Mạng neuron
và ứng dụng trong xử lý tín hiệu

Giảng viên

Trần Thị Thanh Hải

International Research Institute MICA

Multimedia, Information, Communication & Applications

UMI 2954

Hanoi University of Science and Technology

1 Dai Co Viet - Hanoi - Vietnam

Bài 7:

Phương pháp huấn luyện MLP

(Multi Layer Perceptron)

Giới thiệu

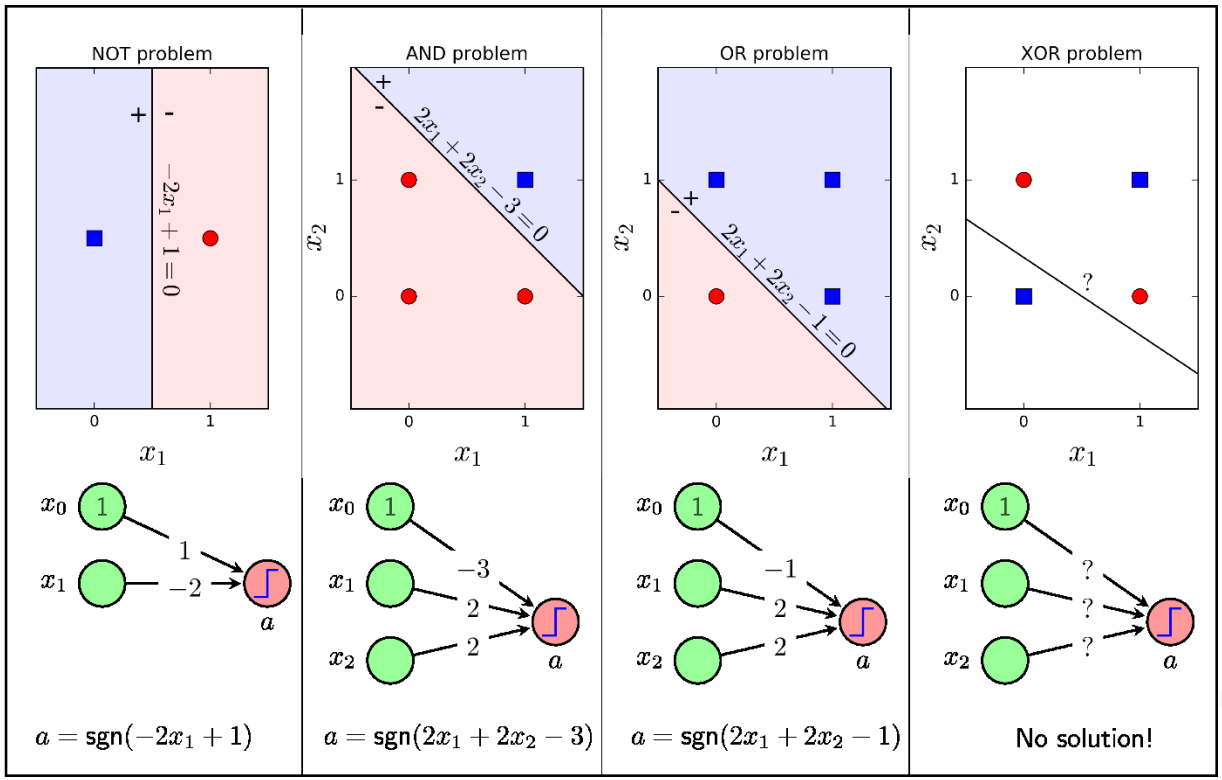
- Trong bài trước, chúng ta đã biết cách huấn luyện mạng neuron perceptron 1 lớp (1 đầu vào và một đầu ra)
- Đối với perceptron, việc tính toán đạo hàm là khả thi vì chỉ có một lớp
- Trong các bài toán thực tế đòi hỏi mạng neuron có cấu trúc phức tạp hơn, nhiều lớp hơn
- Khi đó việc tính toán đạo hàm của hàm mục tiêu là rất phức tạp

Mục tiêu của bài học

- Giới thiệu một giải pháp để tính toán đạo hàm của hàm mục tiêu với mạng đa lớp sử dụng kỹ thuật lan truyền ngược (Backpropagation)
- Thực hành thông qua một ví dụ

Các hạn chế của mạng Perceptron 1 lớp

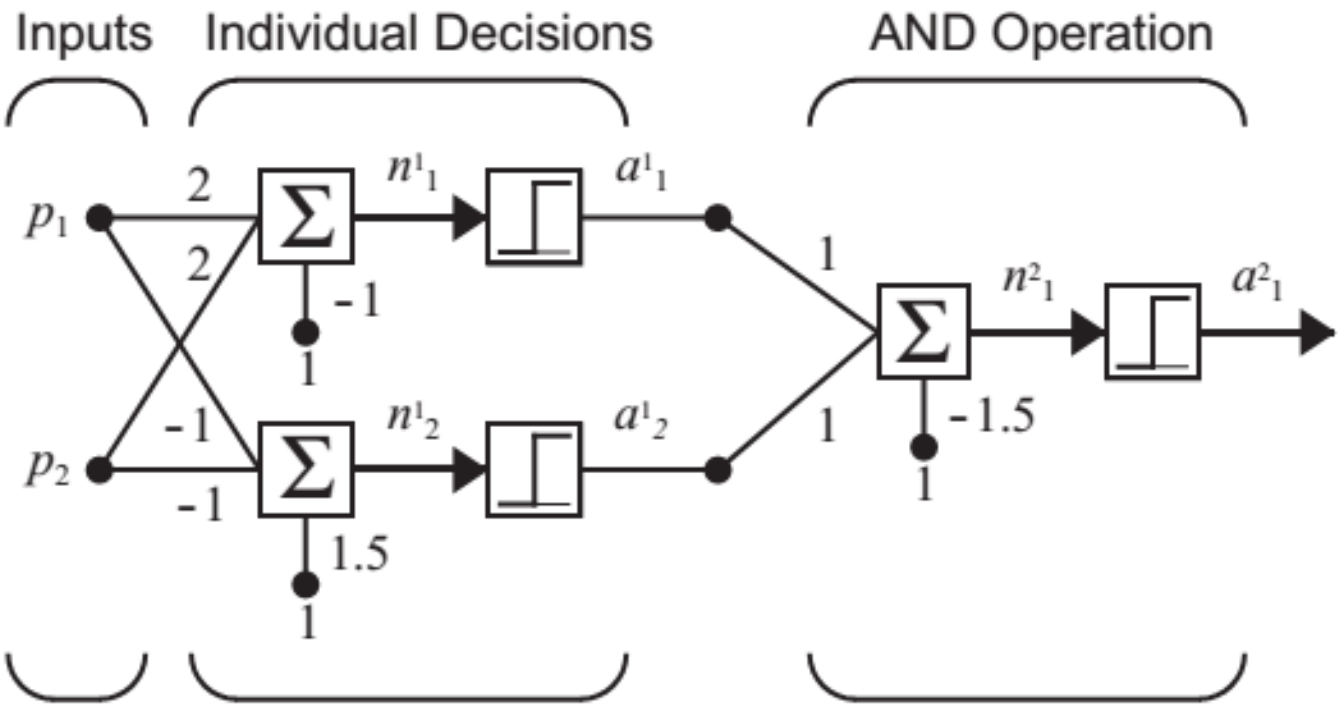
- Perceptron: mạng 1 lớp đầu vào và một lớp đầu ra, không có hidden layer
- Mạng perceptron không thể phân loại các dữ liệu có phân tách phi tuyến (XOR function)



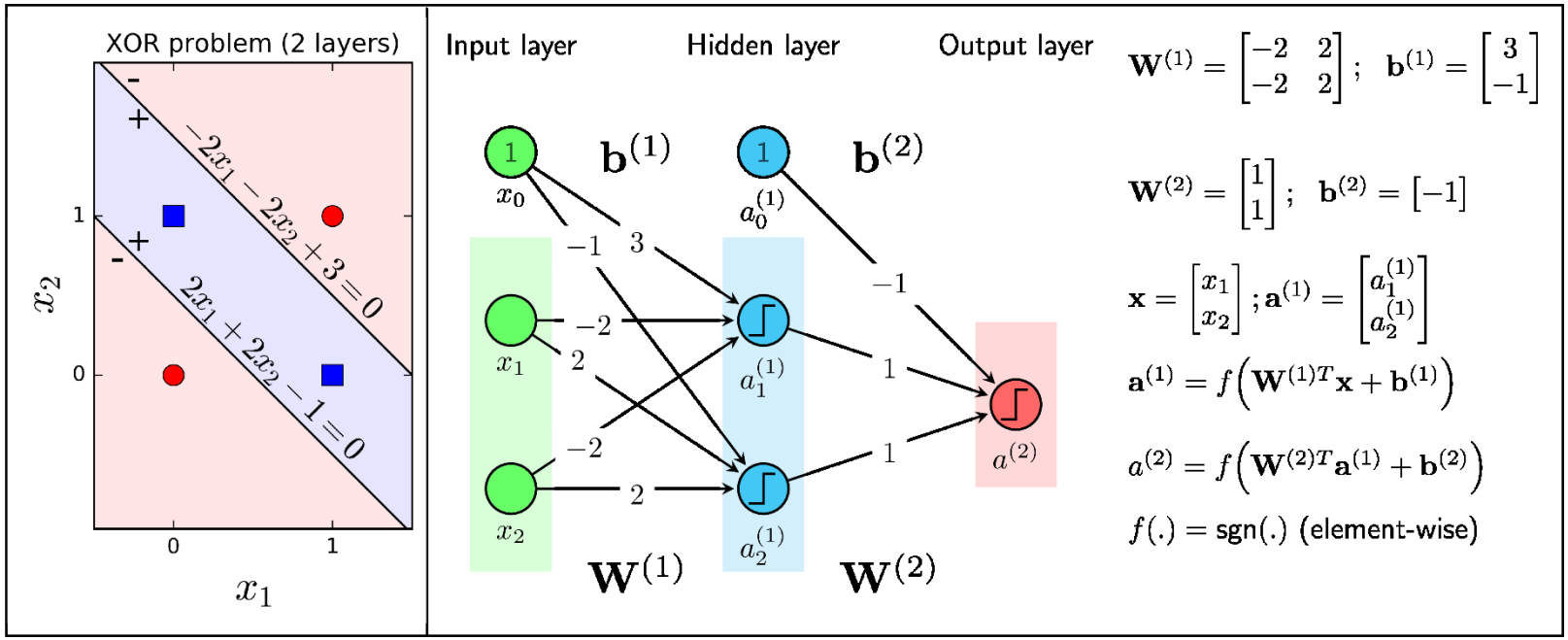
Khắc phục hạn chế của Perceptron 1 lớp

- **Thực tế có nhiều mạng đa lớp có thể giải quyết bài toán XOR**
- **Một lời giải:**
  - Thiết kế 2 neuron để tạo ra hai bao đóng
  - Thêm một neuron sau đó để kết hợp hai bao đóng lại thành một (toán tử AND)

Giải pháp cho toán tử XOR

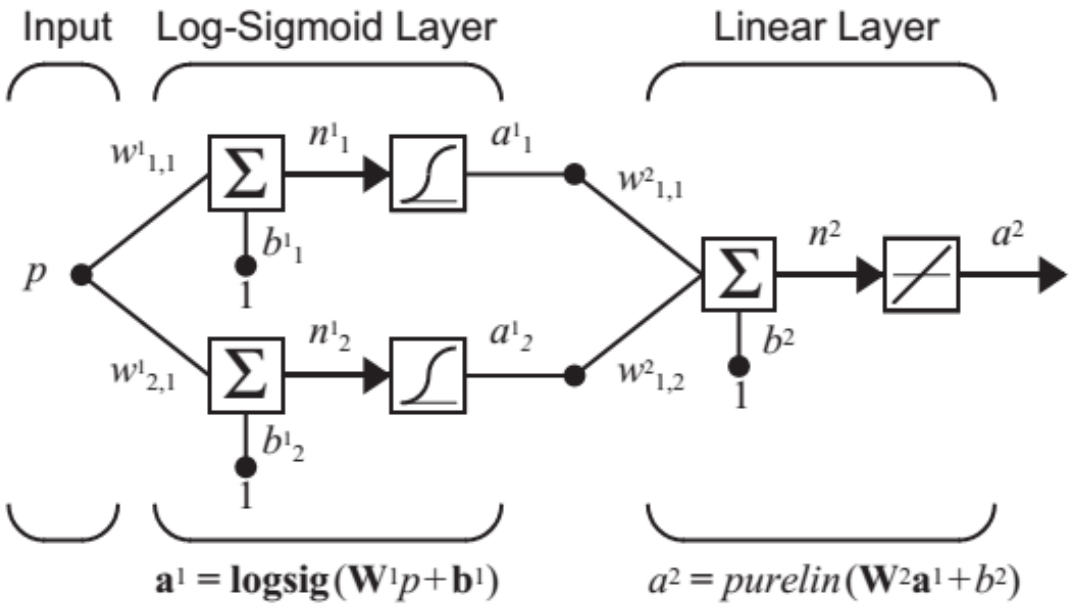


Giải pháp cho toán tử XOR



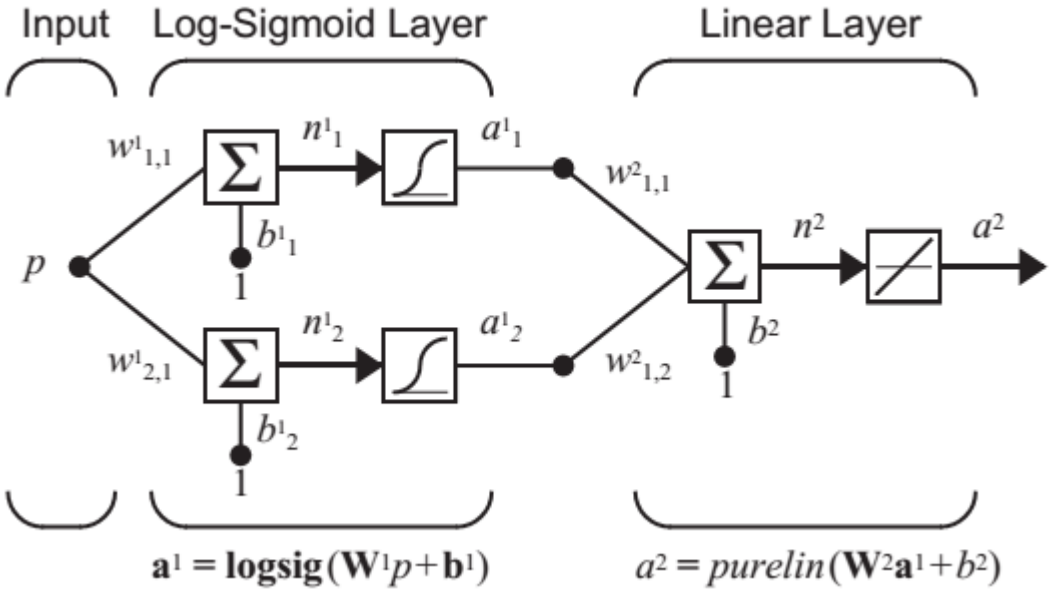
Xấp xỉ hàm (function approximation)

- Ngoài ứng dụng trong phân lớp, các mạng neuron đa lớp cũng được sử dụng để tạo ra các hàm xấp xỉ trong các hệ thống điều khiển



Xấp xỉ hàm (approximate function)

$$f^1(n) = \frac{1}{1+e^{-n}} \text{ and } f^2(n) = n$$

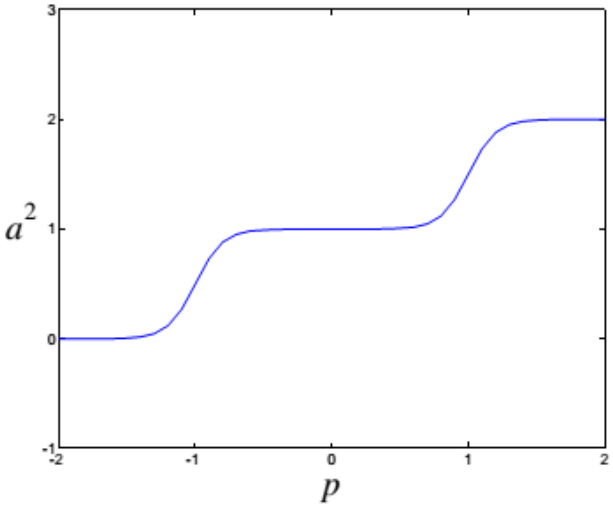


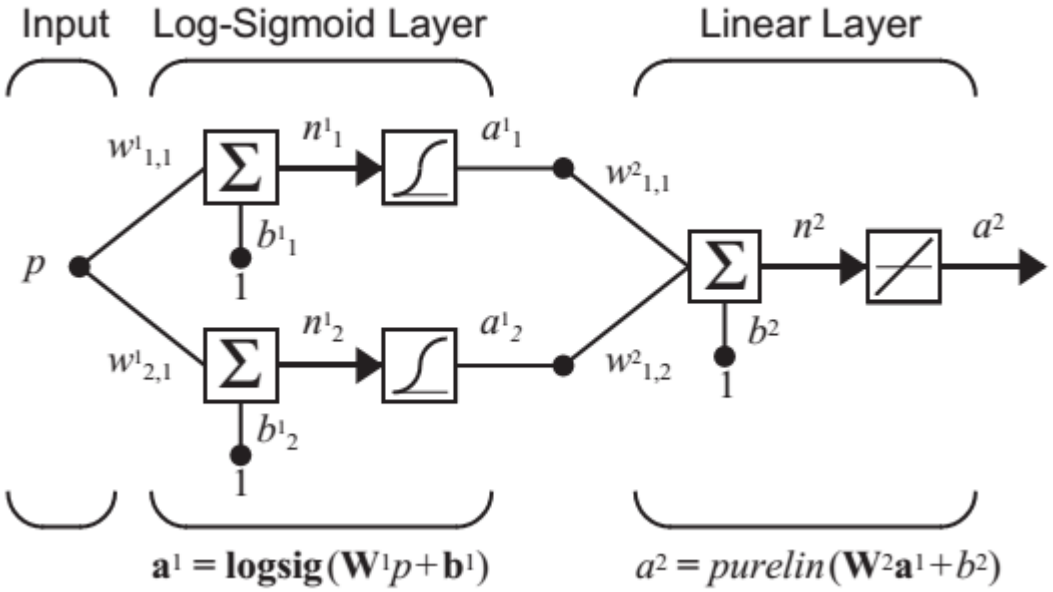
- **Giả thiết các thông số của mạng như sau**

$$w^1_{1,1} = 10,\ w^1_{2,1} = 10,\ b^1_1 = -10,\ b^1_2 = 10,$$

$$w^2_{1,1} = 1,\ w^2_{1,2} = 1,\ b^2 = 0.$$

Xấp xỉ hàm (approximate function)





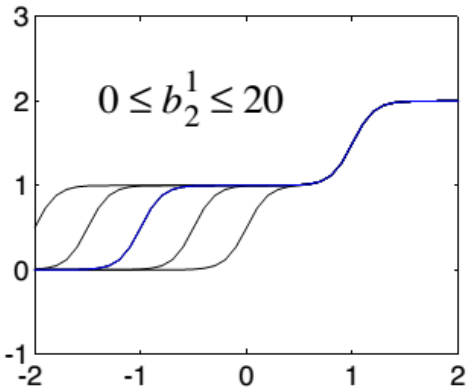
- **Giả thiết các thông số của mạng như sau**

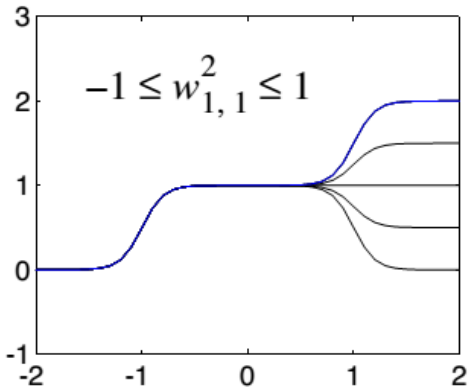
$$w^1_{1,1} = 10,\ w^1_{2,1} = 10,\ b^1_1 = -10,\ b^1_2 = 10,$$

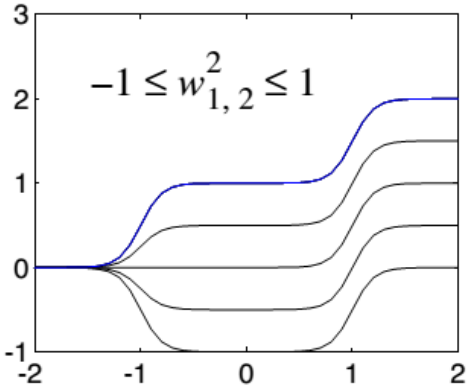
$$w^2_{1,1} = 1,\ w^2_{1,2} = 1,\ b^2 = 0.$$

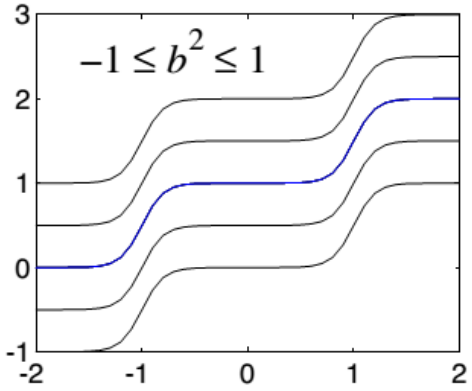
Xấp xỉ hàm (approximate function)

- Đây là xấp xỉ hàm hai bước. Nếu ta thêm các trọng số thì sẽ làm thay đổi hình dáng của đáp ứng.





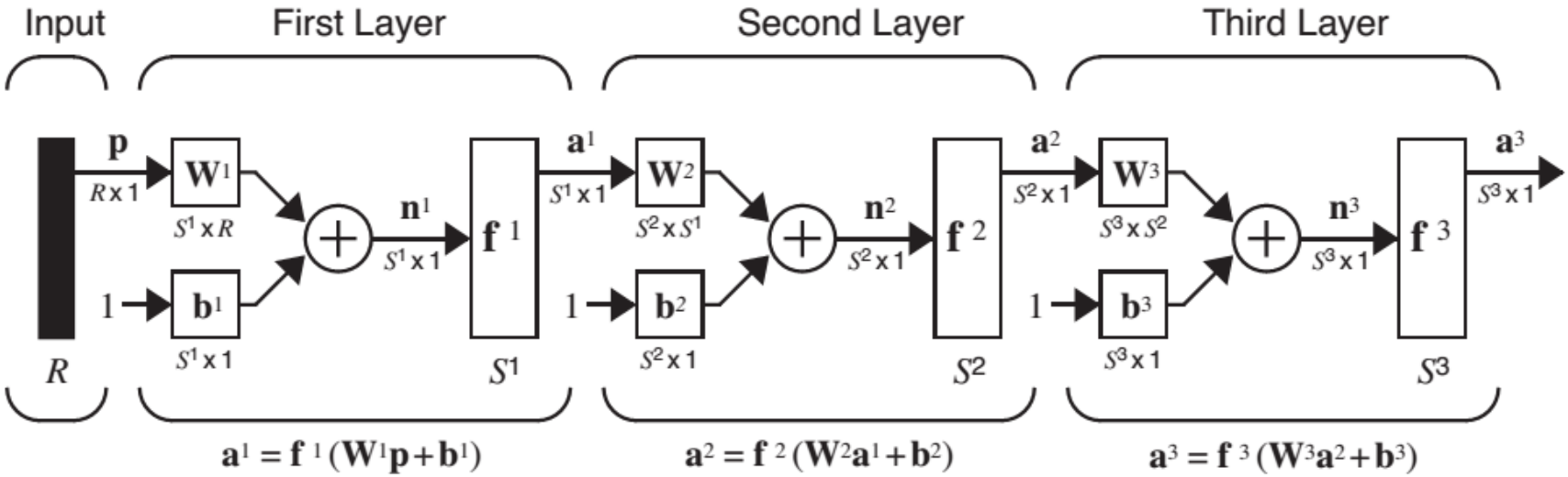




Mạng đa tầng

- Mạng với 1 tầng ẩn có thể biểu diễn bất kỳ hàm Boolean nào
- Khả năng của mạng nhiều tầng đã được khám phá từ lâu, tuy nhiên chỉ đến những năm 80 người ta mới biết cách để huấn luyện các mạng này
- Mạng nhiều tầng, mỗi tầng có hàm kích hoạt tuyến tính thì vẫn chỉ có thể biểu diễn các hàm tuyến tính
- Để biểu diễn các hàm phi tuyến thì các hàm kích hoạt phải là hàm phi tuyến

Multilayer Network



$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^{m} + \mathbf{b}^{m+1}) \qquad m = 0, 2, \ldots, M-1$$

$$\mathbf{a}^{0} = \mathbf{p}$$

$$\mathbf{a} = \mathbf{a}^{M}$$

Huấn luyện mạng MLP

Training Set

$$\{\mathbf{p}_1,\mathbf{t}_1\},\{\mathbf{p}_2,\mathbf{t}_2\},\ldots,\{\mathbf{p}_Q,\mathbf{t}_Q\}$$

Mean Square Error

$$F(\mathbf{x}) = E[e^2] = E[(t-a)^2]$$

Vector Case

$$F(\mathbf{x}) = E[\mathbf{e}^T\mathbf{e}] = E[(\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})]$$

Approximate Mean Square Error (Single Sample)

$$\hat{F}(\mathbf{x}) = (\mathbf{t}(k)-\mathbf{a}(k))^T(\mathbf{t}(k)-\mathbf{a}(k)) = \mathbf{e}^T(k)\mathbf{e}(k)$$

Approximate Steepest Descent

$$w^m_{i,j}(k+1) = w^m_{i,j}(k) - \alpha\frac{\partial\hat{F}}{\partial w^m_{i,j}} \qquad b^m_i(k+1) = b^m_i(k) - \alpha\frac{\partial\hat{F}}{\partial b^m_i}$$

Giải thuật huấn luyện mạng MLP

- Giải thuật GD được sử dụng
- Tuy nhiên việc tính toán Gradient của hàm mục tiêu trên tất cả các tầng là rất phức tạp
- Giải thuật Backpropagation cho một giải pháp hiệu quả và đơn giản để tính toán đạo hàm của hàm mục tiêu theo trọng số và bias ở các tầng khác nhau

Giải thuật huấn luyện mạng MLP

- **Sử dụng kỹ thuật lan truyền ngược**
- **Kỹ thuật này được nghiên cứu đề xuất nhiều lần bởi các nhà khoa học**
  - Bryson an Ho [1969]
  - Werbos [1974]
  - Parker [1985]
  - Rumelhart et al. [1986]

Chain Rule

$$\frac{df(n(w))}{dw} = \frac{df(n)}{dn} \times \frac{dn(w)}{dw}$$

Example

$$f(n) = \cos(n) \qquad n = e^{2w} \qquad f(n(w)) = \cos(e^{2w})$$

$$\frac{df(n(w))}{dw} = \frac{df(n)}{dn} \times \frac{dn(w)}{dw} = (-\sin(n))(2e^{2w}) = (-\sin(e^{2w}))(2e^{2w})$$

Application to Gradient Calculation

$$\frac{\partial \hat{F}}{\partial w_{i,j}^{m}} = \frac{\partial \hat{F}}{\partial n_{i}^{m}} \times \frac{\partial n_{i}^{m}}{\partial w_{i,j}^{m}} \qquad \frac{\partial \hat{F}}{\partial b_{i}^{m}} = \frac{\partial \hat{F}}{\partial n_{i}^{m}} \times \frac{\partial n_{i}^{m}}{\partial b_{i}^{m}}$$

Gradient calculation

$$n_i^m = \sum_{j=1}^{S^{m-1}} w_{i,j}^m a_j^{m-1} + b_i^m$$

$$\frac{\partial n_i^m}{\partial w_{i,j}^m} = a_j^{m-1} \qquad \frac{\partial n_i^m}{\partial b_i^m} = 1$$

Sensitivity

$$s_i^m \equiv \frac{\partial \hat{F}}{\partial n_i^m}$$

Gradient

$$\frac{\partial \hat{F}}{\partial w_{i,j}^m} = s_i^m a_j^{m-1} \qquad \frac{\partial \hat{F}}{\partial b_i^m} = s_i^m$$

Steepest Descent

$$w^m_{i,j}(k+1) = w^m_{i,j}(k) - \alpha s^m_i a^{m-1}_j \qquad b^m_i(k+1) = b^m_i(k) - \alpha s^m_i$$

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T \qquad \mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m$$

$$\mathbf{s}^m \equiv \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \begin{bmatrix} \frac{\partial \hat{F}}{\partial n^m_1} \\ \frac{\partial \hat{F}}{\partial n^m_2} \\ \vdots \\ \frac{\partial \hat{F}}{\partial n^m_{S^m}} \end{bmatrix}$$

Next Step: Compute the Sensitivities (Backpropagation)

Jacobian Matrix

$$\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^{m}} \equiv \begin{bmatrix} \frac{\partial n_1^{m+1}}{\partial n_1^m} & \frac{\partial n_1^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_1^{m+1}}{\partial n_{S^m}^m} \\ \frac{\partial n_2^{m+1}}{\partial n_1^m} & \frac{\partial n_2^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_2^{m+1}}{\partial n_{S^m}^m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_1^m} & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_{S^m}^m} \end{bmatrix}$$

$$\frac{\partial n_i^{m+1}}{\partial n_j^m} = \frac{\partial \left(\sum_{l=1}^{S^m} w_{i,l}^{m+1} a_l^m + b_i^{m+1} \right)}{\partial n_j^m} = w_{i,j}^{m+1} \frac{\partial a_j^m}{\partial n_j^m}$$

$$\frac{\partial n_i^{m+1}}{\partial n_j^m} = w_{i,j}^{m+1} \frac{\partial f^m(n_j^m)}{\partial n_j^m} = w_{i,j}^{m+1} \dot{f}^m(n_j^m)$$

$$\dot{f}^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m}$$

$$\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^{m}} = \mathbf{W}^{m+1} \dot{\mathbf{F}}^m(\mathbf{n}^m)$$

$$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \begin{bmatrix} \dot{f}^m(n_1^m) & 0 & \cdots & 0 \\ 0 & \dot{f}^m(n_2^m) & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \dot{f}^m(n_{S^m}^m) \end{bmatrix}$$

Backpropagation (Sensitivities)

$$\mathbf{s}^m = \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \left(\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^m}\right)^T \frac{\partial \hat{F}}{\partial \mathbf{n}^{m+1}} = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T \frac{\partial \hat{F}}{\partial \mathbf{n}^{m+1}}$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T \mathbf{s}^{m+1}$$

The sensitivities are computed by starting at the last layer, and then propagating backwards through the network to the first layer.

$$\mathbf{s}^M \rightarrow \mathbf{s}^{M-1} \rightarrow \ldots \rightarrow \mathbf{s}^2 \rightarrow \mathbf{s}^1$$

Initialization (last layer)

$$s_i^M = \frac{\partial \hat{F}}{\partial n_i^M} = \frac{\partial (\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})}{\partial n_i^M} = \frac{\partial \sum_{j=1}^{S^M} (t_j - a_j)^2}{\partial n_i^M} = -2(t_i - a_i)\frac{\partial a_i}{\partial n_i^M}$$

$$\frac{\partial a_i}{\partial n_i^M} = \frac{\partial a_i^M}{\partial n_i^M} = \frac{\partial f^M(n_i^M)}{\partial n_i^M} = \dot{f}^M(n_i^M)$$

$$s_i^M = -2(t_i - a_i)\dot{f}^M(n_i^M)$$

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t}-\mathbf{a})$$

Summary

Forward Propagation

$$\mathbf{a}^0 = \mathbf{p}$$

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1}) \qquad m = 0, 2, \ldots, M-1$$

$$\mathbf{a} = \mathbf{a}^M$$

Backpropagation

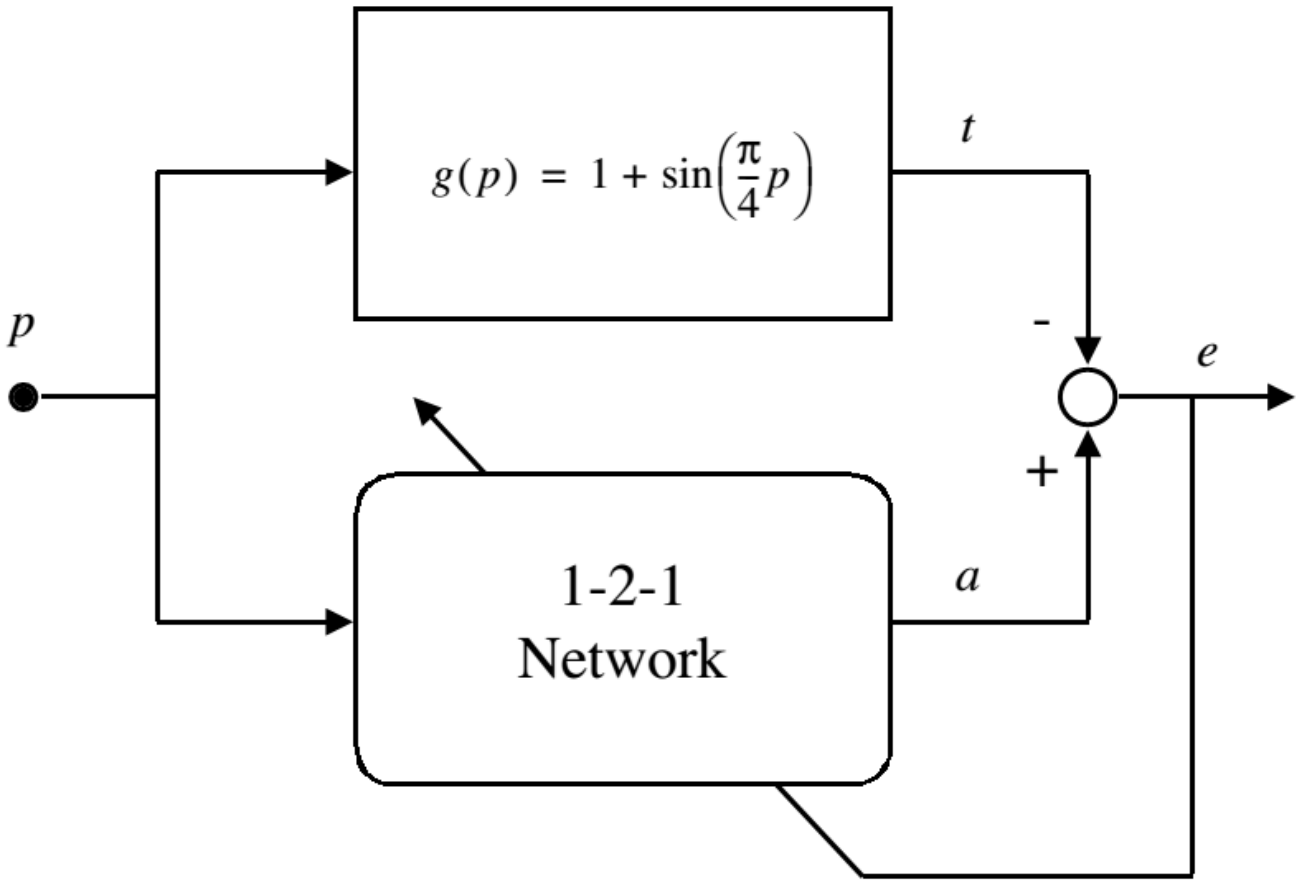
$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t} - \mathbf{a})$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1} \qquad m = M-1, \ldots, 2, 1$$

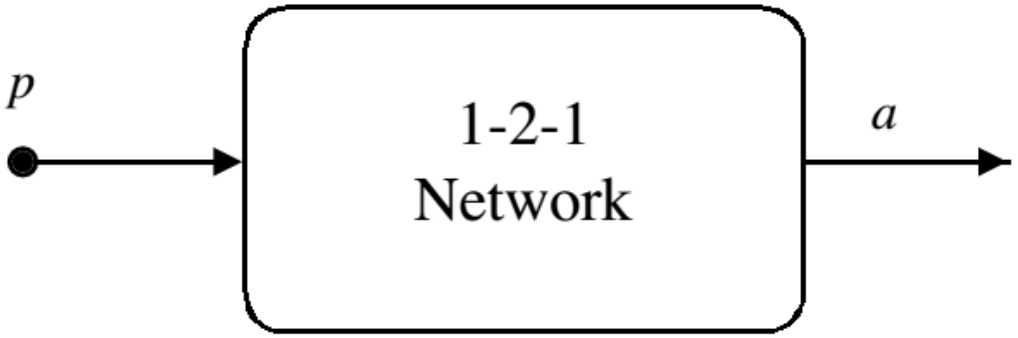
Weight Update

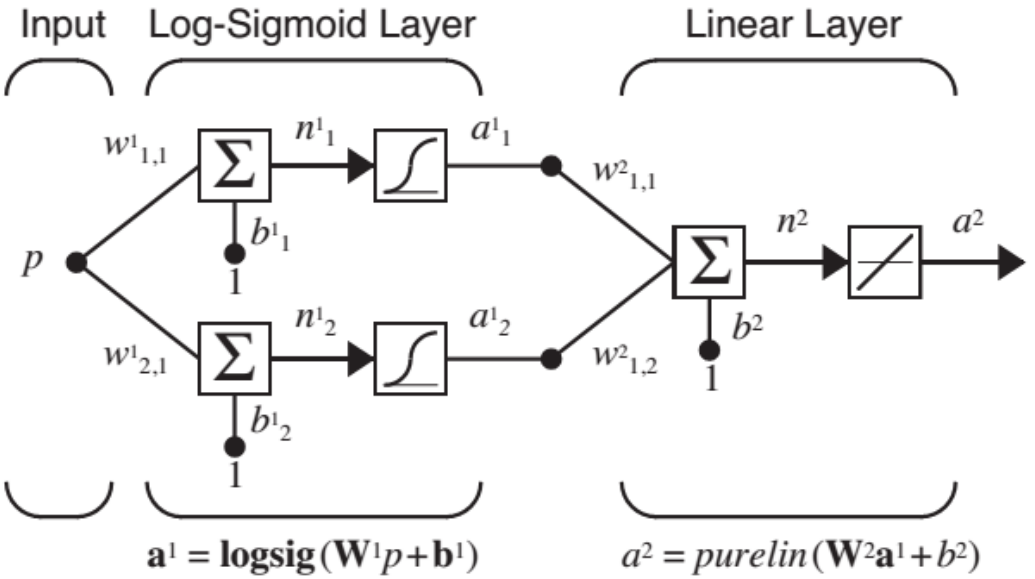
$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha\mathbf{s}^m(\mathbf{a}^{m-1})^T \qquad \mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha\mathbf{s}^m$$

Ví dụ: Xấp xỉ hàm



Kiến trúc của mạng





Khởi tạo các thông số của mạng

$$\mathbf{W}^1(0) = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix}, \mathbf{b}^1(0) = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix}, \mathbf{W}^2(0) = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix}, \mathbf{b}^2(0) = \begin{bmatrix} 0.48 \end{bmatrix}$$

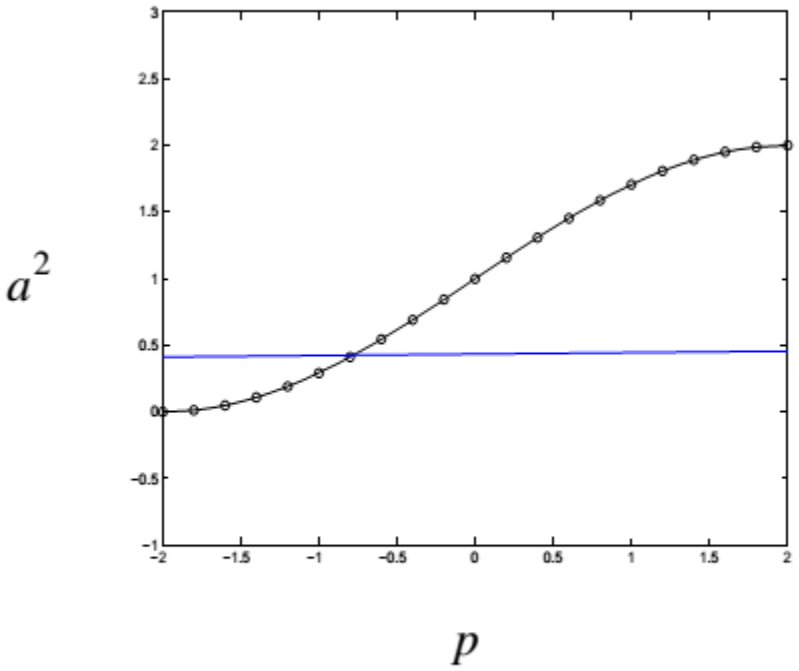


Figure 11.9 Initial Network Response

Dữ liệu huấn luyện mạng

- **Chuẩn bị tập dữ liệu huấn luyện:** lấy mẫu hàm với 21 điểm trong khoảng [-2, 2] (các chấm tròn trên hình vẽ), các điểm cách đều 0.2

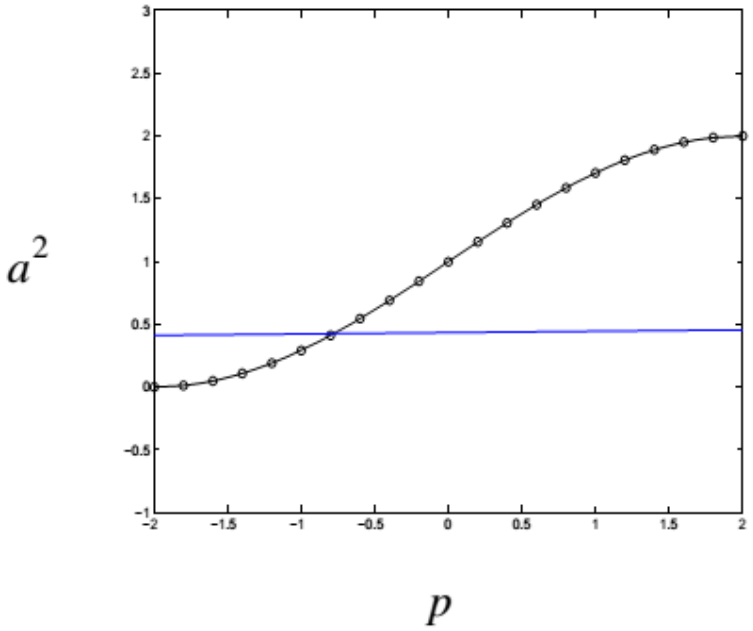


Figure 11.9 Initial Network Response

Huấn luyện mạng

- Bắt đầu với điểm p = 1 (điểm thứ 16)

$$a^0 = p = 1$$

- Đầu ra của lớp thứ nhất:

$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{a}^0 + \mathbf{b}^1) = \mathbf{logsig}\left(\begin{bmatrix}-0.27\\-0.41\end{bmatrix}[1] + \begin{bmatrix}-0.48\\-0.13\end{bmatrix}\right) = \mathbf{logsig}\left(\begin{bmatrix}-0.75\\-0.54\end{bmatrix}\right)$$

$$= \begin{bmatrix}\frac{1}{1+e^{0.75}}\\ \frac{1}{1+e^{0.54}}\end{bmatrix} = \begin{bmatrix}0.321\\0.368\end{bmatrix}.$$

Huấn luyện mạng

- Bắt đầu với điểm p = 1 (điểm thứ 16)

$$a^0 = p = 1$$

- Đầu ra của lớp thứ hai:

$$a^2 = f^2(\mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2) = purelin\left(\begin{bmatrix}0.09 & -0.17\end{bmatrix}\begin{bmatrix}0.321\\0.368\end{bmatrix} + \begin{bmatrix}0.48\end{bmatrix}\right) = \begin{bmatrix}0.446\end{bmatrix}$$

- Giá trị của hàm lỗi

$$e = t - a = \left\{1 + \sin\left(\frac{\pi}{4}p\right)\right\} - a^2 = \left\{1 + \sin\left(\frac{\pi}{4}1\right)\right\} - 0.446 = 1.261$$

Huấn luyện mạng

- **Tính đạo hàm của các hàm truyền đạt**
  - Lớp thứ nhất:

$$\dot{f}^1(n) = \frac{d}{dn}\left(\frac{1}{1+e^{-n}}\right) = \frac{e^{-n}}{(1+e^{-n})^2} = \left(1-\frac{1}{1+e^{-n}}\right)\left(\frac{1}{1+e^{-n}}\right) = (1-a^1)(a^1)$$

  - Lớp thứ 2:

$$\dot{f}^2(n) = \frac{d}{dn}(n) = 1$$

Lan truyền ngược

- **Ở lớp thứ 2:**

$$\mathbf{s}^2 = -2\dot{\mathbf{F}}^2(\mathbf{n}^2)(\mathbf{t}-\mathbf{a}) = -2\left[\dot{f}^2(n^2)\right](1.261) = -2\left[1\right](1.261) = -2.522$$

- **Lan truyền ngược về lớp thứ nhất:**

$$\mathbf{s}^1 = \dot{\mathbf{F}}^1(\mathbf{n}^1)(\mathbf{W}^2)^T\mathbf{s}^2 = \begin{bmatrix}(1-a_1^1)(a_1^1) & 0\\ 0 & (1-a_2^1)(a_2^1)\end{bmatrix}\begin{bmatrix}0.09\\ -0.17\end{bmatrix}\begin{bmatrix}-2.522\end{bmatrix}$$

$$= \begin{bmatrix}(1-0.321)(0.321) & 0\\ 0 & (1-0.368)(0.368)\end{bmatrix}\begin{bmatrix}0.09\\ -0.17\end{bmatrix}\begin{bmatrix}-2.522\end{bmatrix}$$

$$= \begin{bmatrix}0.218 & 0\\ 0 & 0.233\end{bmatrix}\begin{bmatrix}-0.227\\ 0.429\end{bmatrix} = \begin{bmatrix}-0.0495\\ 0.0997\end{bmatrix}.$$

Cập nhật trọng số

$$\mathbf{W}^2(1) = \mathbf{W}^2(0) - \alpha \mathbf{s}^2 (\mathbf{a}^1)^T = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix} - 0.1 \begin{bmatrix} -2.522 \end{bmatrix} \begin{bmatrix} 0.321 & 0.368 \end{bmatrix}$$

$$= \begin{bmatrix} 0.171 & -0.0772 \end{bmatrix},$$

$$\mathbf{b}^2(1) = \mathbf{b}^2(0) - \alpha \mathbf{s}^2 = \begin{bmatrix} 0.48 \end{bmatrix} - 0.1 \begin{bmatrix} -2.522 \end{bmatrix} = \begin{bmatrix} 0.732 \end{bmatrix},$$

$$\mathbf{W}^1(1) = \mathbf{W}^1(0) - \alpha \mathbf{s}^1 (\mathbf{a}^0)^T = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} \begin{bmatrix} 1 \end{bmatrix} = \begin{bmatrix} -0.265 \\ -0.420 \end{bmatrix},$$

$$\mathbf{b}^1(1) = \mathbf{b}^1(0) - \alpha \mathbf{s}^1 = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} = \begin{bmatrix} -0.475 \\ -0.140 \end{bmatrix}.$$

Thảo luận về kỹ thuật lan truyền ngược

- **Lựa chọn kiến trúc mạng**
  - Mạng đa lớp có thể sử dụng để xấp xỉ bất kỳ một hàm nào nếu có đủ số neuron trong mỗi lớp ẩn
  - Tuy nhiên, bao nhiêu layers và bao nhiêu units là đủ để đạt được một hiệu quả nhất định
  - Ví dụ: cần xấp xỉ hàm, với i = 1, 2, 4, 8

$$g(p) = 1 + \sin\left(\frac{i\pi}{4}p\right) \text{ for } -2 \le p \le 2$$

  - Khi i tăng, hàm trở nên phức tạp vì có nhiều chu kỳ của sóng sin trong khoảng [-2, 2]

Thảo luận về kỹ thuật lan truyền ngược

- Nếu sử dụng kiến trúc mạng 1-3-1, kết quả thu được sau quá trình huấn luyện mạng như sau:
- Khi i=1,2,4: mạng đạt được đúng với hàm ban đầu
- Khi i = 8, không xấp xỉ được đúng với hàm ban đầu, hàm chỉ xấp xỉ được 1 đoạn

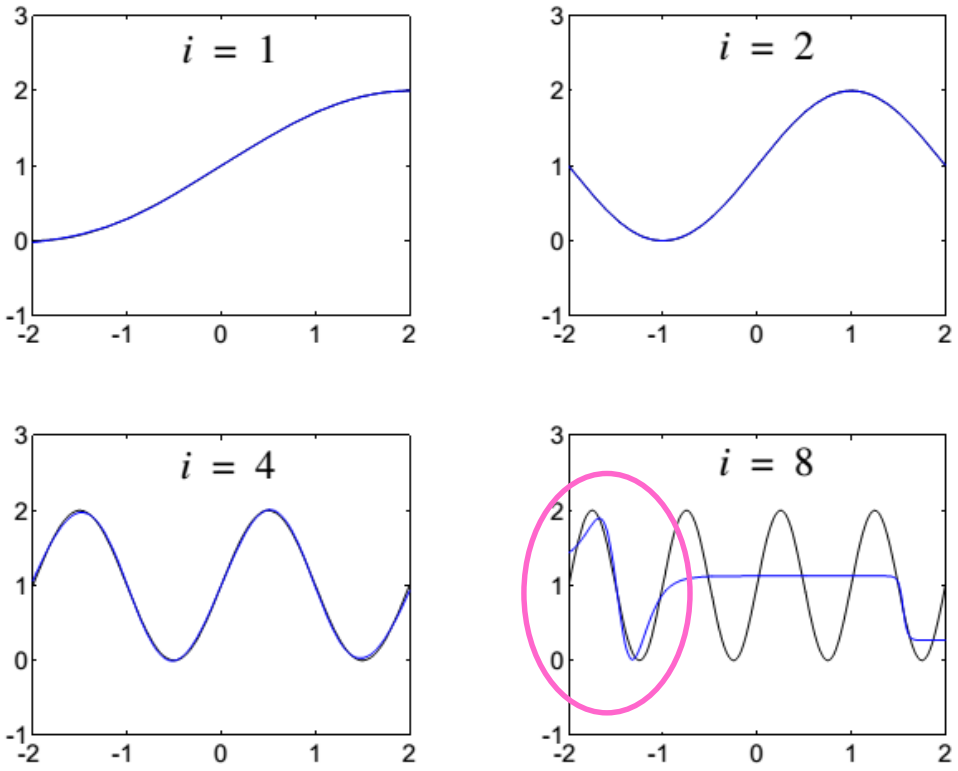


Figure 11.10 Function Approximation Using a 1-3-1 Network

Thảo luận về kỹ thuật lan truyền ngược

- I = 6
- Số neuron ở lớp ẩn tăng lên đã cho phép xấp xỉ tốt nhất hàm đã cho

$$g(p) = 1 + \sin\left(\frac{6\pi}{4}p\right) \text{ for } -2 \le p \le 2$$

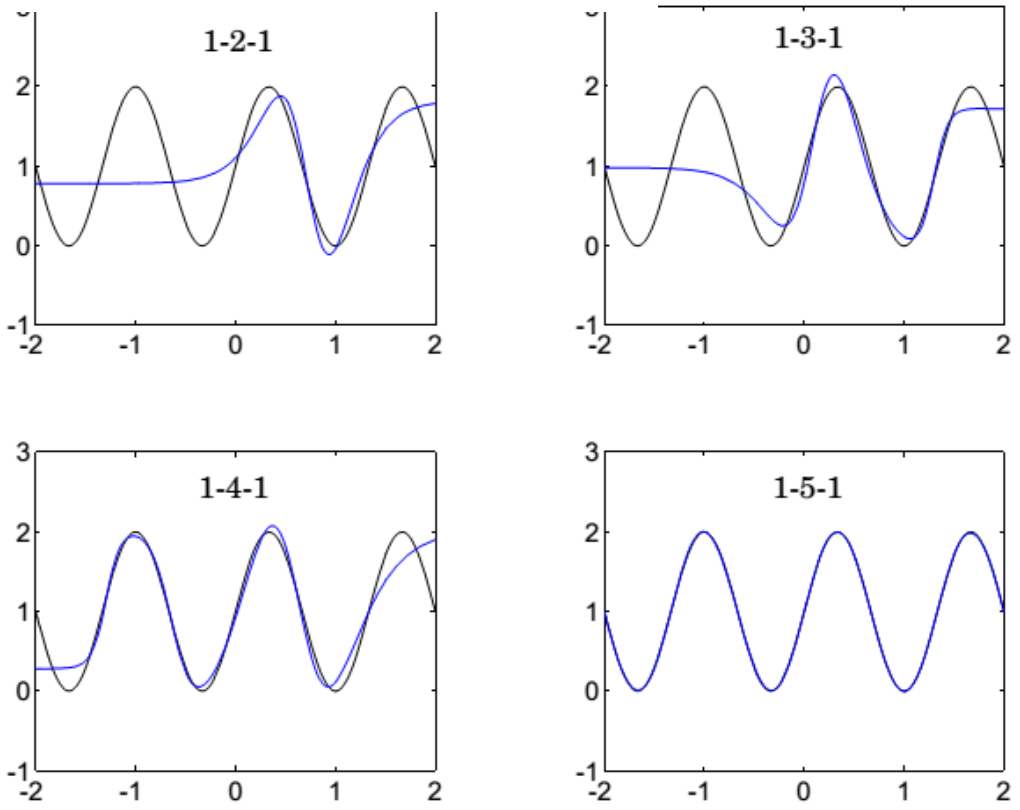


Figure 11.11 Effect of Increasing the Number of Hidden Neurons

Thảo luận về kỹ thuật lan truyền ngược

- **Độ hội tụ** $g(p) = 1 + \sin(\pi p)$ for $-2 \le p \le 2$
  - Giả sử ta muốn xấp xỉ hàm sau với kiến trúc 1-3-1

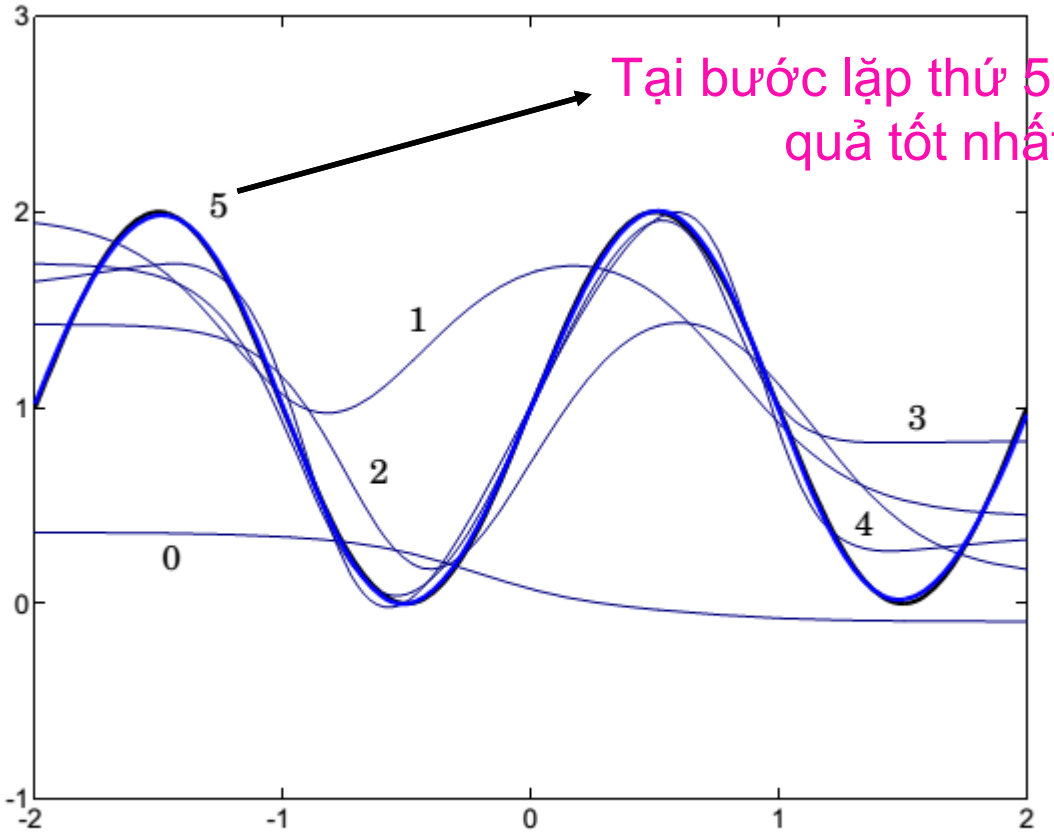


Figure 11.12 Convergence to a Global Minimum

Thảo luận về kỹ thuật lan truyền ngược

- Nếu khởi tạo không tốt, có thể rơi vào cực trị địa phương và không cho lời giải mong muốn:

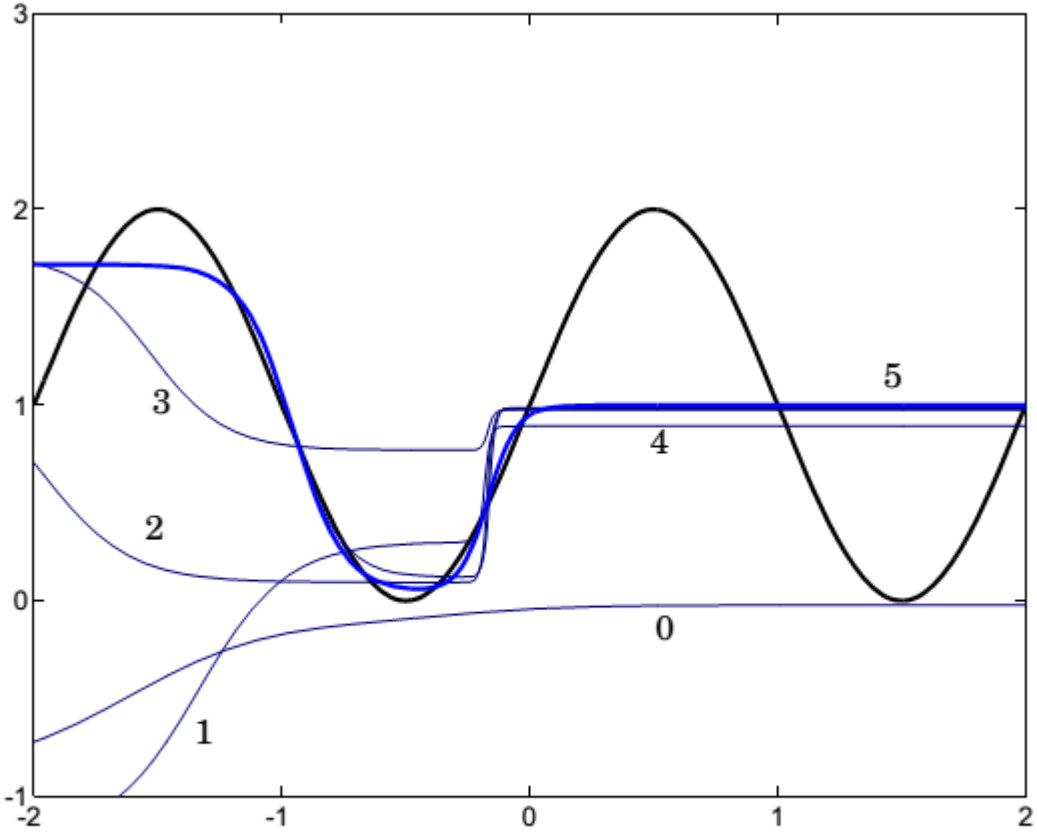
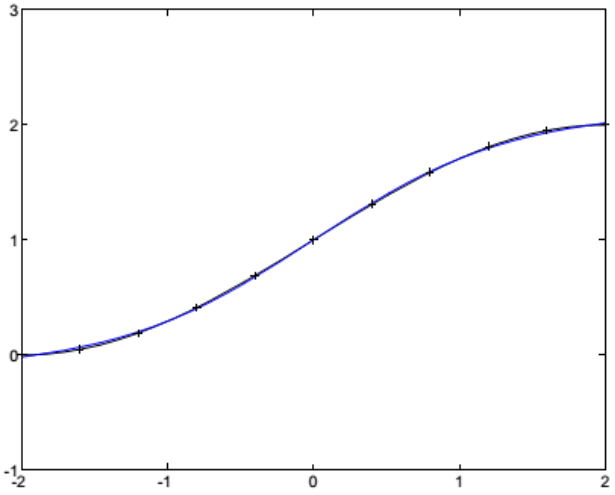


Figure 11.13 Convergence to a Local Minimum

Thảo luận về kỹ thuật lan truyền ngược

- **Tính khái quát hóa (generalization)**
  - Thông thường mạng đa lớp được huấn luyện với một số mẫu hữu hạn
  - Ví dụ để xấp xỉ hàm $g(p) = 1 + \sin\left(\frac{\pi}{4}p\right)$
  - Số mẫu là 11 trong khoảng [-2, 2]
  - Mạng có kiến trúc 1-2-1



Mạng 1-2-1 có 7 tham số, Có 11 cặp dữ liệu

Figure 11.14 1-2-1 Network Approximation of $g(p)$

Thảo luận về kỹ thuật lan truyền ngược

- **Tính khái quát hóa (generalization)**
  - Thông thường mạng đa lớp được huấn luyện với một số mẫu hữu hạn
  - Ví dụ để xấp xỉ hàm $g(p) = 1 + \sin\left(\frac{\pi}{4}p\right)$
  - Số mẫu là 11 trong khoảng [-2, 2]
  - Mạng có kiến trúc 1-9-1

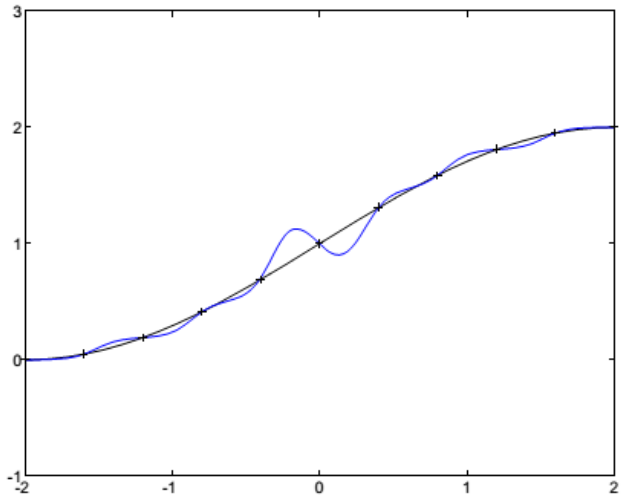


Figure 11.15 1-9-1 Network Approximation of $g(p)$

Mạng 1-9-1 có nhiều tham số: (18 weights và 10 bias

Tuy nhiên lại chỉ có 11 cặp dữ liệu

Thảo luận về kỹ thuật lan truyền ngược

- **Tính khái quát hóa (generalization)**
  - Mạng phải có số tham số ít hơn số điểm dữ liệu trong bộ huấn luyện

Don’t use a bigger network when a smaller network will work

Cross Entropy error function

$$E_{ce}(w_{ij}) = -\sum_p \sum_j \left[targ_j^p . \log\left(out_j(in_i^p)\right) + \left(1 - targ_j^p\right). \log\left(1 - out_j(in_i^p)\right) \right]$$

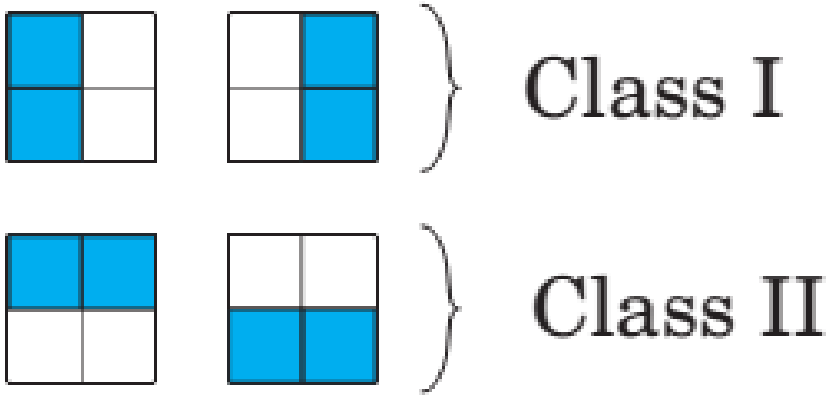
This is appropriate if we want to interpret the network outputs as probabilities, and has several advantages over the *SSE* function. When we compute the partial derivatives for the gradient descent weight update equations, the sigmoid derivative cancels out leaving

$$\Delta w_{hl}^{(N)} = \eta \sum_p \left(targ_l - out_l^{(N)} \right). out_h^{(N-1)}$$

which is easier to compute than the *SSE* equivalent, and no longer has the property of going to zero when the outputs are totally wrong (so no need for offsets, etc.).

Bài tập thảo luận

- **Cho hai lớp sau:**
  - Hai lớp có phân tách tuyến tính không ?
  - Thiết kế mạng neuron để phân tách lớp này



Bài 1

- Quy ước ô xanh có giá trị là 1, ô trắng có giá trị là -1
- Khi đó hai lớp dữ liệu sẽ có các mẫu như sau

$$\mathbf{p}_1 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \quad \mathbf{p}_2 = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \end{bmatrix}$$

$$\mathbf{p}_3 = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \quad \mathbf{p}_4 = \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \end{bmatrix}$$

Bài 1

- **Nếu dữ liệu phân tách tuyến tính, sẽ tồn tại một siêu phẳng phân tách, nghĩa là có ma trận W và b thỏa mãn:**

$$\mathbf{W}\mathbf{p}_1 + b > 0,\ \mathbf{W}\mathbf{p}_2 + b > 0,\ \mathbf{W}\mathbf{p}_3 + b < 0,\ \mathbf{W}\mathbf{p}_4 + b < 0$$

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

$$\begin{bmatrix} -w_{1,1} - w_{1,2} + w_{1,3} + w_{1,4} \end{bmatrix} > 0,$$

$$\begin{bmatrix} w_{1,1} - w_{1,2} + w_{1,3} - w_{1,4} \end{bmatrix} < 0,$$

$$\begin{bmatrix} -w_{1,1} + w_{1,2} - w_{1,3} + w_{1,4} \end{bmatrix} < 0.$$

Bài 1

- **Hai điều kiện đầu bị mâu thuẫn**

$$w_{1,1} + w_{1,2} > w_{1,3} + w_{1,4} \text{ and } w_{1,3} + w_{1,4} > w_{1,1} + w_{1,2}$$

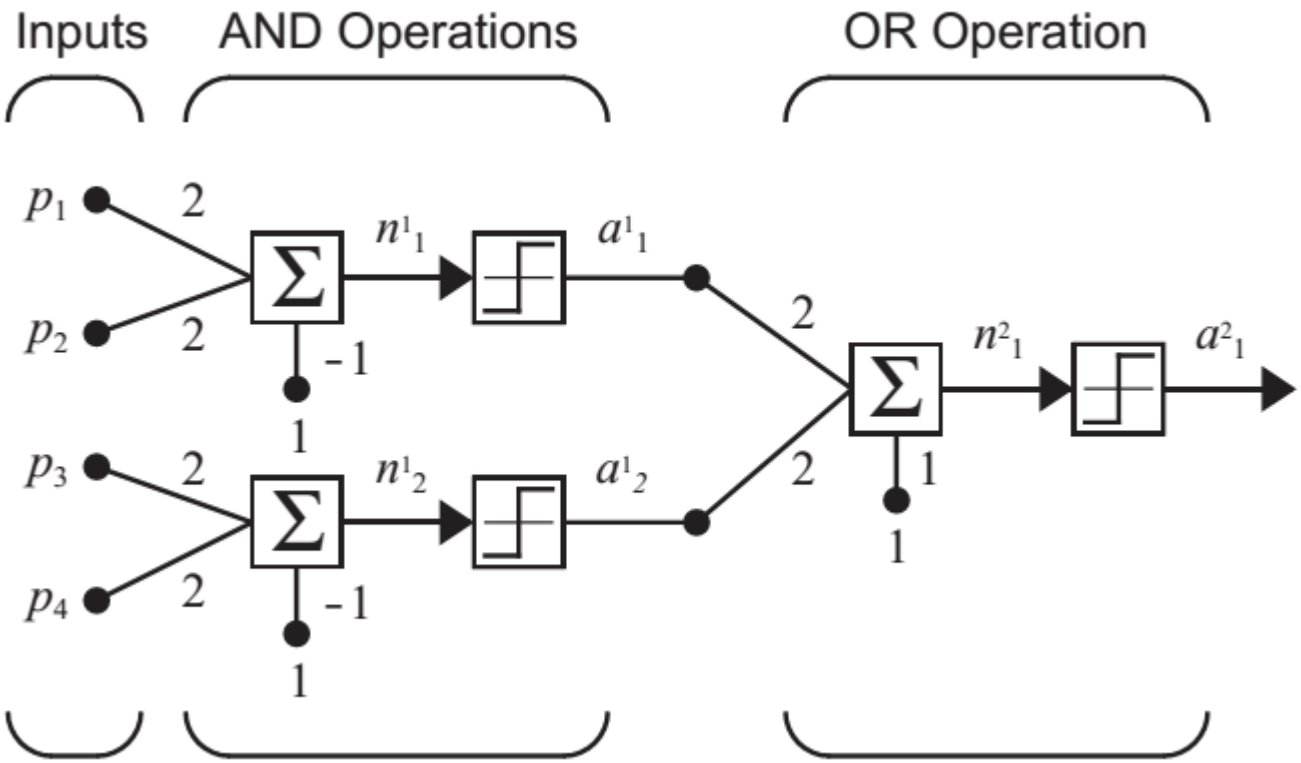
- **Hai điều kiện sau cũng bị mâu thuẫn:**

$$w_{1,1} + w_{1,3} > w_{1,2} + w_{1,4} \text{ and } w_{1,2} + w_{1,4} > w_{1,1} + w_{1,3}$$

- **Vì vậy bài toán này không phân tách tuyến tính**

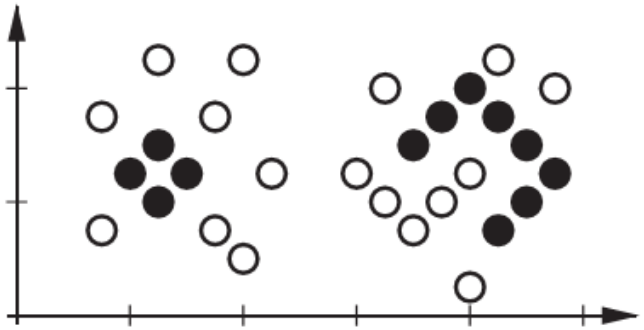
Bài 1

- Có nhiều mạng để phân tách dữ liệu này
- Đây là một ví dụ

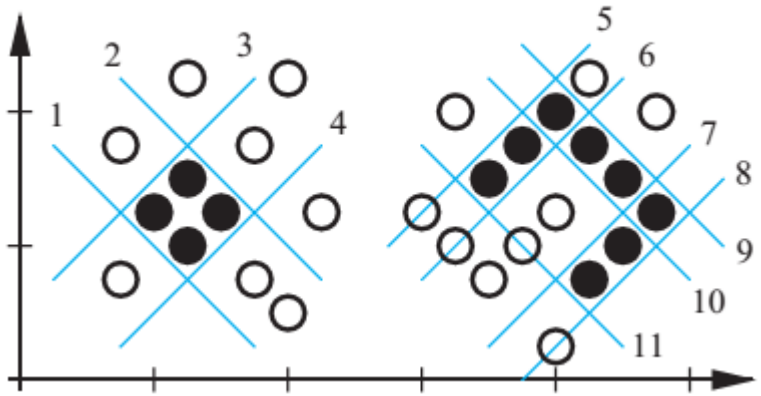


Bài tập thảo luận

- **Cho hai lớp sau:**
  - Hãy thiết kế mạng neuron để phân tách hai lớp dữ liệu này



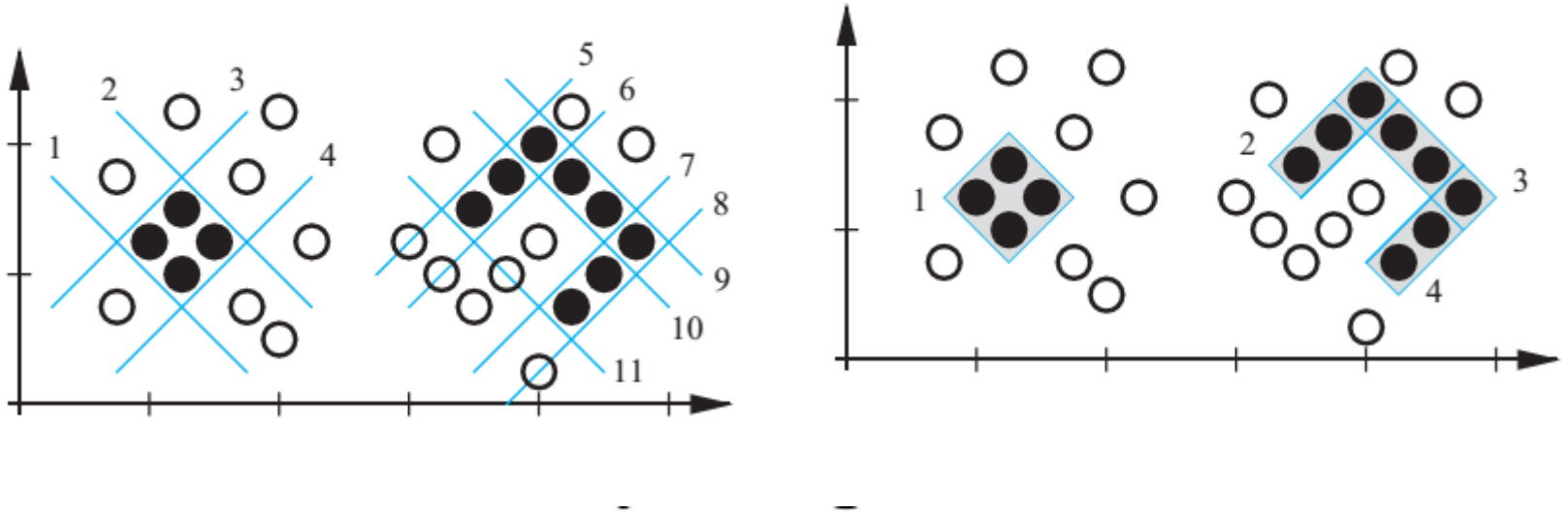
Lời giải: Tầng thứ nhất



$$(\mathbf{W}^1)^T = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & 1 \end{bmatrix},$$

$$(\mathbf{b}^1)^T = \begin{bmatrix} -2 & 3 & 0.5 & 0.5 & -1.75 & 2.25 & -3.25 & 3.75 & 6.25 & -5.75 & -4.75 \end{bmatrix}.$$

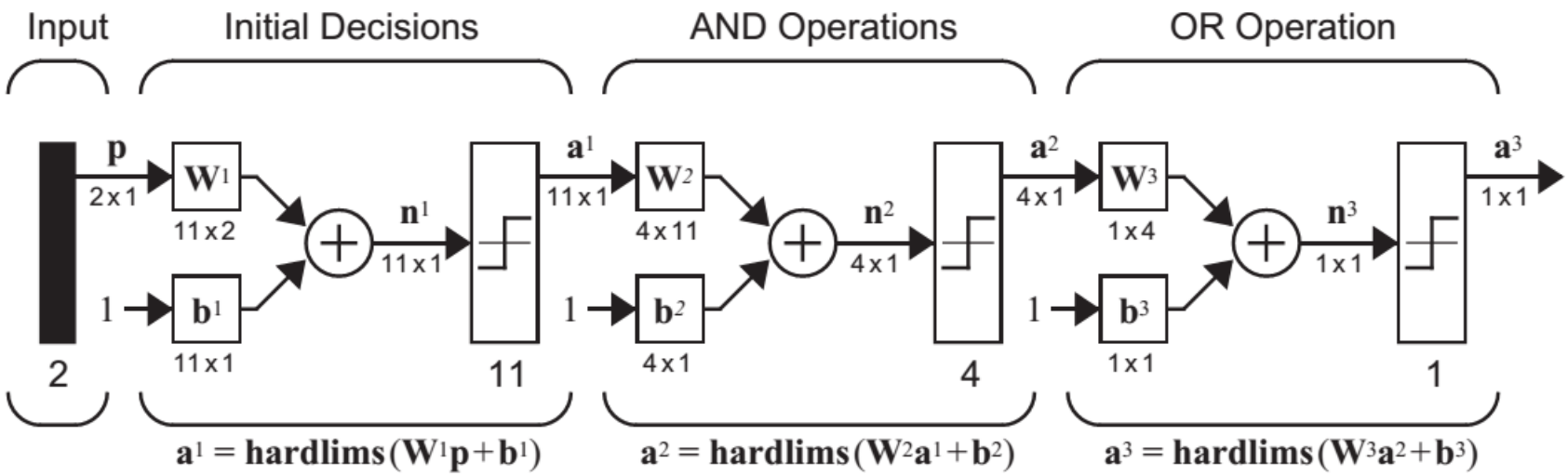
Lời giải: Tầng thứ 2



$$\mathbf{W}^2 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}, \quad \mathbf{b}^T = \begin{bmatrix} -3 \\ -3 \\ -3 \\ -3 \end{bmatrix}$$

Lời giải: tầng thứ 3

$$\mathbf{W}^3 = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}, \quad \mathbf{b}^3 = \begin{bmatrix} 3 \end{bmatrix}.$$



Thực hành matlab

- Gọi Nntoolbox trong matlab
- Viết một CT tạo 4 lớp dữ liệu, sử dụng mạng neuron 2 lớp để phân lớp 4 lớp dữ liệu này

CSDL MINIST

- **Dữ liệu: MNIST**
  - Trainning set: 60.000 images
  - Validation set: 10.000 images
  - Kích thước ảnh: 28x28

Code matlab MLP for MNIST

- **Code: https://github.com/Myasuka/matlab-mnist-two-layer-perceptron**
  - 2 layer perceptron
  - 28x28 inputs
  - 10 output units (10 different digits)
- **Stochastic training**
- **Sum square error function**
- **Error back propagation**

Hàm huấn luyện

```
function [hiddenWeights, outputWeights, error] = trainStochasticSquaredErrorTwoLayerPerceptron(activationFunction, dA
% trainStochasticSquaredErrorTwoLayerPerceptron Creates a two-layer perceptron
% and trains it on the MNIST dataset.
%
% INPUT:
% activationFunction             : Activation function used in both layers.
% dActivationFunction            : Derivative of the activation
% function used in both layers.
% numberOfHiddenUnits            : Number of hidden units.
% inputValues                    : Input values for training (784 x 60000)
% targetValues                   : Target values for training (1 x 60000)
% epochs                         : Number of epochs to train.
% batchSize                      : Plot error after batchSize images.
% learningRate                   : Learning rate to apply.
%
% OUTPUT:
% hiddenWeights                  : Weights of the hidden layer.
% outputWeights                  : Weights of the output layer.
%
```

Hàm truyền đạt và đạo hàm

```
function y = logisticSigmoid(x)
% simpleLogisticSigmoid Logistic sigmoid activation function
%
% INPUT:
% x     : Input vector.
%
% OUTPUT:
% y     : Output vector where the logistic sigmoid was applied element by
% element.
%
```

```
function y = dLogisticSigmoid(x)
% dLogisticSigmoid Derivative of the logistic sigmoid.
%
% INPUT:
% x     : Input vector.
%
% OUTPUT:
% y     : Output vector where the derivative of the logistic sigmoid was
% applied element by element.
%
```

Hàm validation

```
function [correctlyClassified, classificationErrors] = validateTwoLayerPerceptron(activationFunction, hiddenWeights,
% validateTwoLayerPerceptron Validate the twolayer perceptron using the
% validation set.
%
% INPUT:
% activationFunction             : Activation function used in both layers.
% hiddenWeights                  : Weights of the hidden layer.
% outputWeights                  : Weights of the output layer.
% inputValues                    : Input values for training (784 x 10000).
% labels                         : Labels for validation (1 x 10000).
%
% OUTPUT:
% correctlyClassified            : Number of correctly classified values.
% classificationErrors           : Number of classification errors.
%
```

Tài liệu tham khảo

- https://cs231n.github.io/neural-networks-1/
- https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b
- https://cs231n.github.io/optimization-2/
- Chương 11 của Neural Network Design