BÀI 7. PHÁT TRIỂN HỆ THỐNG TTNT

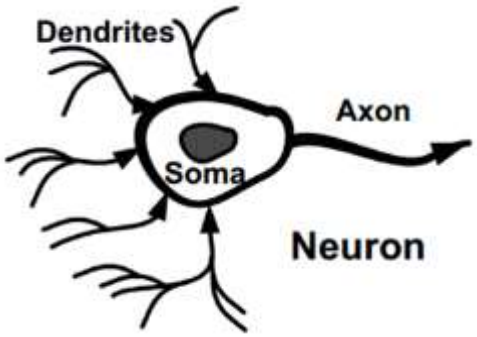
7.1. Mạng noron nhân tạo

a) Mô hình một nơ ron nhân tạo

Mạng nơ-ron nhân tạo (tiếng Anh: Neural Network) là một chuỗi các thuật toán được đưa ra để nỗ lực tìm kiếm các mối quan hệ cơ bản trong một tập hợp dữ liệu, thông qua quá trình bắt chước cách thức hoạt động của bộ não con người.

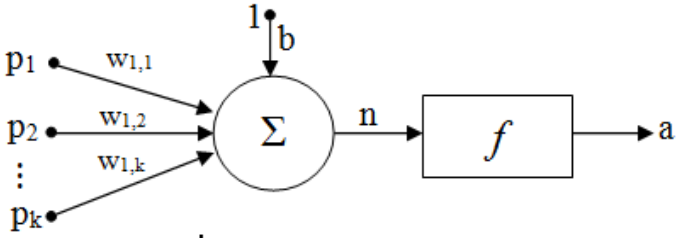
Mạng nơ-ron nhân tạo hoạt động tương tự như mạng nơ-ron của con người. Một "nơ-ron thần kinh" trong mạng nơ-ron nhân tạo là một hàm toán học có chức năng thu thập và phân loại thông tin theo một cấu trúc cụ thể.

Mạng nơ-ron nhân tạo có sự tương đồng chuẩn mạnh với các phương pháp thống kê như các đồ thị đường cong và phân tích hồi qui.



Hình 4.9. Cấu tạo của nơ ron sinh học

Từ cấu tạo của nơ ron sinh học chúng ta có thể mô hình hóa nơ ron nhân tạo như hình vẽ 4.10 sau:



Hình 4.10. Mô hình nơ ron nhân tạo

Các đầu vào p_1, p_2, …, p_k được kết nối thông qua trọng số có thể thay đổi được $w_{1,1}$, $w_{1,2}$, …, $w_{1,k}$ tương ứng để đưa vào bộ tổng. Ngoài ra, nơ ron còn có độ lệch (bias) b được cộng cùng với các đầu vào được trọng số hóa để tạo ra đầu ra n được tính theo công thức sau:

$$n = p_1 \times w_{1,1} + p_2 \times w_{1,2} + \cdots + p_k \times w_{1,k} + b = \sum_{i=1}^{k} p_i \times w_{1,i} + b = W \times p^T \quad (4.22)$$

Trong đó:

W là ma trận trọng số và $W = \begin{bmatrix} w_{1,1} \\ \vdots \\ w_{1,k} \end{bmatrix}$ và $p = [p_1, p_2, \dots, p_k]$

Giá trị đầu ra của nơ ron được tính theo công thức sau:

$$a = f(n) \quad (4.23)$$

Trong đó f được gọi là hàm truyền đạt (transfer function) hoặc hàm kích hoạt (activate function).

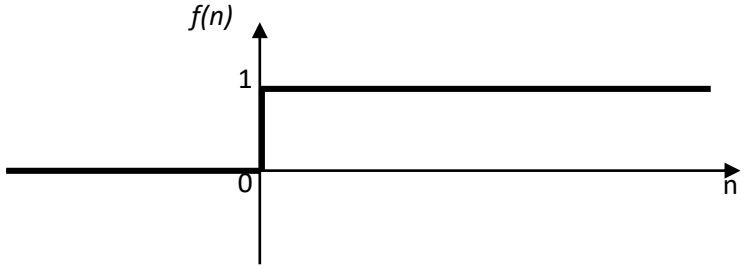
b) Hàm truyền đạt

Hàm truyền đạt có thể là tuyến tính hoặc phi tuyến, việc chọn hàm truyền đạt phù hợp cho nơ ron tùy thuộc vào mục đích sử dụng để giải quyết những vấn đề cụ thể. Một số hàm truyền đạt thường được sử dụng là:

-Hàm truyền hard limit (giới hạn cứng): Có biểu thức toán học như biểu thức 3.3 sau:

$$f(n) = \begin{cases} 0 \; với \; n < 0 \\ 1 \; với \; n \geq 0 \end{cases} \quad (4.24)$$

Trong thư viện của Matlab thì hàm truyền đạt tuyến tính có tên là hardlim

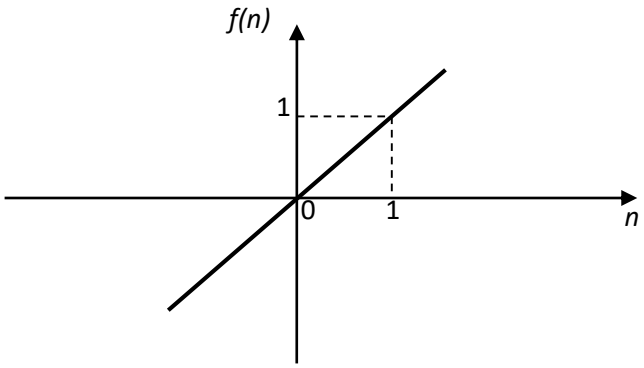


Hình 4.11. Hàm truyền đạt hard limit

- Hàm truyền đạt tuyến tính (linear transfer function): Biểu thức toán học của hàm truyền đạt tuyến tính như sau:

$$f(n) = n \quad (4.25)$$

Trong thư viện của Matlab hàm truyền tuyến tính là purelinvới cú pháp sau: a = purelin(n). Hình vẽ 4.12 bên dưới minh họa hàm truyền đạt tuyến tính.



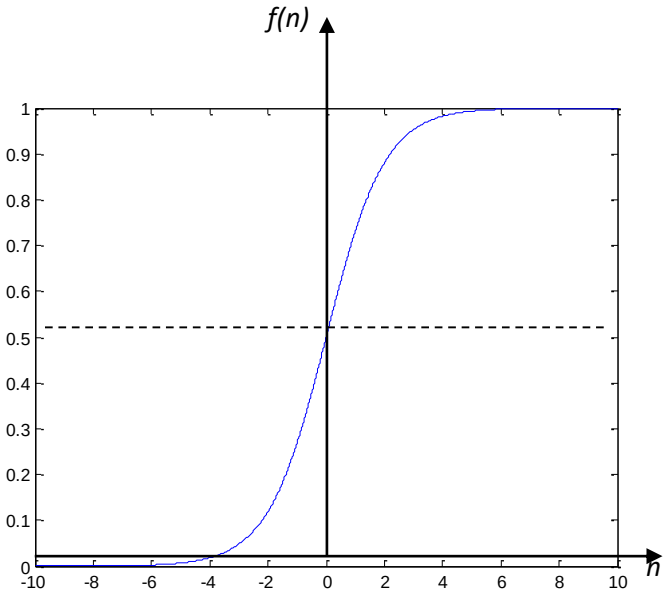
Hình 4.12. Hàm truyền đạt tuyến tính

- Hàm truyền đạt log-sigmoid

  Biểu thức toán học của hàm truyền đạt log-sigmoid như sau:

$$f(n) = \frac{1}{1+e^{-n}} \tag{4.26}$$

  Tên hàm trong thư viện của Matlab là logsig. Hình vẽ 4.13 bên dưới minh họa hàm truyền log-sigmoid.



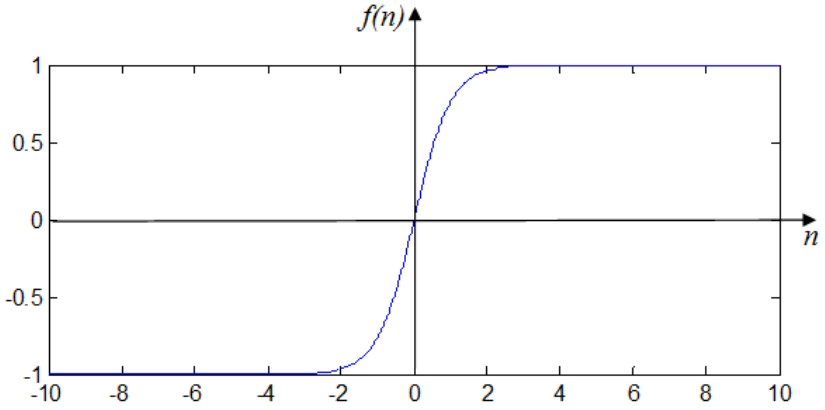
Hình 4.13. Hàm truyền đạt log-sigmoid

- Hàm truyền đạt tangent-sigmoid

  Biểu thức toán học của hàm truyền đạt tangent-sigmoid như sau:

$$f(n) = \frac{e^n - e^{-n}}{e^n + e^{-n}} \quad (4.27)$$

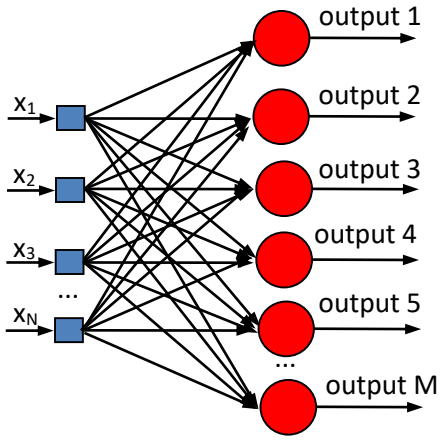
Hàm truyền đạt tangent-sigmoid trong thư viện của Matlab có tên là tansigvà được biểu diễn như hình vẽ 3.6 bên dưới.



Hình 4.14. Hàm truyền đạt tangent - sigmoid

c) Kiến trúc mạng Perceptron

Mạng Perceptron đơn giản chỉ bao gồm một lớp (*Singer Layer Network*). Nó được tạo nên từ *M* nơ ron nhân tạo, mỗi nơ ron nhận *N* tín hiệu đầu vào. Hình 4.3. chỉ ra kiến trúc của một mạng như vậy. Theo đó, có *N* trạm nhận tín hiệu vào để truyền tới *M* nơ ron cho ta một mạng có *M* × *N* đường truyền. Hiển nhiên, chúng ta cần thiết lập một ma trận trọng số có *M* × *N* giá trị.



Hình 4.15. Kiến trúc một mạng Perceptron

Gọi $x(x_1, x_2, \ldots, x_N)$ là véctơ tín hiệu đầu vào cho một mạng có M nơ ron. Ma trận trọng số w kích thước $M \times N$ với mỗi dòng w_i là véc tơ trọng số của nơ ron thứ i:

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} & \ldots & w_{1,N} \\ w_{2,1} & w_{2,2} & \ldots & w_{2,N} \\ \ldots & \ldots & \ldots & \ldots \\ w_{M,1} & w_{M,2} & \ldots & w_{MN} \end{bmatrix} .$$

Xét nơ ron thứ i (i=1,…, M). Véc tơ trọng số tương ứng của nó có thể biểu diễn dưới dạng véc tơ cột w_i bao gồm N giá trị ($w_i \in R^N$):

$$w_i = \begin{bmatrix} w_{i,1} \\ w_{i,2} \\ \ldots \\ w_{i,N} \end{bmatrix}.$$

Khi đó ma trận w được viết lại là (ký hiệu T là chuyển vị):

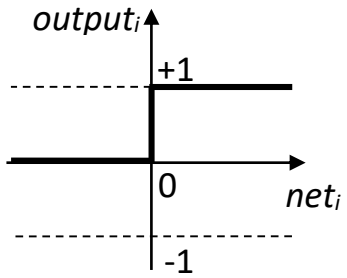
$$w = \begin{bmatrix} w_1^T \\ w_2^T \\ \ldots \\ w_M^T \end{bmatrix}.$$

và $b = (b_1, b_2, \ldots, b_M)^T$ là véctơ các độ lệch của M nơ ron. Giá trị *net* của nơ ron thứ i, ký hiệu net_i được tính như sau:

$$net_i = w_i^T p + b_i. \qquad (4.28)$$

Hàm kích hoạt $f(.)$ của các nơ ron trong mạng Perceptron thường là hàm *Hardlim* (Hình 4.4) cho ta một giá trị $output_i$ tại đầu ra ứng với giá trị đầu vào net_i :

$$ouput_i = f(net_i) = \text{Hardlim}(net_i) = \begin{cases} 1 & \text{if } net_i \geq 0 \\ 0 & \text{if } net_i < 0 \end{cases}. \qquad (4.29)$$

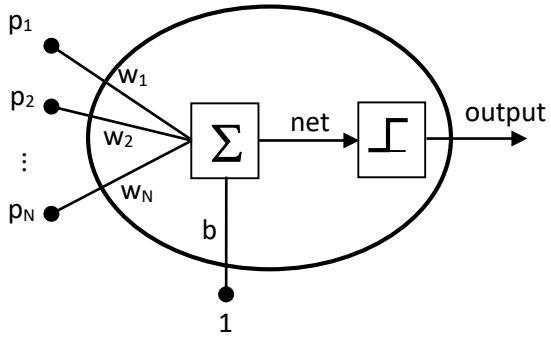


Hình 4.16. Đồ thị hàm kích hoạt Hardlim.

d) Huấn luyện mạng Perceptron

Dễ thấy, việc tính toán giá trị đầu ra của mạng ngoài phụ thuộc vào các tín hiệu vào, hàm kích hoạt được chọn, nó còn phụ thuộc mạnh vào bộ trọng số *w*. Nếu các giá trị này được chọn ngẫu nhiên, chất lượng của mạng có thể sẽ xấu. Do vậy, việc xác định bộ trọng số *w* tốt ứng với từng loại dữ liệu là rất quan trọng đối với một mạng nơ ron nhân tạo. Để làm được điều đó, mạng trải qua một quá trình huấn luyện.

Xét trường hợp mạng có 1 nơron: trong trường hợp này mạng có *N* đầu vào và 1 đầu ra (Hình 4.5).



Hình 4.17. Mạng Perceptron có 1 nơron

Ta xét quá trình huấn luyện mạng Perceptron dựa trên phương pháp học có giám sát. Giả sử ta có một mẫu dữ liệu huấn luyện {*x, y*} với $x \in N$ là *N* giá trị đầu vào và $y \in R$ là giá trị đầu ra tương ứng, đã biết trước. Ý tưởng thuật toán như sau: với bộ tín hiệu đầu vào *x* mạng sẽ cho ra một giá trị đầu ra tương ứng *ouput*. Giá trị *output* sẽ được so sánh với *y* và ta sẽ điều chỉnh các trọng số *w* và *b* sao cho độ chênh lệch *e = y - output* ngày càng giảm.

Giải thuật huấn luyện mạng Perceptron

Bước 1*: Khởi tạo ngẫu nhiên bộ trọng số w, b.*

Bước 2*: Lấy lần lượt các mẫu dữ liệu {x, y} trong tập dữ liệu huấn luyện, tính đầu ra output của mạng:*

$$ouput = Hardlim(w^T x + b).$$

***Bước 3**: So sánh output với y*

Nếu output ≠ y thì cập nhật $w = w + (y - output)x^T$.

$$b = b + (y - output).$$

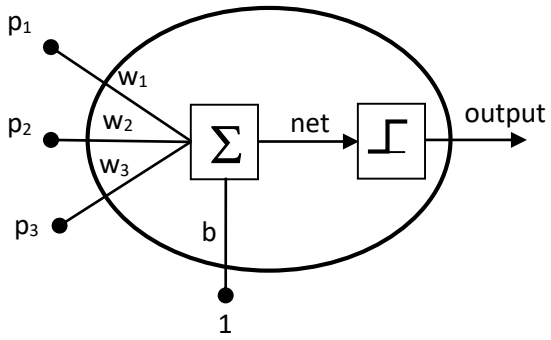
***Bước 4**: Lặp bước 2, bước 3 cho đến khi output = y với mọi mẫu dữ liệu {x, y}, hoặc thỏa mãn điều kiện dừng.*

Ví dụ 4.6. Xét bộ dữ liệu huấn luyện gồm hai cặp tín hiệu vào/ra (*n*=2) với mỗi bộ gồm 3 tín hiệu vào (*N*=3) và 1 tín hiệu ra như sau:

$$\{x_1 = (1 \ -1 \ -1)^T, y_1 = 0\},$$

$$\{x_2 = (1 \ 1 \ -1)^T, \ y_2 = 1\}.$$

Ta sẽ huấn luyện một mạng Perceptron có duy nhất 1 nơ ron với 3 đầu vào và 1 đầu ra (Hình 4.6) để tìm ra bộ trọng số tốt nhất.



Hình 4.18. Kiến trúc mạng Perceptron cho ví dụ trên

Trước tiên, ta khởi tạo bộ trọng số (*w, b*) một cách ngẫu nhiên, giả sử:

$$w = [0.5, -1, \ -0.5], \ b = 0.5.$$

Lần lặp thứ 1:

Xét mẫu dữ liệu huấn luyện thứ nhất $\{x_1, y_1\}$:

$$output_1 = \text{Hardlim}(w x_1 + b) = \text{Hardlim}\left([0.5 \ -1 \ -0.5] \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} + 0.5 \right)$$

$$= \text{Hardlim}(2.5) = 1.$$

Do $output_1 \neq y_1$ nên ta tiến hành cập nhật lại trọng số:

$w = w + (y_1 - output_1)\, x_1^T$ = [0.5 -1 -0.5] + (-1) [1 -1 -1] = [-0.5 0 0.5].

$b = b + (y_1 - output_1)$ = 0.5 + (-1) = -0.5.

Xét mẫu dữ liệu huấn luyện thứ 2 $\{x_2, y_2\}$:

$$output_2 = \text{Hardlim}(w\, x_2 + b) = \text{Hardlim}\left(\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} - 0.5 \right)$$

= hardlim(-0.5) = 0.

Do $output_2 \neq t_2$ nên ta cập nhật lại trọng số:

$w = w + (y_2 - output_2)\, x_2^T$ = [-0.5 0 0.5] + (1) [1 1 -1] = [0.5 1 -0.5].

$b = b + (y_2 - output_2)$ = -0.5 + 1 = 0.5.

Lần lặp thứ 2:

$$output_1 = \text{Hardlim}(w\, x_1 + b) = \text{Hardlim}\left(\begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} + 0.5 \right)$$

= Hardlim(0.5) = 1 ≠ y_1.

Ta tiến hành cập nhật lại bộ trọng số:

$w = w + (y_1 - output_1)\, x_1^T$ = [0.5 1 -0.5] + (-1)[1 -1 -1] = [-0.5 2 0.5].

$b = b + (y_1 - output_1)$ = 0.5 + (-1) = -0.5.

$$output_2 = \text{Hardlim}(w\, x_2 + b) = \text{Hhardlim}\left(\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} - 0.5 \right)$$

= Hardlim(0.5) = 1.

Do $output_2 = y_2$ nên ta giữ nguyên trọng số.

Lần lặp thứ 3:

$$output_1 = \text{Hardlim}(w\,x_1 + b) = \text{Hardlim}\left([-0.5\ \ 2\ 0.5] \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} - 0.5 \right) = 0 = t_1.$$

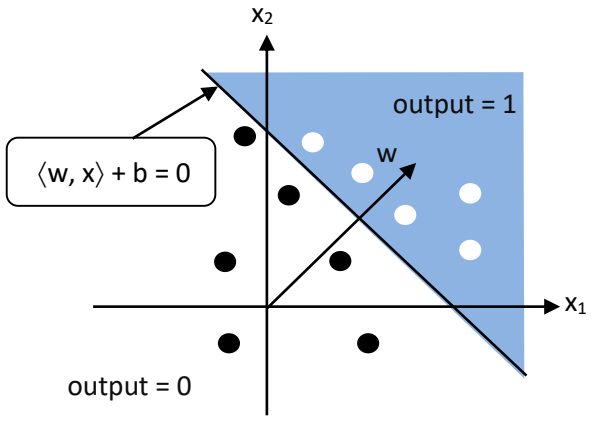
$$output_2 = \text{Hardlim}(w\,x_2 + b) = \text{Hardlim}\left([-0.5\ 2\ 0.5] \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} - 0.5 \right) = 1 = t_2.$$

Đến đây thuật toán có thể dừng và ta nhận được bộ trọng mới số: w = [-0.5 2 0.5]; b = -0.5. Kết thúc quá trình huấn luyện, ta tìm được bộ trọng số tối ưu cho mạng theo nghĩa là sai số xảy ra là thấp nhất. Giả sử cần phân loại một mẫu dữ liệu mới x = [1 -1 1]^T. Ta tính đầu ra của mạng tương ứng với đầu vào là x:

$$output = \text{Hardlim}(w\,x + b) = \text{Hardlim}\left([-0.5\ 2\ 0.5] \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} - 0.5 \right) = 0.$$

Nhận xét: từ kết quả mẫu x có nhiễu so với x_1 (x_1 = [1 -1 -1]^T) nhưng mạng vẫn có thể phân lớp chính xác.

Việc tìm ra bộ trọng số (w, b) tối ưu của một mạng Perceptron có 1 nơ ron thực chất là việc tìm ra một siêu phẳng có dạng $\langle w, x \rangle + b = 0$ sao cho siêu phẳng này phân tách bộ dữ liệu huấn luyện thành hai lớp (0 hoặc 1) một cách tốt nhất, tương tự như kỹ thuật SVM. Hình 4.7 dưới đây cho thấy bộ (w, b) tối ưu tìm được sẽ tương ứng với siêu phẳng tối ưu chia tập dữ liệu huấn luyện thành hai lớp 0 và 1 (giả sử dữ liệu ở không gian hai chiều với mỗi điểm dữ liệu có dạng $x(x_1, x_2)$.



Hình 4.19. *Siêu phẳng phân tách bộ dữ liệu huấn luyện một cách tối ưu sẽ cho ta các giá trị trọng số (w, b) tối ưu.*

Xét trường hợp mạng có nhiều nơ ron

Trong khi mạng Perceptron có 1 nơ ron chỉ phân chia tập các tín hiệu vào thành 2 lớp do đầu ra là một trong hai giá trị 0 hoặc 1 thì mạng Perceptron nhiều nơ ron có thể phân chia tập các tín hiệu vào một thành nhiều lớp do nó có nhiều đầu ra. Mỗi véc tơ đầu ra gồm M thành phần, mỗi thành phần lại nhận một trong hai giá trị 0 hoặc 1. Do vậy, số véc tơ đầu ra (hay số lớp) tối đa của mạng là 2^M.

Trong trường hợp này, bộ trọng số w của mạng là một ma trận $w \in R^{M\times N}$ và b là một véc tơ $b \in R^N$. Hiển nhiên, mỗi mẫu dữ liệu huấn luyện $\{x, y\}$ vẫn có $x \in R^N$ nhưng $y \in R^M$, tức là một véc tơ đầu ra thay vì 1 giá trị đầu ra.

Ta có thể sử dụng quy tắc huấn luyện của mạng Perceptron có 1 nơ ron ở trên để cập nhật từng dòng cho ma trận w và từng giá trị cho b. Quy tắc huấn luyện mạng như sau: hàng thứ k của ma trận w (tương ứng với trọng số của nơ ron thứ k) được cập nhật theo công thức ($k=1,\ldots,M$):

$$w_k = w_k + (y_k - output_k)\, x,$$

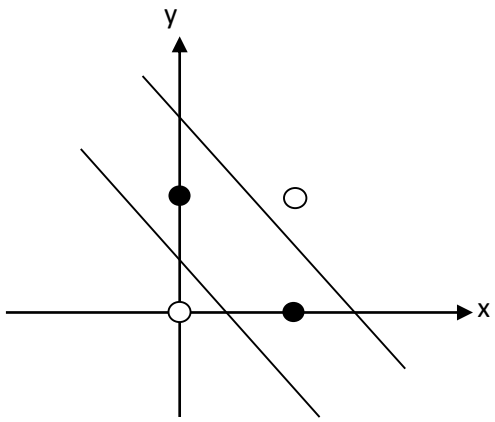
và phần tử thứ k của vectơ độ lệch b được cập nhật theo công thức:

$$b_k = b_k + (y_k - output_k)\ .$$

Quy tắc huấn luyện trên có thể viết lại dưới dạng ma trận:

$$w = w + (y - output)\, x^T \text{ và } b = b + (y - output).$$

Dễ thấy, do đường phân tách các lớp là siêu phẳng nên mạng Perceptron thường chỉ thực hiện tốt trên dữ liệu có khả năng phân tách tuyến tính (*linearly separable*). Đối với trường hợp dữ liệu không phân tách tuyến tính, mạng có thể thực hiện nhưng độ chính xác sẽ có thể bị ảnh hưởng. Chẳng hạn với bài toán XOR (Hình 4.8), không thể dùng một đường thẳng để phân chia tập 4 điểm dữ liệu thành hai lớp riêng biệt (lớp màu đen và lớp màu trắng). Để giải quyết bài toán này có thể phải mở rộng kiến trúc mạng thành nhiều lớp (*Multilayer Perceptrons –MLP*) đồng thời xây dựng thuật toán lan truyền ngược (*Backpropagation Algorithm*) trên kiến trúc mạng này.



Hình 4.20. Bài toán phân lớp cho mạch XOR