}{\partial \widehat{y}_k} = -\sum_{j=1}^{3} y_j \frac{\partial \log \widehat{y}_j}{\widehat{y}_k} = -\sum_{j=1}^{3} y_j \frac{1}{\widehat{y}_k}$$

Tiếp theo ta đưa đạo hàm về layer cuối cùng, tương đương với $l = 2$, ta có $\mathbf{g}^{(l)} = \mathbf{g}^{(l)} \odot f'(\mathbf{a}^{(l)})$, với hàm $f(\mathbf{z}^{(2)})$ là hàm softmax. Để biểu thức sau này của chúng ta bớt cồng kềnh, tôi sẽ tính riêng đạo hàm của hàm softmax. Đối với hàm này chúng ta cần lưu ý rằng nó có hai trường hợp là đạo hàm phần tử trùng với tử số hoặc khác tử số.

Giả sử hàm softmax chúng ta có như sau: $p_i = \dfrac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}$ và ta cần tính đạo hàm của p_i đối với a_j bất kì, tương đương với biểu thức $\dfrac{\partial p_i}{\partial a_j} = \dfrac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j}$

Với $i = j$,

$$\begin{aligned}
\frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j} &= \frac{e^{a_i} \sum_{k=1}^{N} e^{a_k} - e^{a_j} e^{a_i}}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2} \\
&= \frac{e^{a_i} \left(\sum_{k=1}^{N} e^{a_k} - e^{a_j}\right)}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2} \\
&= \frac{e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} \times \frac{\left(\sum_{k=1}^{N} e^{a_k} - e^{a_j}\right)}{\sum_{k=1}^{N} e^{a_k}} \\
&= p_i(1 - p_j)
\end{aligned} \tag{2.10}$$

với $i \neq j$,

$$\begin{aligned}
\frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j} &= \frac{0 - e^{a_j} e^{a_i}}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2} \\
&= \frac{-e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} \times \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}} \\
&= -p_j . p_i
\end{aligned} \tag{2.11}$$

Từ biểu thức 2.10 và 2.11 ta có thể viết lại đạo hàm của softmax :

$$\frac{\partial p_i}{\partial a_j} = \begin{cases} p_i(1 - p_j) & if \quad i = j \\ -p_j . p_i & if \quad i \neq j \end{cases} \tag{2.12}$$

Từ đó ta tính được đạo hàm như sau:

$$\begin{aligned}
g_k^{(2)} &= -y_k \frac{1}{\widehat{y}_k} f(z_k)(1 - f(z_k)) - \sum_{\substack{j=1,\\ j\neq k}}^{3} y_j \frac{1}{\widehat{y}_k}(-f(z_k)f(z_j)) \\
&= -y_k \frac{1}{\widehat{y}_k} \widehat{y}_k(1 - \widehat{y}_k) - \sum_{\substack{j=1,\\ j\neq k}}^{3} y_j \frac{1}{\widehat{y}_k}(-\widehat{y}_k \widehat{y}_j) \\
&= -y_k(1 - \widehat{y}_k) + \sum_{\substack{j=1,\\ j\neq k}}^{3} y_j \widehat{y}_j \\
&= -y_k + y_k \widehat{y}_k + \sum_{\substack{j=1,\\ j\neq k}}^{3} y_j \widehat{y}_j \\
&= \widehat{y}_k(y_k + \sum_{\substack{j=1,\\ j\neq k}}^{3} y_j) - y_k
\end{aligned} \tag{2.13}$$

Do đầu ra thực tế $\mathbf{y}$ ở dạng onehot vì thế $\sum_{j=1}^{3} y_j = 1$ và $y_k + \sum_{\substack{j=1,\\ j\neq k}}^{3} y_j = 1$. Vì thế ta được

$$g_k^{(2)} = \widehat{y}_k - y_k$$

Nếu ta sử dụng biểu thức 2.8 thì đạo hàm $\dfrac{\partial J}{\partial z_k^{(l)}}$ đơn giản hơn khá nhiều

$$\begin{aligned}
\frac{\partial J}{\partial z_k} &= \frac{-\sum_{j=1}^{C} y_{kj} z_{kj}^{(l)} + \log(\sum_{m=1}^{C} e^{z_m^{(l)}})}{\partial z_k^{(l)}} \\
&= -y_k + \frac{e_k^{(z^{(l)})}}{\sum_{m=1}^{C} e^{(z_m^{(l)})})} \\
&= -y_k + \widehat{y}_k = \widehat{y}_k - y_k
\end{aligned}$$

Tổng quát hóa với cặp dữ liệu thứ i ta được

$$\mathbf{g}_i^{(2)} = \frac{\partial J}{\partial \mathbf{z}_i^{(2)}} = \widehat{\mathbf{y}}_i - \mathbf{y}_i \tag{2.14}$$

Như vậy chúng ta có

$$\frac{\partial J}{\partial \mathbf{b}^{(2)}} = \mathbf{g}_i^{(2)} \tag{2.15}$$

$$\frac{\partial J}{\partial \mathbf{W}^{(2)}} = \mathbf{g}_i^{(2)} \mathbf{a}^{(1)T} \tag{2.16}$$

$$\mathbf{g}_i^{(2)} = \mathbf{W}^{(1)T} \mathbf{g}_i^{(2)} \tag{2.17}$$

$$\tag{2.18}$$

Tiếp theo chúng ta tính đạo hàm ở layer $l-1$ tương đương với layer đầu tiên. Layer này khá là đơn giản vì hàm activation là hàm Relu và có đạo hàm là

$$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$$

Từ đó ta tính được

$$\begin{aligned} \mathbf{g}_i^{(1)} &= \mathbf{g}_i^{(1)} \odot f'(\mathbf{z}^{(1)}) \\ \frac{\partial J}{\partial b^{(1)}} &= \mathbf{g}_i^{(1)} \\ \frac{\partial J}{\partial \mathbf{W}^{(1)}} &= \mathbf{g}_i^{(1)} \mathbf{a}^{(0)} \end{aligned}$$

2.3.5 Overfitting

2.3.5.1 Khái niệm

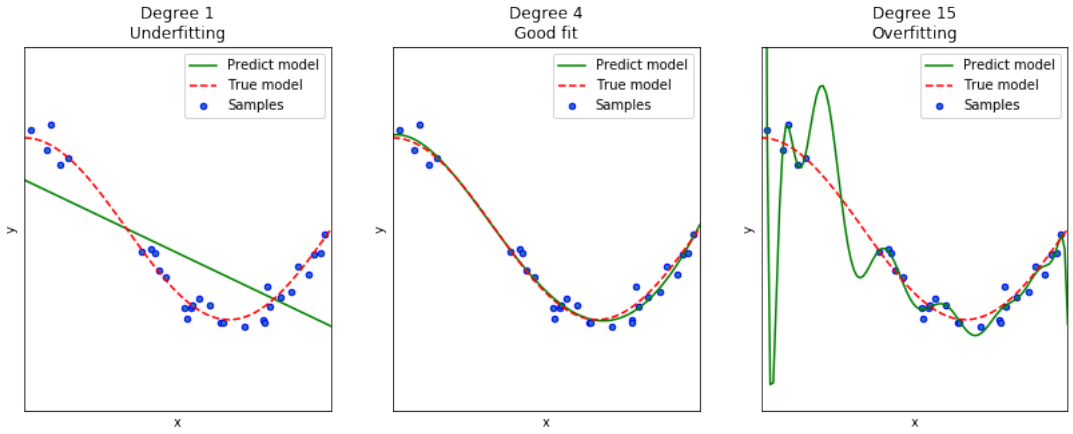
Overfitting là một hiệnn tượng không mong muốn thường gặp trong quá trình xây dựng mô hình, người xây dựng mô hình cần nắm được một số kỹ thuật để giải quyết khi vấn đề này xảy ra. Trước tiên chúng ta cần biết ba khái niệm dưới đây:

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.

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.*

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.

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.

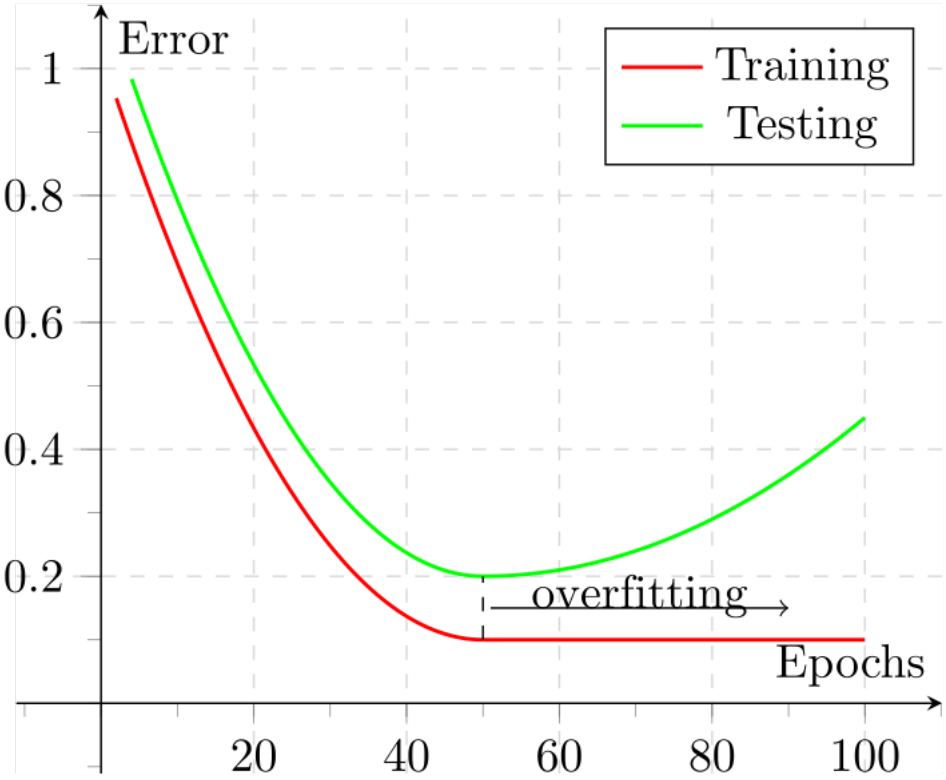
Ở hình thứ nhất chúng ta có thể thấy mô hình dự đoán là một hàm tuyến tính (bậc bằng 1) rất khác với mô hình thực, xa với các điểm dữ liệu. Hiện tượng này ta nói mô hình bị *underfitting*. Với mô hình dự đoán là đa thức bậc 4 chúng ta có thể thấy mô hình dự đoán xấp xỉ như mô hình thực (hình thứ 2). 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*. Để tránh vấn đề này xảy ra chúng ta có phương pháp khá là hữu dụng đó là *regularization*.

2.3.5.2 Một số phương pháp ngăn chặn overfitting

Nhìn chung các phương pháp làm thay đổi nhỏ các tham số hay kiến trúc mô hình, chấp nhận hy sinh độ chính xác trong quá trình huấn luyện để giảm độ phức tạp của mô hình giúp tránh được hiện tượng overfitting mà vẫn giữ được tính tổng quát của mô hình.

kỹ thuật regulazation

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.



Hình 2.10: Early stopping

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:

$$J_{reg}(\mathbf{W}) = J(\mathbf{W}) + \lambda R(\mathbf{W}) \tag{2.19}$$

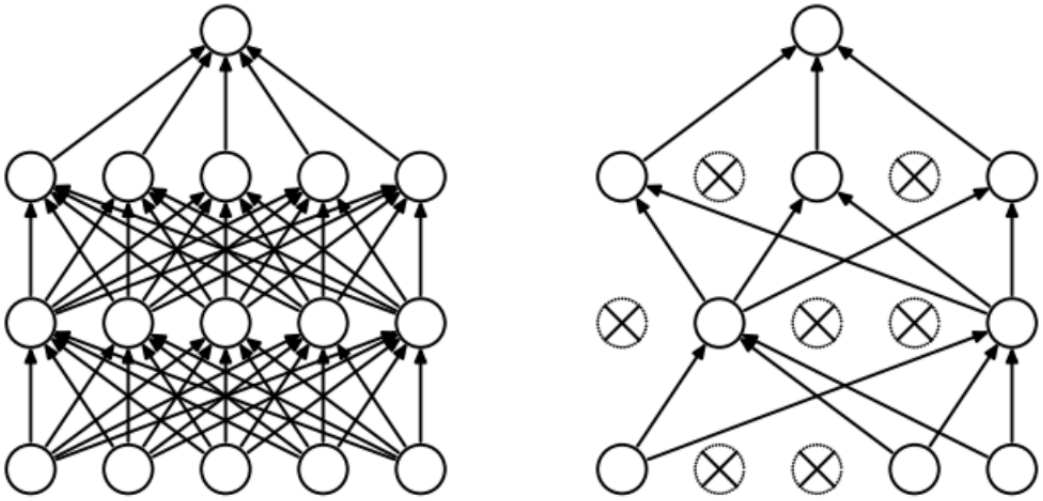
   $J(\mathbf{W})$ là hàm mất mát ban đầu và cụm $\lambda R(\mathbf{W})$ mới thêm vào là số hạng chính quy hoá (hay số hạng regularization) đóng vai trò như một biện pháp phạt lỗi (penalization).Trong đó, tham số chính quy hoá (*regularizaton parameter*) λ được chọn từ trước để cân bằng giữa $J(\mathbf{W})$ 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:

$$R(\mathbf{W}) = \frac{1}{p}\|\mathbf{W}\|_p^p = \frac{1}{p}\sum_i^n |\mathbf{W}|^p \tag{2.20}$$

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

   Phương pháp chính quy hoá này còn có tên là phân rã 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.

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 **p** cho trước. Các mạng mới sau khi áp dụng dropout được gọi là subsample. Thông thường xác suất ở layer input bằng 0.8 hay ta loại bỏ khoảng 20% số unit, ở hidden layer thì xác 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

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 xác 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 xác 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*.

2.3.6 Dữ liệu

Dữ liệu thường được chia làm ba phần: tập huấn luyện, tập đánh giá, tập kiểm tra. Tỉ lệ thường được lấy của các tập dữ liệu: tập kiểm tra bằng 20% tổng dữ liệu, tập huấn luyện bằng 80% tổng dữ liệu, tập đánh giá bằng 20% tập huấn luyện. Tuy nhiên vì tỉ lệ được so với số lượng dữ liệu ta thu thập được nên không phải lúc nào cũng giống nhau. Nếu thu thập được nhiều dữ liệu thì ta có thể tăng tỉ lệ tập huấn luyện và giảm với tập kiểm tra, đánh giá. Một số vai trò chính của các tập:

- Tập huấn luyện đóng vai trò làm đầu vào cho mạng nơ-ron giúp tìm ra được các trọng số phù hợp với dữ liệu cần đánh giá.
- Tập đánh giá sẽ được chạy trong quá trình huấn luyện, sau mỗi epoch và giúp đánh giá mô hình đã tốt chưa, liệu có cần thay đổi giá trị của các tham số.
- Tập kiểm tra cũng giống như tập đánh giá, cho ta biết độ chính xác của mô hình đối với các dữ liệu không có trong tập huấn luyện nhưng được chạy khi mô hình đã được huấn luyện.

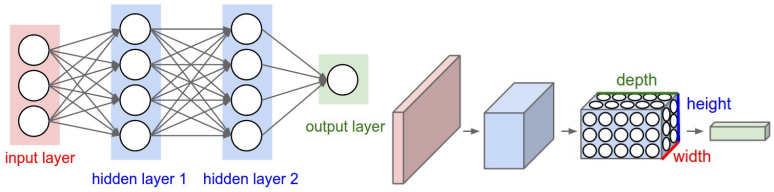
Dữ liệu cần được chuẩn hóa trước khi đưa vào mạng huấn luyện, ví dụ như nên đưa dữ liệu về khoảng [0,1] để tiện cho việc tính toán và các feature không bị chênh lệch nhau hay nếu dữ liệu là ảnh thì cần đưa về cùng kích thước.

2.4 Mạng nơ-ron tích chập

2.4.1 Tổng quan

Mạng nơ-ron tích chập, có tên tiếng anh Convolutional Neural Network và được viết tắt là CNN, là một trong những mô hình học sâu tiên tiến giúp xây dựng được những hệ thống thông minh với độ chính xác cao. Trong khóa luận này, tôi sẽ trình bày về tổng quan về mạng nơ-ron tích chập và ứng dụng CNN cho bài toán phân loại biển báo giao thông.

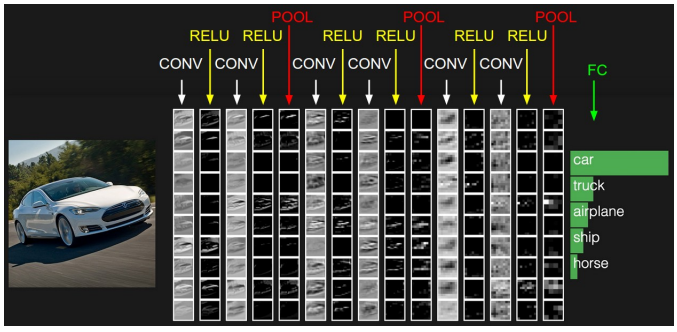
Mạng nơ-ron tích chập lấy cảm hứng từ xử lý ảnh nên đầu vào và các lớp của CNNs có dạng như một bức ảnh, là các ma trận chứ không có dạng vector như mạng nơ-ron thông thường. Cụ thể, một bức ảnh sau khi số hoá có dạng *dài* × *rộng* × *sâu* (dài: số lượng điểm ảnh trên chiều rộng, rộng : số lượng điểm 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 2.12: Kiến trúc hai mạng

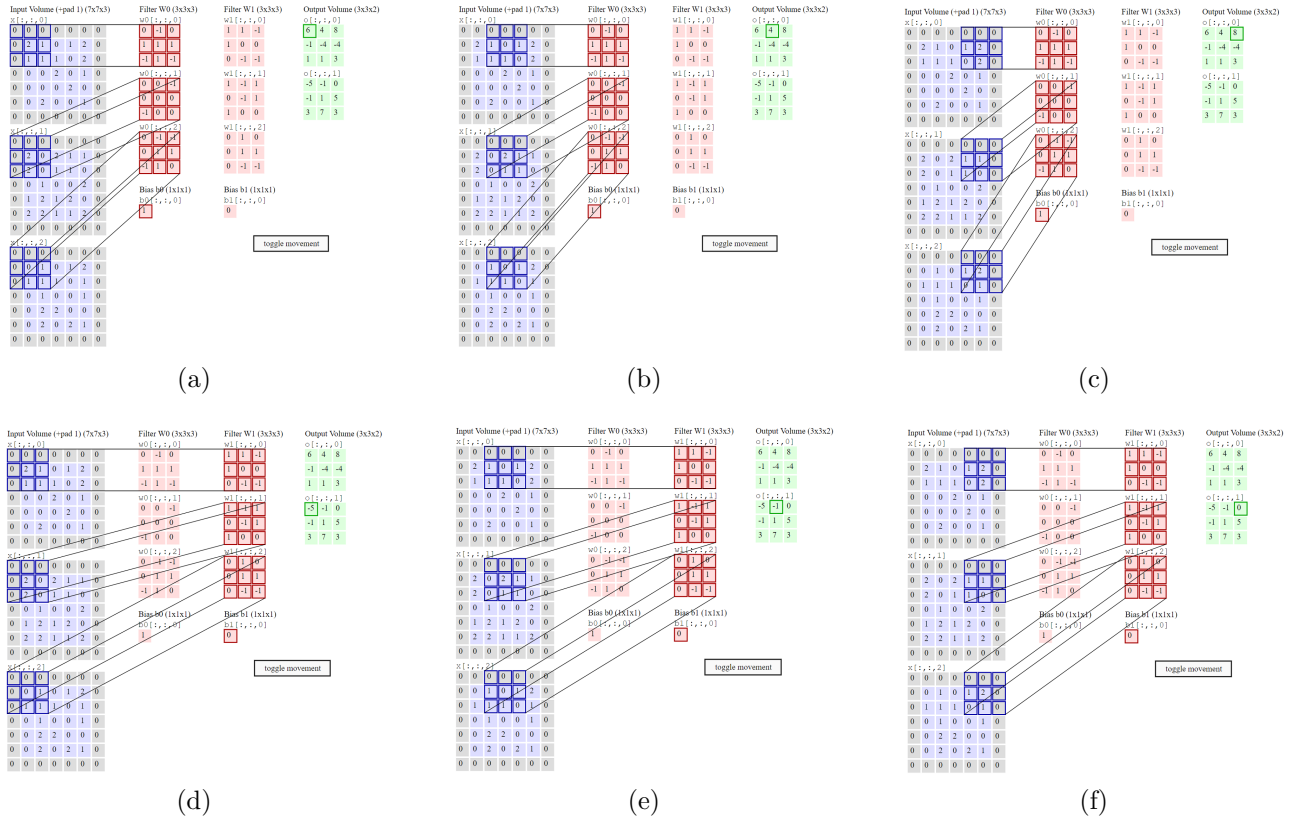
Một mạng CNNs đơn giản có ba lớp chính: lớp tích chập (*convolutional layer*), lớp giảm số chiều (*pooling layer*), lớp fully-connected. Các lớp convolutional và lớp pooling được xếp xen kẽ nhau Hình 2.13.



Hình 2.13: ví dụ về kiến trúc của ConvNets

2.4.2 Lớp tích chập - Convolutional layer

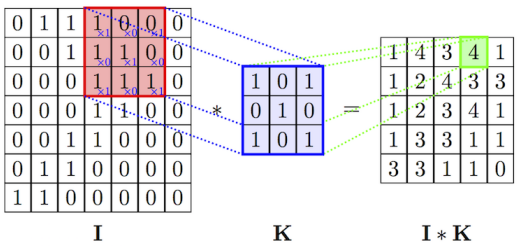
Convolution layer là khối cốt lõi, cơ bản của ConvNets, lớp này chủ yếu nặng về việc tính toán. Lớp này chính là nơi thể hiện tư tưởng ban đầu của mạng nơron tích chập. Thay vì kết nối toàn bộ điểm ảnh, lớp này sẽ sử dụng *bộ lọc (filter)* áp vào một vùng trong ảnh và tiến hành tính element-wise giữa bộ lọc và *vùng tiếp nhận cục bộ (local receptive filter)*. Sau đó ta dùng phương pháp *cửa sổ trượt (sliding window)* để bộ lọc quét hết các vùng trong ảnh.



Hình 2.14: Tính toán trong lớp tích chập

2.4.2.1 Bộ lọc

Bộ lọc hay còn gọi là filter hoặc kennel, là các ma trận có kích thước nhỏ $(3 \times 3, 5 \times 5, 7, \ldots)$ giúp ta thay đổi các giá trị đầu vào dựa vào các giá trị lân cận theo một nguyên tắc, công thức nào đó. Tùy theo đầu vào của bài toán thì kích thước của bộ lọc được thay đổi để phù hợp với hình dạng của đầu vào. Chúng ta sẽ rõ hơn khi xem hình 2.15, ta có ma trận I là ma trận đầu vào, ma trận K là bộ lọc, ma trận I*K là ma trận kết quả với mỗi phần tử là kết quả của phép tính element-wise giữa bộ lọc và vùng re

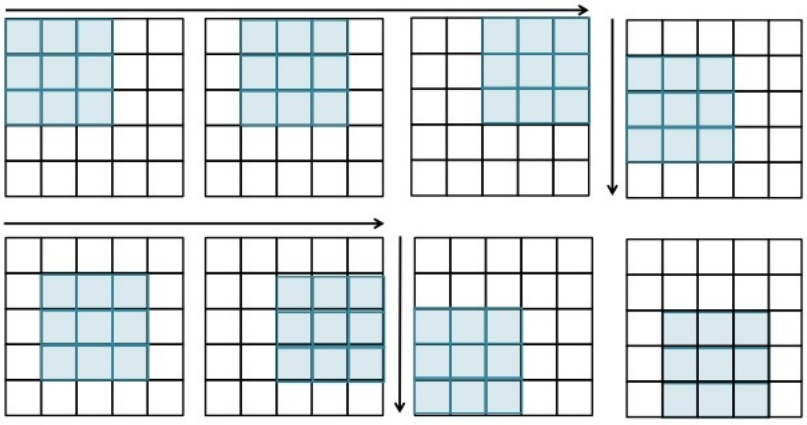


Hình 2.15: Bộ lọc - filter

Nhưng khi với đầu vào giống như Hình 2.14 thì kích thước bộ lọc là ma trận 3 chiều. 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.

2.4.2.2 Cửa sổ trượt

Cửa sổ trượt hay còn có tên tiếng anh là 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 2.16, 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 2.16: Sliding window

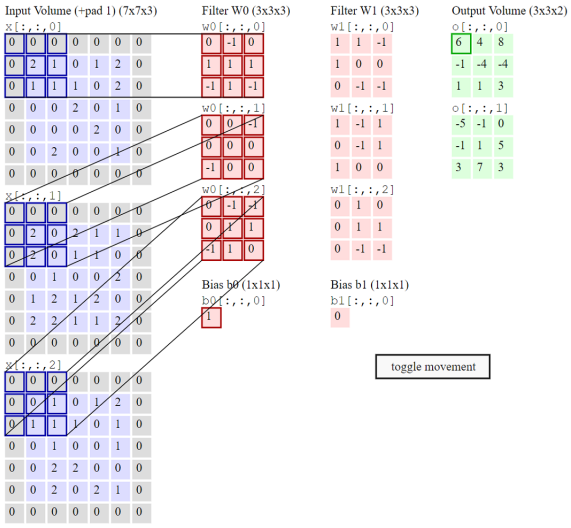
2.4.2.3 Cách thức hoạt động

Theo Hình 2.14 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 2.14a 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 2.14a, 2.14b, 2.14c mô tả giai đoạn này. Và tương tự như trên ta thực hiện với $filter\ W1$ theo Hình 2.14d, 2.14e, 2.14f 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 2.17: Ví dụ về tính toán kích thước đầu ra

Giả sử ta có thông số như hình 2.17 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 $3 \times 3 \times 2$.

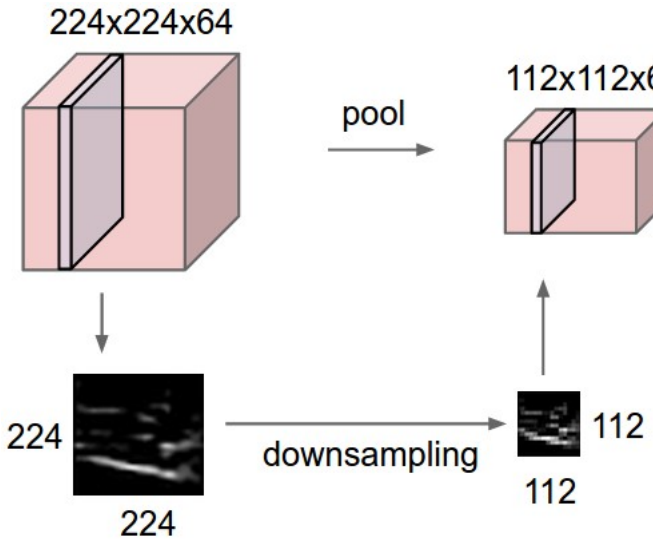
Một phần quan trọng của layer này đó là hàm activation, hàm này được tính với đầu vào là kết quả sau khi thực hiện tích chập. Và ReLU thường được chọn là hàm activation do tính đơn giản của nó. Nhiệm vụ của nó là chuyển toàn bộ giá trị âm trong kết quả sau khi tích chập thành 0. Ý nghĩa của cách cài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

2.4.3 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.

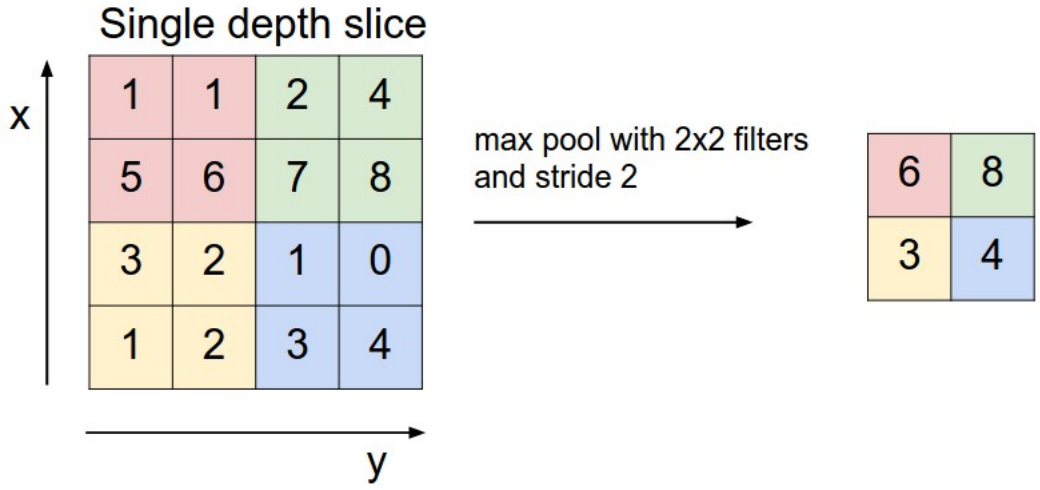
Cách thức hoạt động: Pooling layer sử dụng sliding window và tại mỗi cửa sổ trượt trên đầu vào chỉ có một giá trị được xem là giá trị đại diện cho thông tin ảnh tại vùng đó (giá trị

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 2.18 mô tả tiến hành lấy mẫu bằng max pooling.



Hình 2.18: Minh họa giảm số chiều

Ví dụ ta có ma trận đầu vào 4×4 như hình 2.19, 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 2.19: 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:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$

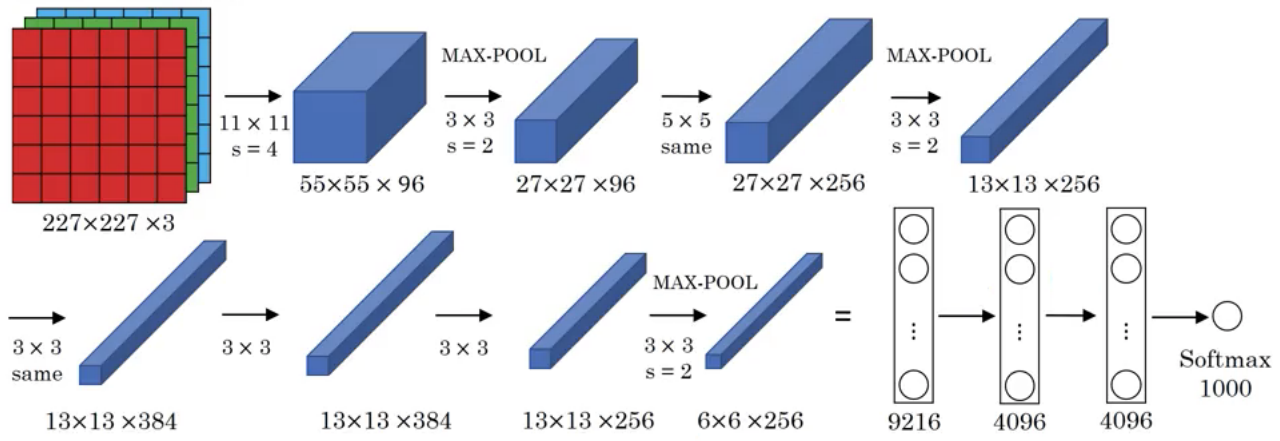
Chú ý rằng chúng ta không thường xuyên sử dụng *zero padding* cho lớp này.

2.4.4 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.

2.5 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 2.20.



Hình 2.20: Ví dụ cấu trúc mạng CNNs

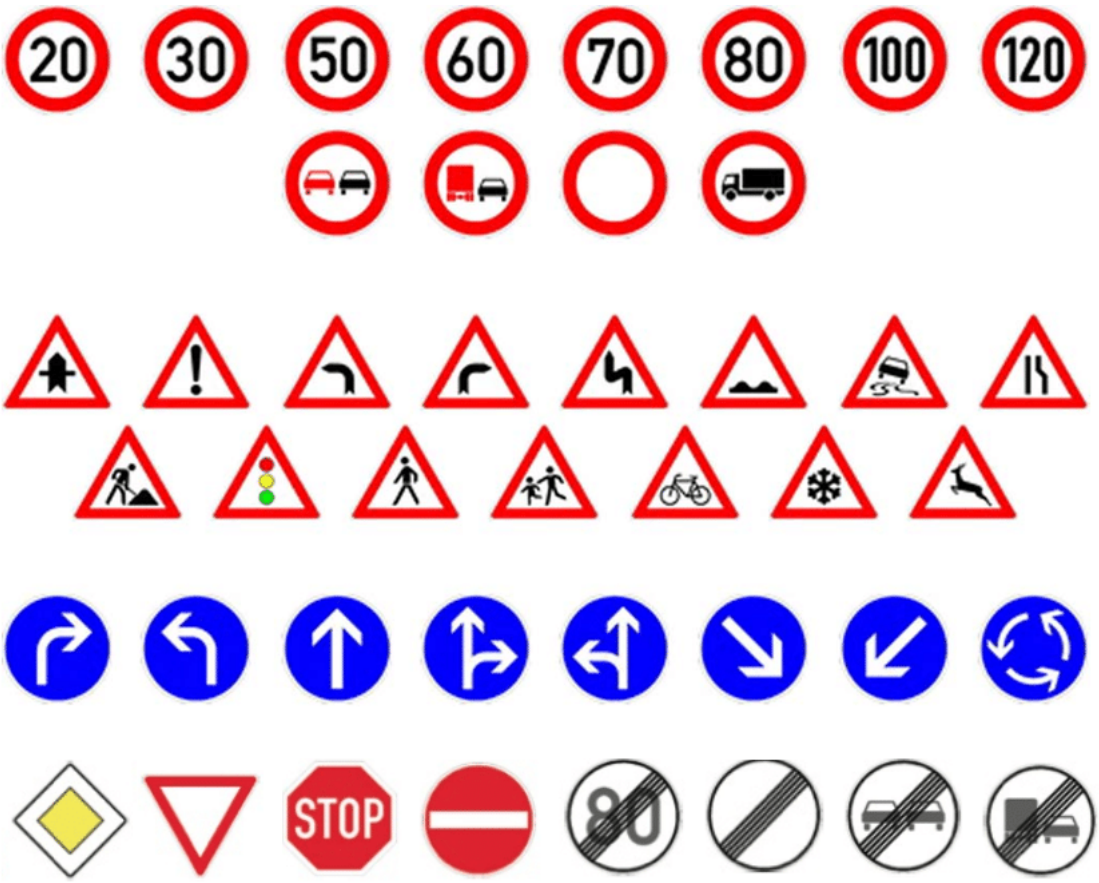
Chương 3

Ứng Dụng Convolutional Neural Network Vào Bài Toán Nhận Diện Biển Báo Giao Thông

3.1 Dữ liệu

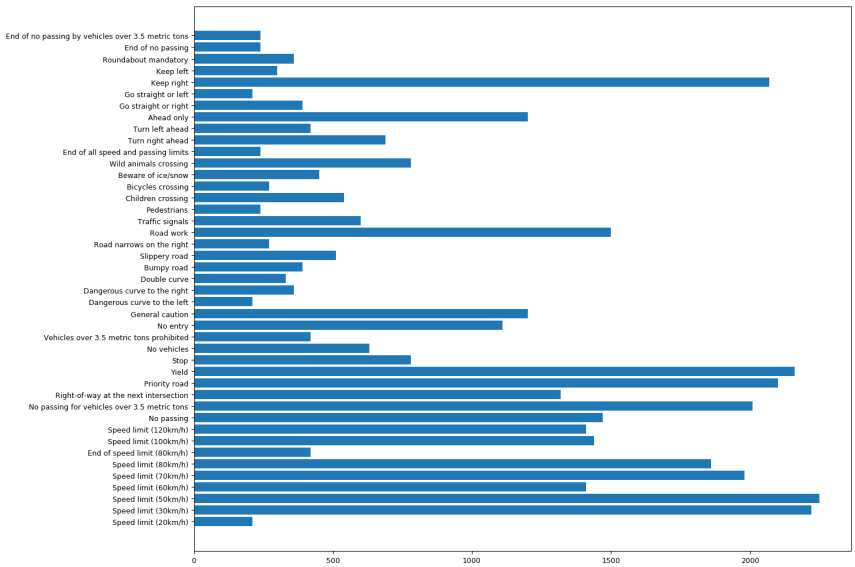
3.1.1 Thông tin dữ liệu

Dữ liệu được sử dụng trong tài liệu này là German Traffic Sign Recognition Benchmark (GTSRB) có hơn 50000 ảnh được chia ra 43 loại biển báo giao thông.



Hình 3.1: Danh sách các loại biển báo

Trong 50000 ảnh biển báo giao thông, có gần 40000 ảnh được dùng trong quá trình huấn luyện, hơn 12000 dùng để kiểm tra và kích thước ảnh được trải từ 15×15 đến 250×250 pixels và có định dạng file là '.ppm'. Biển báo abc có nhiều số lượng nhiều nhất là x ảnh, biển báo có lượng ít nhất abc với y ảnh.



Hình 3.2: Thống kê dữ liệu huấn luyện

Dữ liệu được chia làm hai thư mục chính: thư mục chứa tập dữ liệu huấn luyện và thư mục chứa dữ liệu kiểm tra. Trong thư mục dữ liệu huấn luyện, dữ liệu được phân chia thành 43 thư mục con tương ứng với các loại biển báo và được đánh số từ 0 đến 42, trong mỗi thư mục con này là các ảnh của biển báo tương ứng và một file csv chứa tên đầy đủ của file ảnh và nhãn của nó. Trong thư mục dữ liệu kiểm tra sẽ là các dữ liệu không xuất hiện trong tập huấn luyện và một file lưu thông tin giống như ở trong tập huấn luyện.

3.1.2 Tăng cường dữ liệu

Do dữ liệu chúng ta bị mất cân bằng vì thế chúng ta sẽ tạo ra thêm dữ liệu từ dữ liệu ban đầu. Đối với các loại biển báo có ít mẫu dữ liệu, tôi sử dụng một số phương pháp như: xoay ảnh, phóng to, thu nhỏ, dịch chuyển,... sao cho số lượng mẫu giữa các loại biển báo tương đương nahu và tương đương với lớp có số lượng mẫu dữ liệu lớn nhất.

3.1.3 Chuẩn hóa dữ liệu

Tất cả dữ liệu sẽ được chuyển về ảnh với format là gray và kích thước là 32×32 pixels.