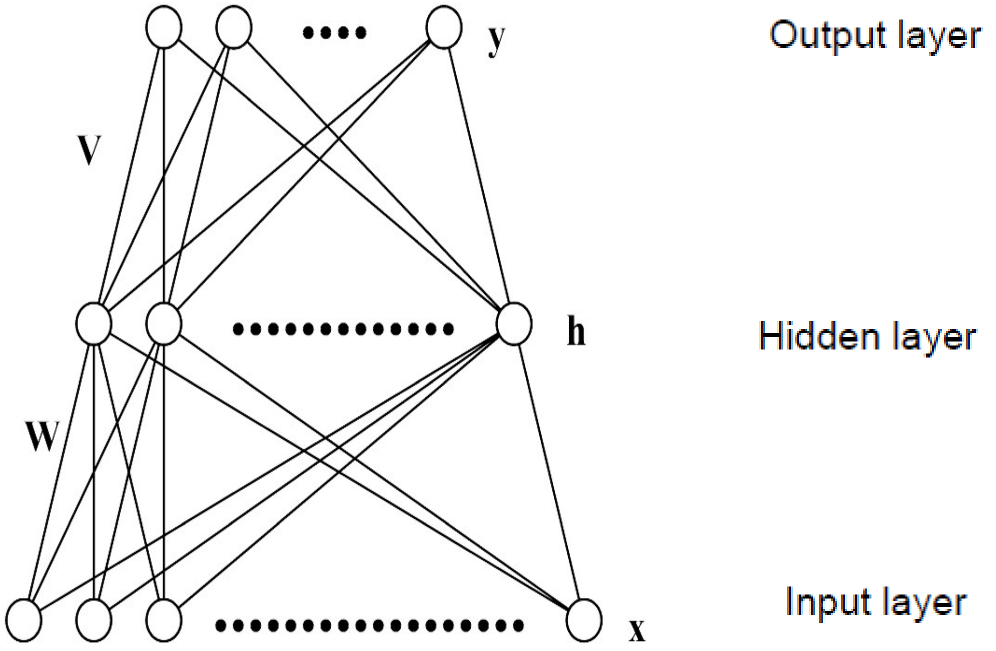
Machine Learning

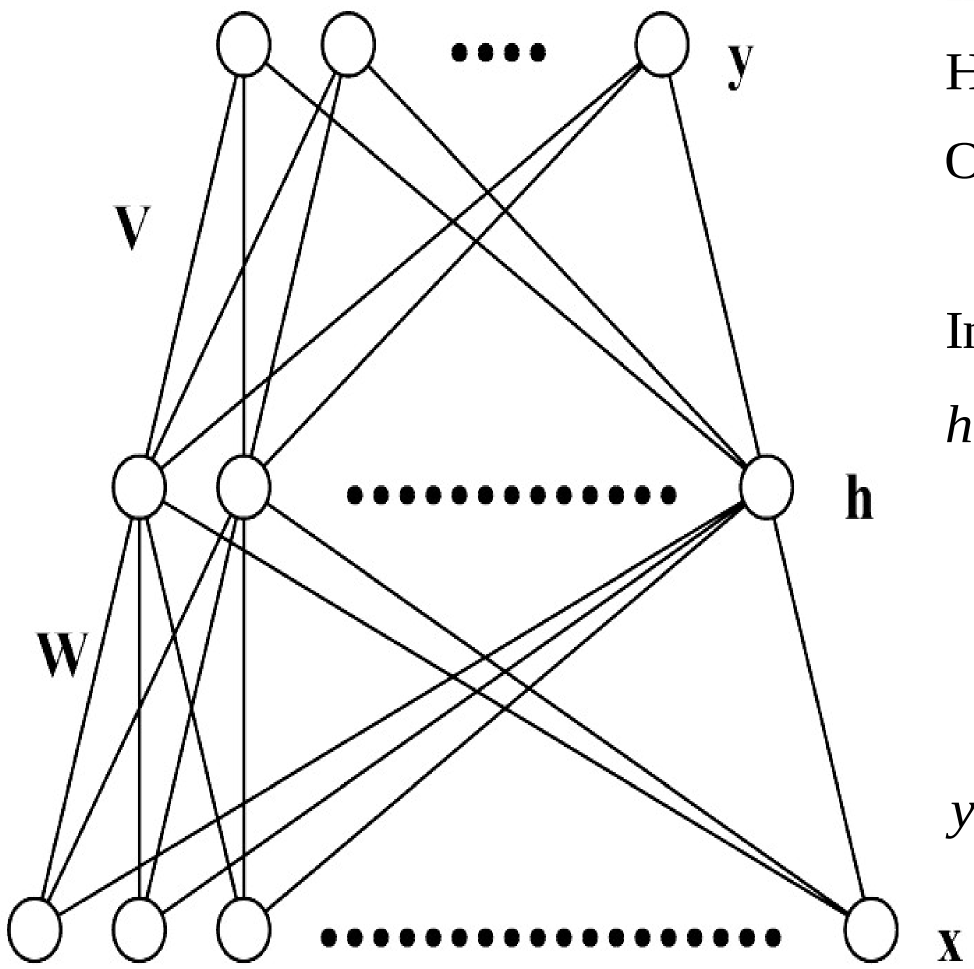
Contents

5.1. Multi-Layer Perceptron (MLP)



- Each layer receives its inputs from the previous layer and forwards its outputs to the next – feed forward structure
- Output layer: sigmoid activation function for classification, linear activation function for regression problem
- Referred to as a two-layer network (two layer of weights)

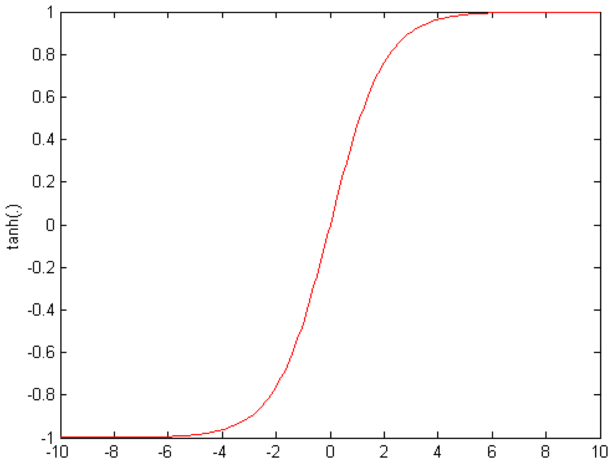
Architecture of MLP (*N-H-M*): Forward Propagation



$$\text{Input Nodes}: \mathbf{x} = [x_1, x_2, \cdots, x_N]^T$$

$$\text{Hidden Nodes}: \mathbf{h} = [h_1, h_2, \cdots, h_H]^T$$

$$\text{Output Nodes}: \mathbf{y} = [y_1, y_2, \cdots, y_M]^T$$



$$\text{Input} \longrightarrow \text{Output(?)}$$

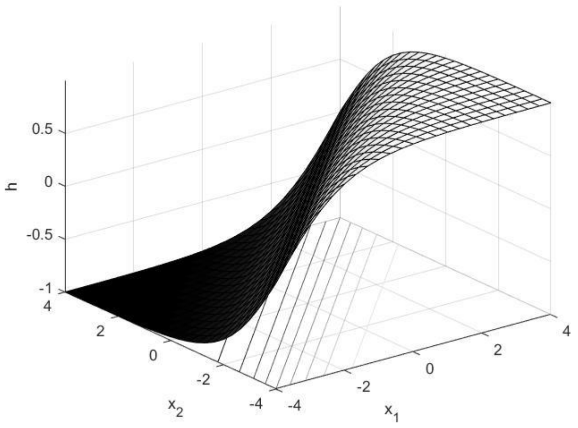
$$h_j = f(\hat{h}_j) = \tanh(\hat{h}_j / 2)$$

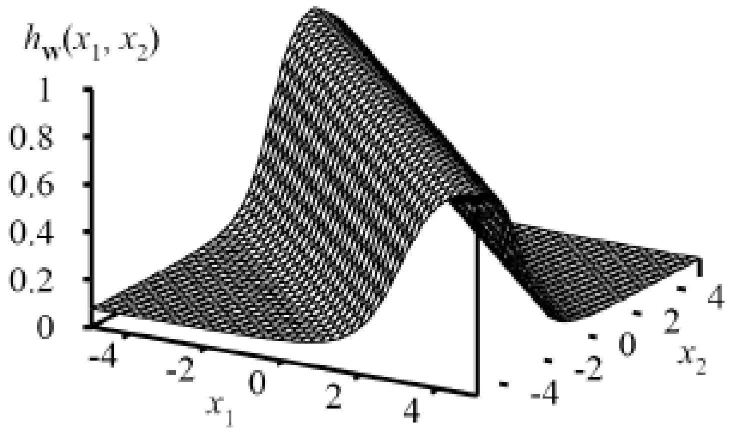
$$\text{where } \hat{h}_j = \sum_{i=1}^{N} w_{ji} x_i + w_{j0}$$

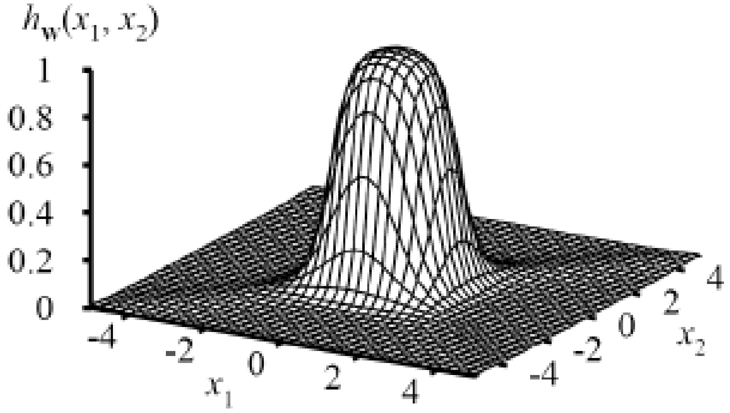
$$y_k = f(\hat{y}_k), \quad \text{where } \hat{y}_k = \sum_{j=1}^{H} v_{kj} h_j + v_{k0}$$

5.2. Representational Power of MLP

- MLP with 3 layers can represent arbitrary function
  - Each hidden unit represents a soft threshold function in the input space
  - Combine two opposite-facing threshold functions to make a ridge
  - Combine two perpendicular ridges to make a bump
  - Add bumps of various sizes and locations to fit any surface







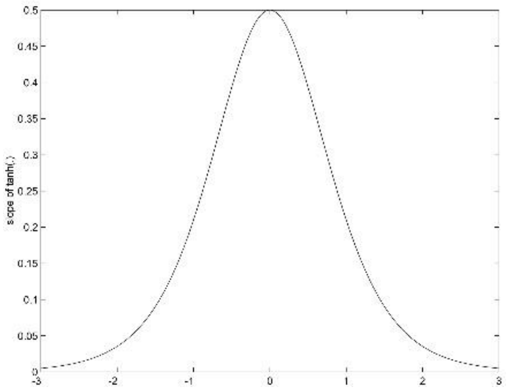
- Universal approximator
  - A two layer (linear output) network can approximate any continuous function on a compact input domain to arbitrary accuracy given a sufficiently large number of hidden units

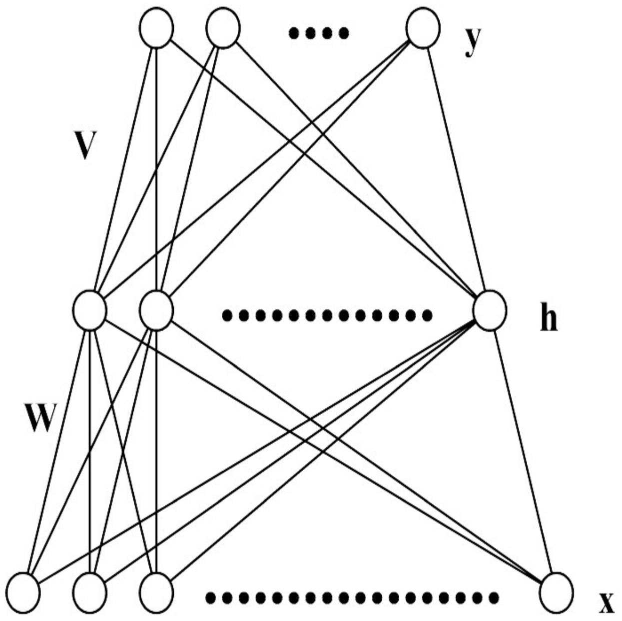
5.3. Training of MLP : Error Back-Propagation Algorithm

Input Pattern: $\mathbf{x} = [x_1, x_2, \cdots, x_N]^T$

Output Vector: $\mathbf{y} = [y_1, y_2, \cdots, y_M]^T$

Desired Output Vector: $\mathbf{t} = [t_1, t_2, \cdots, t_M]^T$





Mean-Squared Error Function:

$$E_m(\mathbf{x}) = \frac{1}{2}\sum_{k=1}^{M}(t_k - y_k)^2$$

$$\Delta v_{kj} = -\eta \frac{\partial E_m(\mathbf{x})}{\partial v_{kj}} = \eta \delta_k^{(out)} h_j \quad \text{where } \delta_k^{(out)} = -\frac{\partial E_m(\mathbf{x})}{\partial \hat{y}_k} = (t_k - y_k) f'(\hat{y}_k)$$

$$\Delta w_{ji} = -\eta \frac{\partial E_m(\mathbf{x})}{\partial w_{ji}} = \eta \delta_j^{(hid)} x_i \quad \text{where } \delta_j^{(hid)} = -\frac{\partial E_m(\mathbf{x})}{\partial \hat{h}_j} = f'(\hat{h}_j)\sum_{k=1}^{M} v_{kj}\delta_k^{(out)}$$

참조:

시그모이드 활성화 함수의 미분을 구하여 보자.

$$y = f(x) = \tanh(x/2) = \frac{e^{x/2} - e^{-x/2}}{e^{x/2} + e^{-x/2}} \tag{5.3.7}$$

이다. $u = x/2$로 치환하여

$$f'(x) = \frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx} \tag{5.3.8}$$

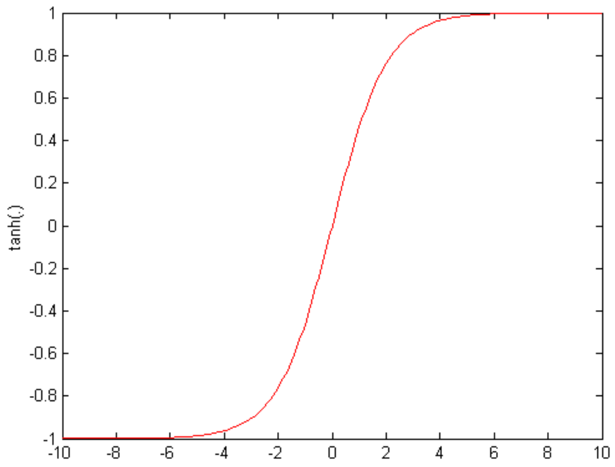
로 변경시켜서 계산하면, $du/dx = 1/2$이고

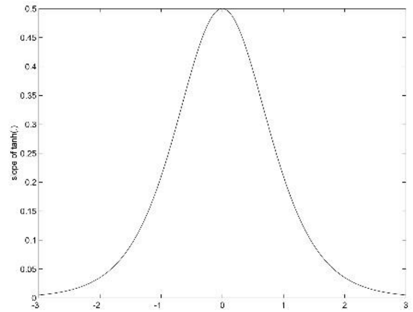
$$\frac{dy}{du} = (1-y)(1+y) \tag{5.3.9}$$

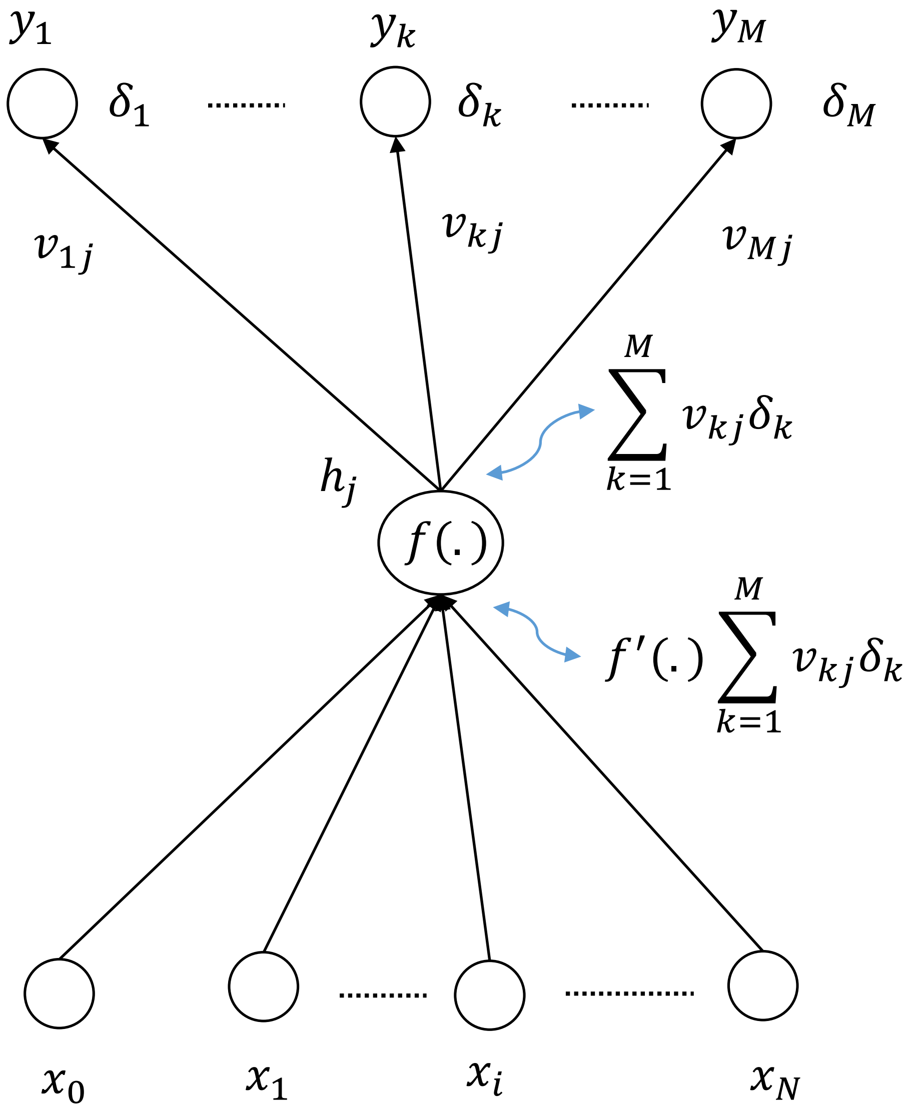
이므로

$$f'(x) = \frac{(1-y)(1+y)}{2} \tag{5.3.10}$$

이다.

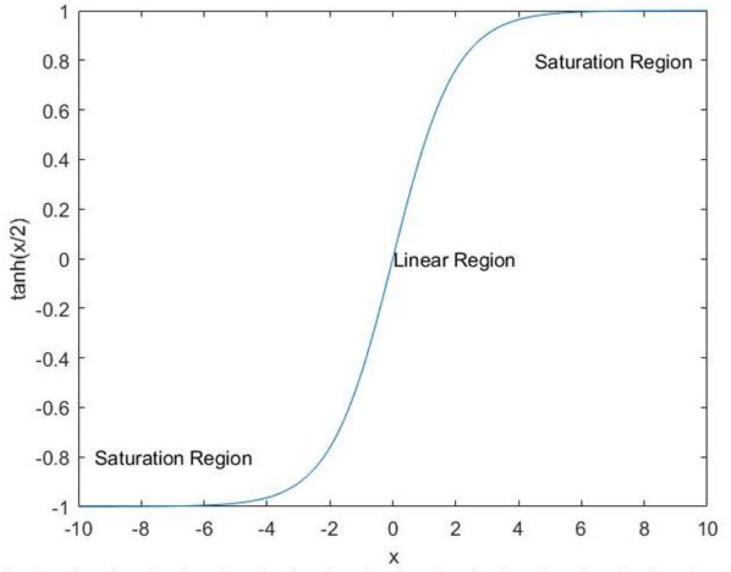






y_1
y_k
y_M
δ_1
δ_k
δ_M
v_{1j}
v_{kj}
v_{Mj}
$\sum_{k=1}^{M} v_{kj}\delta_k$
h_j
$f(.)$
$f'(.)\sum_{k=1}^{M} v_{kj}\delta_k$
x_0
x_1
x_i
x_N

5.4. Incorrect Saturation of Output Nodes



$$\delta_k \equiv -\frac{\partial E_{MSE}}{\partial \hat{y_k}} = (t_k - y_k) f'(\hat{y_k}) \quad (5.3.4)$$

Correct Saturation

$y_k \approx t_k, \delta_k \approx 0$

Incorrect Saturation

If $y_k \approx \pm 1$ and $t_k = \mp 1$, $\delta_k \approx 0$ although $|t_k - y_k| \approx 2$

Incorrect Saturation of Output Nodes → Very Slow Convergence of Learning due to $\delta_k \approx 0$

Training of MLP : Error Back-Propagation Algorithm

< conv. MSE >

$$\delta_k^{out}(\mathbf{x}) = (t_k - y_k) f'(\hat{y}_k)$$

. Incorrect Saturation Problem

$$E_m(\mathbf{x}) = \frac{1}{2}\sum_{k=1}^{M}(t_k - y_k)^2$$

< Cross-Entropy Error >

$$\delta_k^{out}(\mathbf{x}) = (t_k - y_k)$$

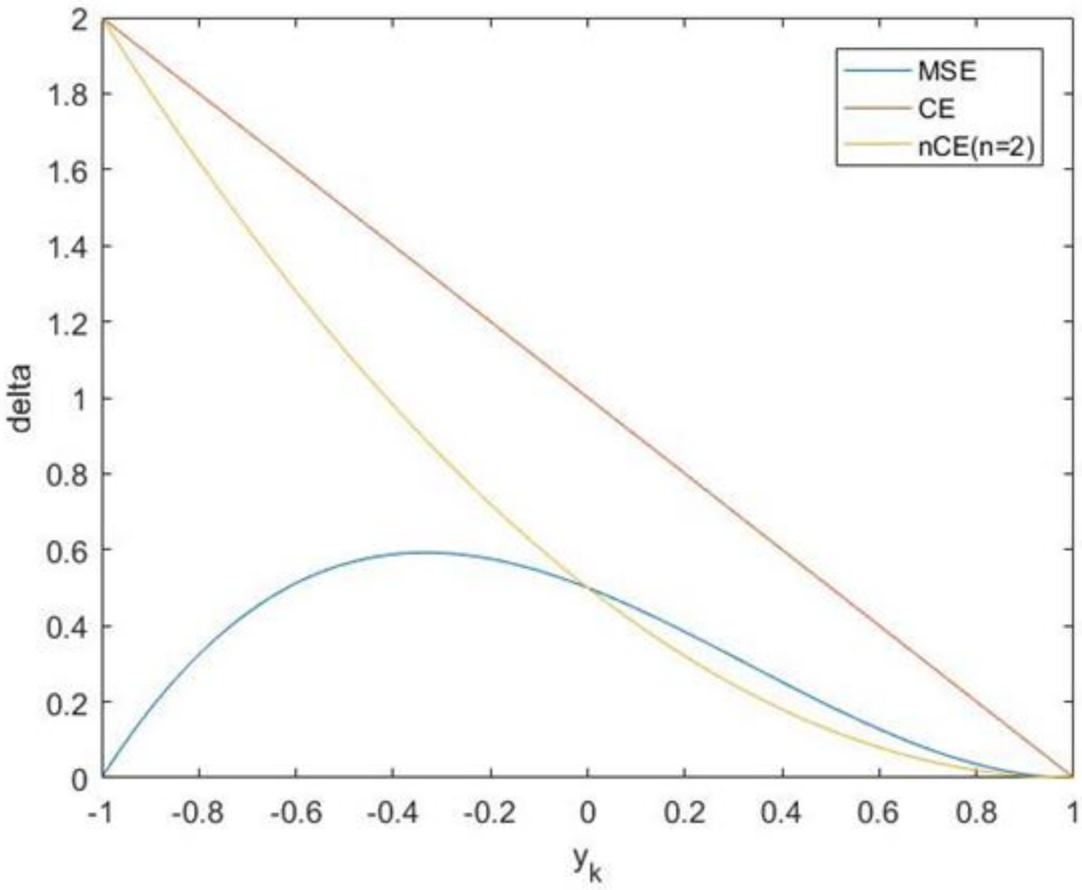
. Overspecialization Problem

$$E_{CE}(\mathbf{x}) = -\sum_{k=1}^{M}\left[(1+t_k)\ln\left(1+y_k(\mathbf{x})\right) + (1-t_k)\ln\left(1-y_k(\mathbf{x})\right)\right]$$

< n-th order Extension of CE >

$$\delta_k^{out}(\mathbf{x}) = \frac{t_k^{\ n+1}(t_k - y_k)^n}{2^{n-1}}$$

$$E_{nCE} = -\sum_{k=1}^{M}\int \frac{t_k^{n+1}(t_k - y_k)^n}{2^{n-2}(1-y_k)(1+y_k)}dy_k$$



Plot of $\delta_k^{out}(\mathbf{x}^{(p)})$ when $t_k = 1$

Error Back-Propagation Algorithm

오류역전파(EBP) 알고리즘

① 다층퍼셉트론의 가중치들을 초기화 시킨다.

② 학습패턴 $\boldsymbol{x}$가 주어지면, 전방향 계산에 의해 출력노드 값을 구한다.

③ 출력노드의 오류 신호 δ_k를 계산한다.

④ 은닉노드의 오류신호 $\delta_j^{(hidden)}$을 역전파 방법에 의해 계산한다.

⑤ 출력층 가중치와 은닉층 가중치의 변경량을 계산한다.

⑥ 출력층 가중치와 은닉층 가중치를 변경시킨다.

참조: 오차함수의 통계적 의미

오차함수의 통계적 의미를 분류문제의 경우에 대하여 유도하여 보자. 분류문제에서 목표 값은

$$t_k = \begin{cases} +1 & \text{if } \mathbf{x} \text{ originats from class k} \\ -1 & \text{otherwise} \end{cases} \tag{5.4.6}$$

로 주어진다. 그러면, 학습패턴의 수가 무한대로 갈 경우 오차함수 E_{MSE}를 최소화 하는 다층퍼셉트론은

$$E[E_{MSE}] = E\left[\frac{1}{2}\sum_{k=1}^{M}(T_k - y_k(\boldsymbol{X}))^2\right] \tag{5.4.7}$$

을 최소화 시키는 다층퍼셉트론으로 수렴한다. 여기서, $E[.]$는 기대치(Expectation) 연산자이고, T_k는 목표값 확률변수이고, $\boldsymbol{X}$는 입력을 나타내는 확률벡터이다. 목표값이 식 (5.4.6)과 같이 코딩되어 있기 때문에 식 (5.4.7)에서

$$E\left[\sum_{k=1}^{M}(T_k - y_k(\boldsymbol{X}))^2\right] = \int \left[(1-y_k(\boldsymbol{x}))^2 Q_k(\boldsymbol{x}) + (-1-y_k(\boldsymbol{x}))^2(1-Q_k(\boldsymbol{x}))\right] f(\boldsymbol{x})d\boldsymbol{x} \tag{5.4.8}$$

로 된다. 여기서, $Q_k(\boldsymbol{X}) = \Pr[\boldsymbol{X} \text{ originates from class k} | \mathbf{X} = \mathbf{x}]$ 이고, $f(\boldsymbol{x})$는 $\boldsymbol{x}$의 확률밀도함수이다. 식 (5.4.8)을 최소화 시키는 함수 $b_k(\boldsymbol{X})$를 $(-1,+1)$ 구간의 함수 공간에서 찾아보자. 고정된 $Q_k(\boldsymbol{X}), 0 < Q_k(\boldsymbol{X}) < 1$,에 대하여 최적 해 $b_k(\boldsymbol{X})$는 식 (5.4.8)의 $y_k(\boldsymbol{x})$에 대한 미분이 0이 되는 조건에 의해

$$b_k(\boldsymbol{X}) = 2Q_k(\boldsymbol{X}) - 1,\ k = 1, 2, \ldots, M \tag{5.4.9}$$

으로 구해진다.

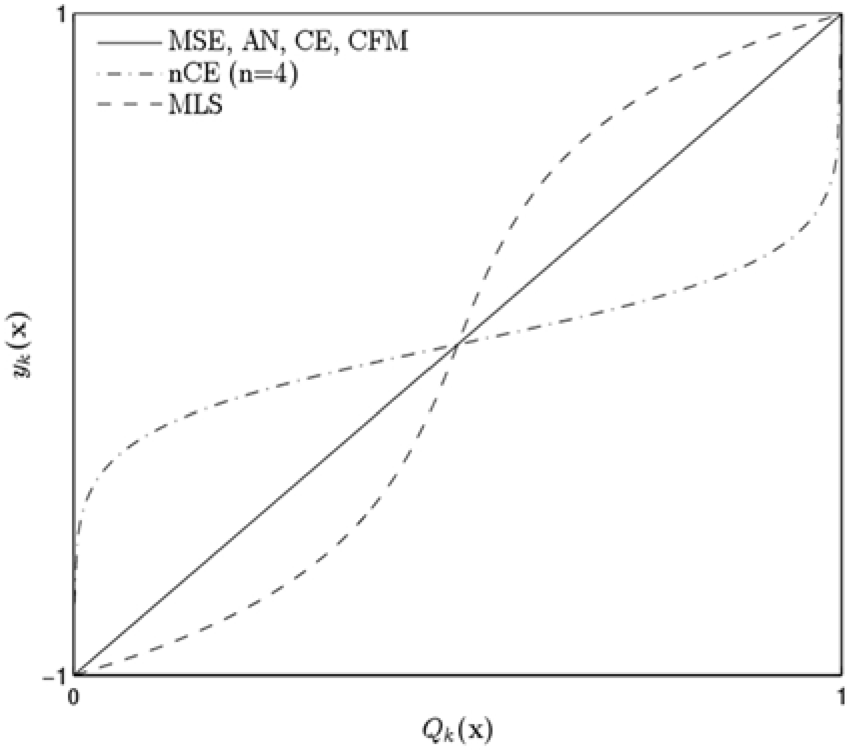
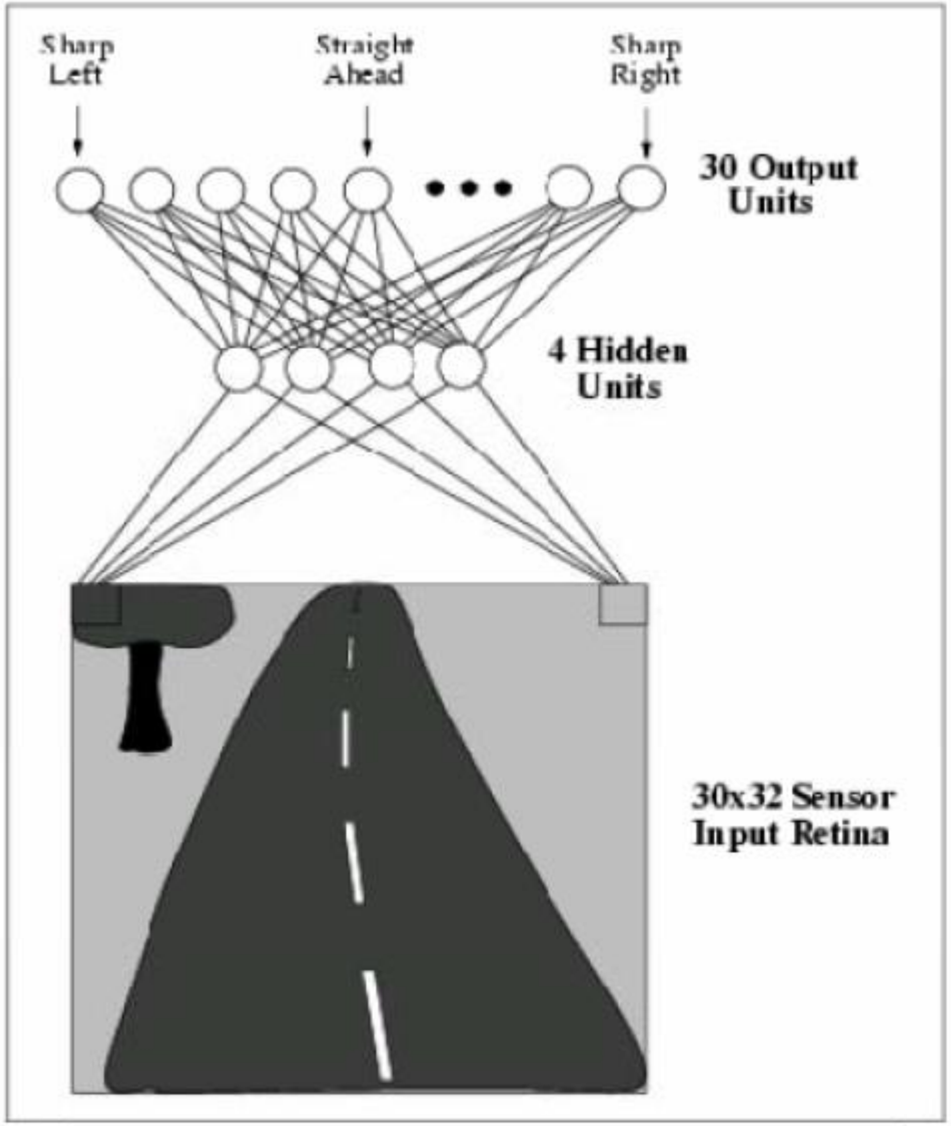
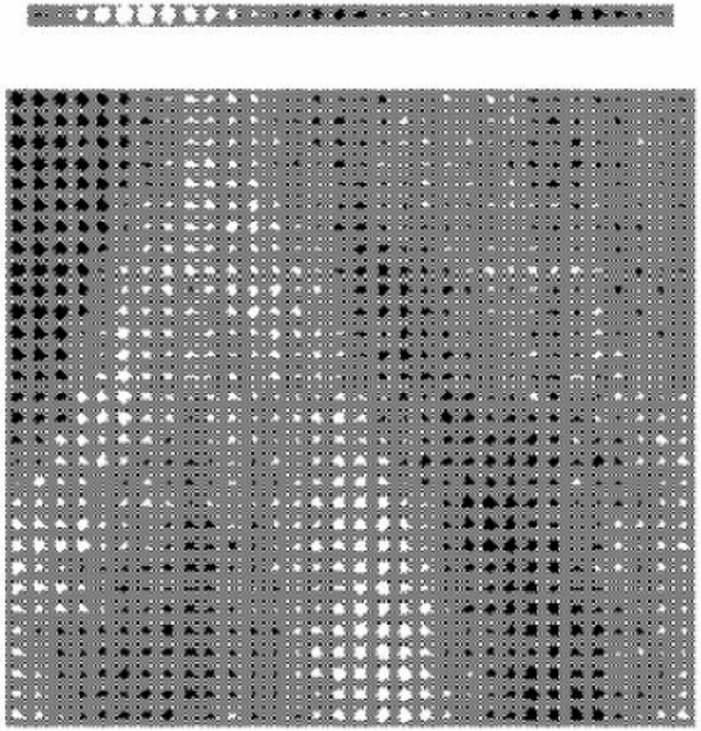


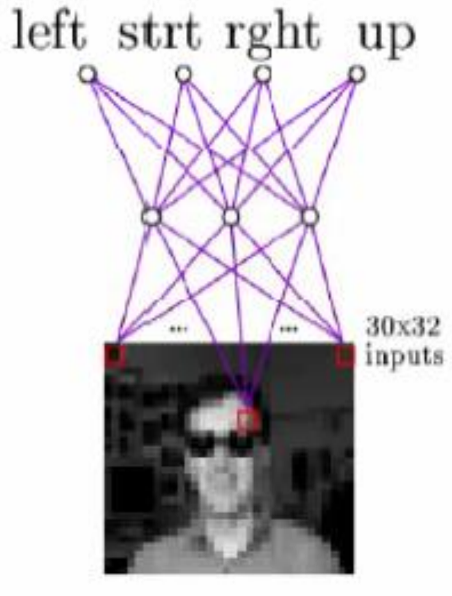
그림 5.6. 오차함수의 최적 해에 해당하는 출력

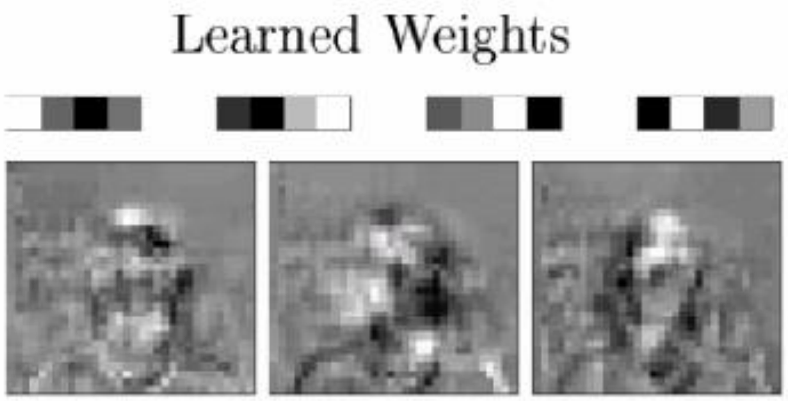
Example


Sharp Left
Straight Ahead
Sharp Right
30 Output Units
4 Hidden Units
30x32 Sensor Input Retina

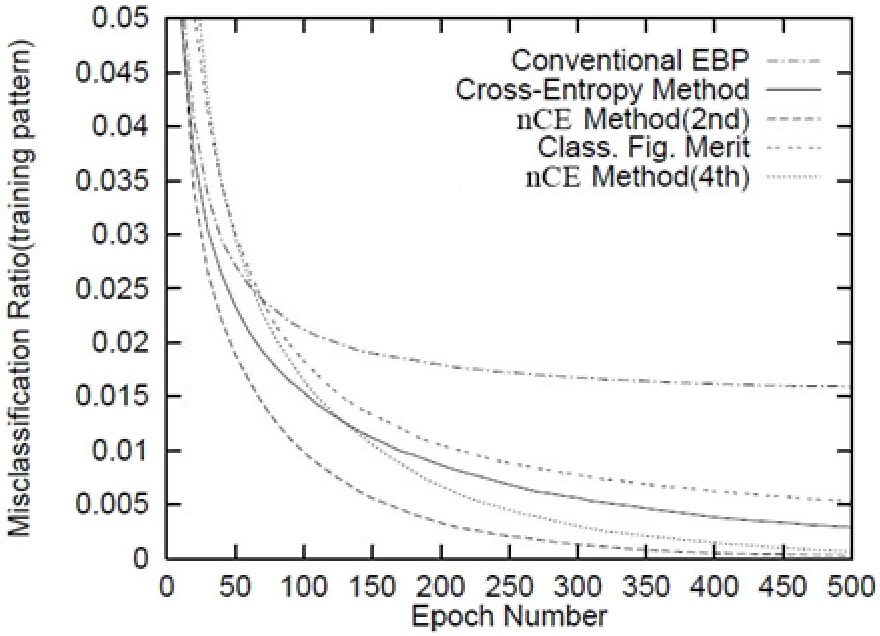


Example

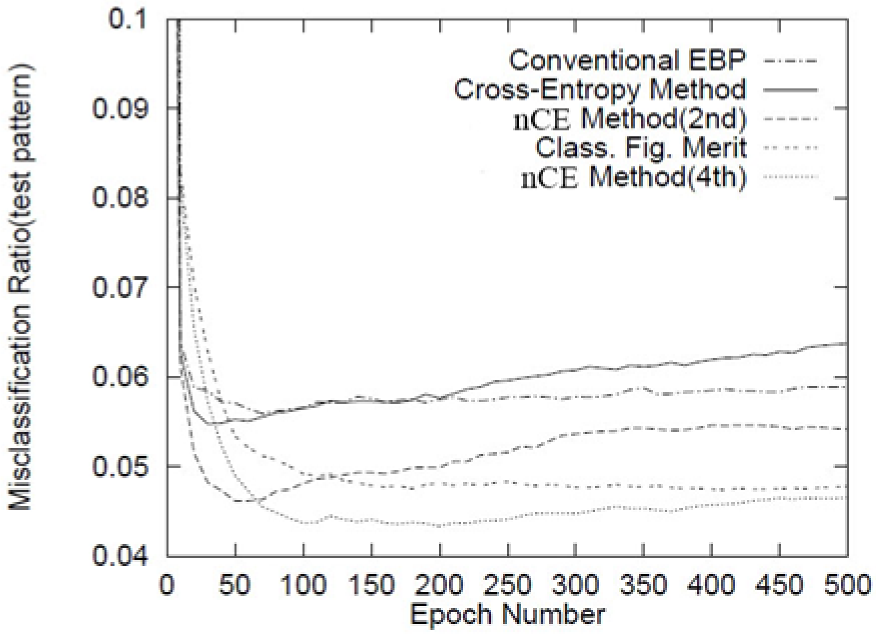




Typical input images



(a) 학습패턴에 대한 오인식률

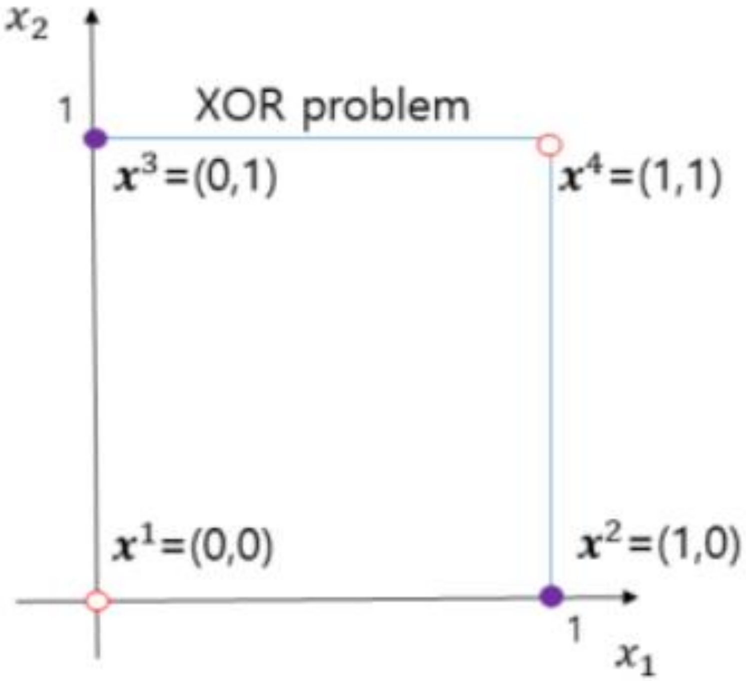


(b) 시험패턴에 대한 오인식률

그림 5.7. 필기체 숫자인식 문제의 학습 시뮬레이션 결과

예제 5.4-1

아래 그림으로 2차원 공간 상에 주어진 XOR 문제를 입력 2, 은닉노드 4, 출력 1개의 노드를 지닌 다층퍼셉트론으로 학습하기 위하여 초기 가중치를 $\boldsymbol{W} = \begin{pmatrix} w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \\ w_{30} & w_{31} & w_{32} \\ w_{40} & w_{41} & w_{42} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0.5 \\ 0 & 1 & 0.9 \\ 0 & 0.9 & 1 \\ 0 & 0.5 & 1 \end{pmatrix}$와 $\boldsymbol{v} = (v_0, v_1, v_2, v_3, v_4) = (0, 0.3, 0.6, 0.9, 1.2)$로 초기화 하였다고 가정하자. 출력 목표값은 입력 $\boldsymbol{x}^1$과 $\boldsymbol{x}^4$에 대해서만 -1이고 입력 $\boldsymbol{x}^2, \boldsymbol{x}^3$에 대해서는 1이다. 4개의 입력 중 임의의 하나를 골라서 다층퍼셉트론에 입력하여 MSE, CE, nCE(n=2) 오차함수에 따른 출력노드와 첫 은닉노드의 오류신호를 구하여 보아라.



풀이

$\boldsymbol{x}^3 = (0,1)^T$이 입력되었다고 하자, 4개의 은닉노드에 대한 가중치 합은 식 (5.1.1)에 의해 $\hat{h}_1 = 0.5,\ \hat{h}_2 = 0.9,\ \hat{h}_3 = 1,\ \hat{h}_4 = 1$과 같이 계산되고, 식 (5.1.2)에 의해 $h_j = \tanh(\hat{h}_j/2)$ 활성화함수를 통과한 은닉노드의 출력값은 $h_1 = 0.2449,\ h_2 = 0.4219,\ h_3 = 0.4621,$ $h_4 = 0.4621$와 같이 계산된다. 그러면 출력노드에 대한 가중치 합 역시 식 (5.1.3)에 따라 $\hat{y} = (0,\ 0.3,\ 0.6,\ 0.9, 1.2)(1,\ 0.2449,\ 0.4219,\ 0.4621, 0.4621)^T = 1.2970$이고 출력은 식 (5.1.4)에 의해 $y = \tanh(\hat{y}/2) = 0.5707$이다.

출력노드의 시그모이드 기울기 항은 $f'(\hat{y}) = (1-y)(1+y)/2 = 0.3372$이고, h_1의 시그모이드 기울기 항은 $f'(\hat{h}_1) = (1-h_1)(1+h_1)/2 = 0.4700$이다. 따라서, 각 오차함수에 대한 출력노드의 오류신호는 식 (5.4.2), (5.3.4), (5.4.4)에 따르고, 은닉노드의 오류신호는 식 (5.3.14)에 따라 다음과 같이 주어진다.

$$
\begin{aligned}
&\text{CE}: \delta_{CE} = t - y = 0.4293 \\
&\qquad \delta_1^{(hidden)} = f'(\hat{h}_1) \times v_1 \times \delta_{CE} = 0.47 \times 0.3 \times 0.4293 = 0.0605
\end{aligned}
$$

$$\text{M.S.E.}: \delta_{MSE} = (t-y)f'(\hat{y}) = 0.4293 \times 0.3372 = 0.1448,\ \ \delta_1^{(hidden)} = 0.0204$$

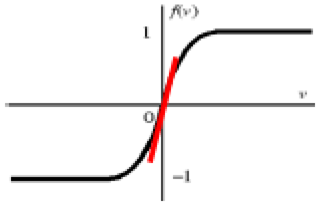
$$\text{nCE}(\text{n}=2): \delta_{nCE} = (t-y)^2/2 = 0.0921,\ \ \delta_1^{(hidden)} = 0.0130$$

5.5. Remarks on Training

- No guarantee of convergence, may oscillate or reach a local minima.
- However, in practice many large networks can be adequately trained on large amounts of data for realistic problems, e.g.
  - Driving a car
  - Recognizing handwritten zip codes
  - Play world championship level Backgammon
- Many epochs (thousands) may be needed for adequate training, large data sets may require hours or days of CPU time.
- Termination criteria can be:
  - Fixed number of epochs
  - Threshold on training set error
  - Increased error on a validation set
- To avoid local minima problems, can run several trials starting from different initial random weights and:
  - Take the result with the best training or validation performance.
  - Build a committee of networks that vote during testing, possibly weighting vote by training or validation accuracy

Notes on Proper Initialization

- Start in the “linear” regions
  - keep all weights near zero, so that all sigmoid units are in their linear regions. Otherwise nodes can be initialized into flat regions of the sigmoid causing for very small gradients



- Break symmetry
  – If we start with the weights all equal, what will happen?
  – Ensure that each hidden unit has different input weights so that the hidden units move in different directions.
- Set each weight to a random number in the range

$$[-1, +1] \times \frac{1}{\sqrt{\text{fan-in}}}$$

where “fan-in” is the number of inputs to the unit.

Batch, Online, and Online with Momentum

- **Batch**: Sum the gradient for each example *i*. Then take a gradient descent step.
- **Online**: Take a gradient descent step with each input as it is computed (this is the algorithm we described)
- **Momentum factor**: Make the *t+1*-th update dependent on the *t*-th update

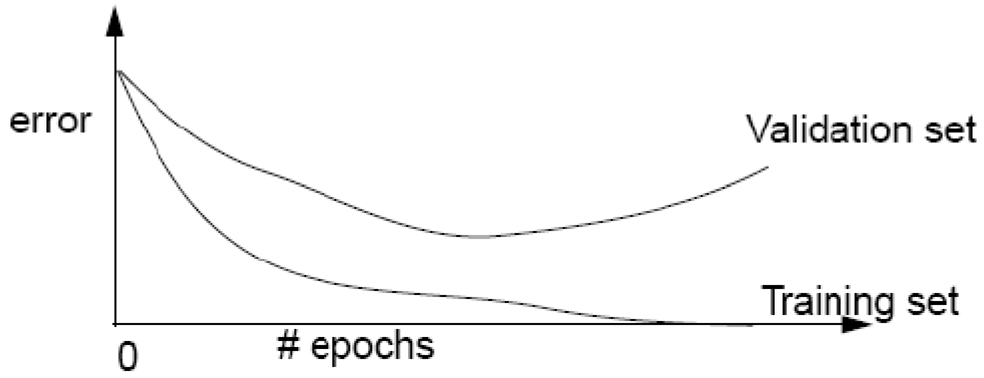
$$\Delta v_{kj}(t) = \eta \delta_k^{(out)}(\mathbf{x}^{(p)}) h_j^{(p)} + \alpha \Delta v_{kj}(t-1)$$

$$\Delta w_{ji}(t) = \eta \delta_j^{hid}(\mathbf{x}^{(p)}) x_i^{(p)} + \alpha \Delta w_{ji}(t-1)$$

α is called the momentum factor, and typically take values in the range [0.7, 0.95]. This tends to keep weight moving in the same direction and improves convergence..

Overtraining Prevention

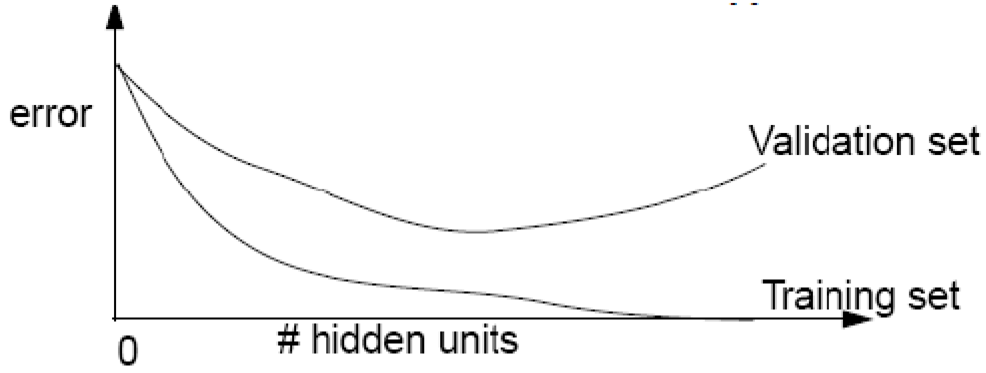
- Running too many epochs may overtrain the network and result in overfitting. Tries too hard to exactly match the training data.



- Keep a validation set and test accuracy after every epoch. Maintain weights for best performing network on the validation set and return it when performance decreases significantly beyond this.
- To avoid losing training data to validation:
  - Use 10-fold cross-validation to determine the average number of epochs that optimizes validation performance.
    - We will discuss cross-validation later in the course
  - Train on the full data set using this many epochs to produce the final result.

Over-fitting Prevention

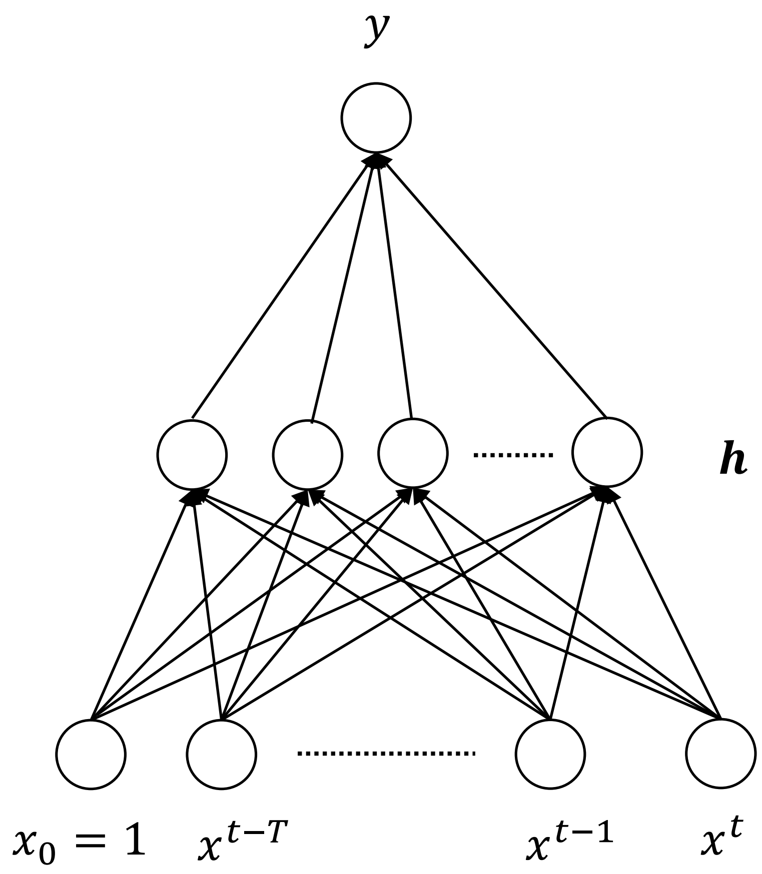
- Too few hidden units prevent the system from adequately fitting the data and learning the concept.
- Too many hidden units leads to over-fitting.



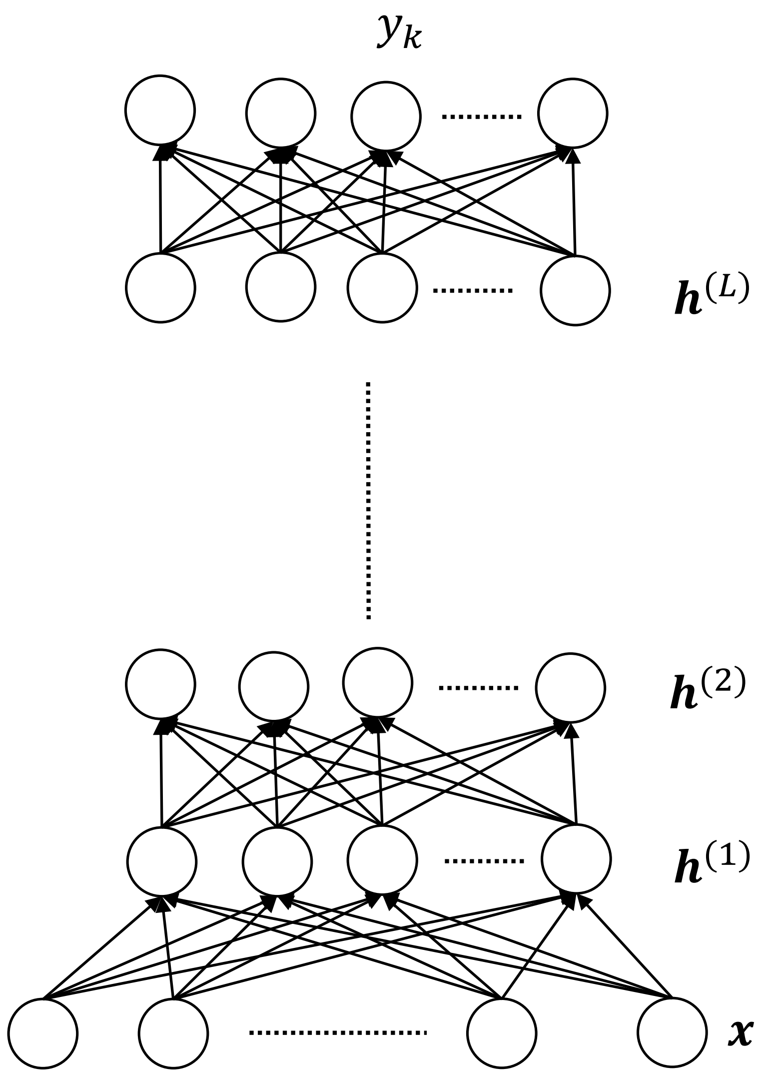
- Can also use a validation set or cross-validation to decide an appropriate number of hidden units.
- Another approach to preventing over-fitting is ***weight decay***, in which we multiply all weights by some fraction between 0 and 1 after each epoch.
  - Encourages smaller weights and less complex hypotheses.
  - Equivalent to including an additive penalty in the error function proportional to the sum of the squares of the weights of the network.

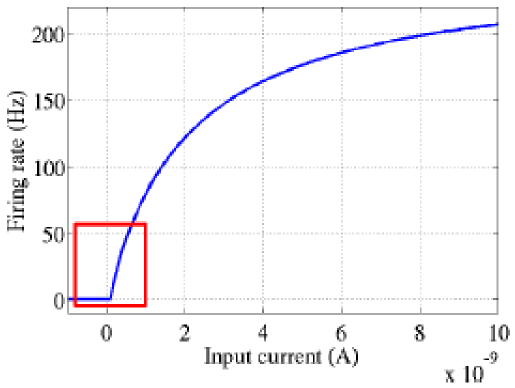
5.6. Learning Time Series Data

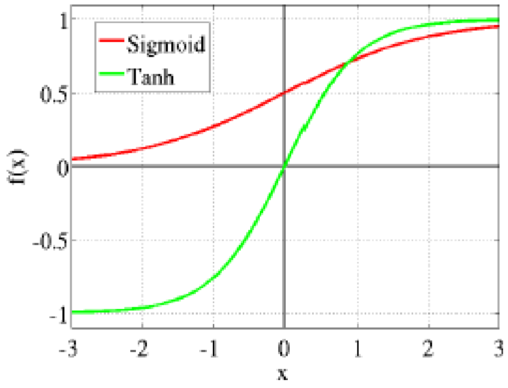
- Time-delay neural networks (TDNN)



5.7. Deep Neural Networks







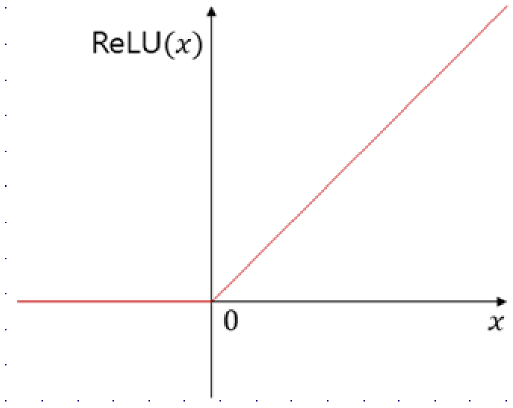
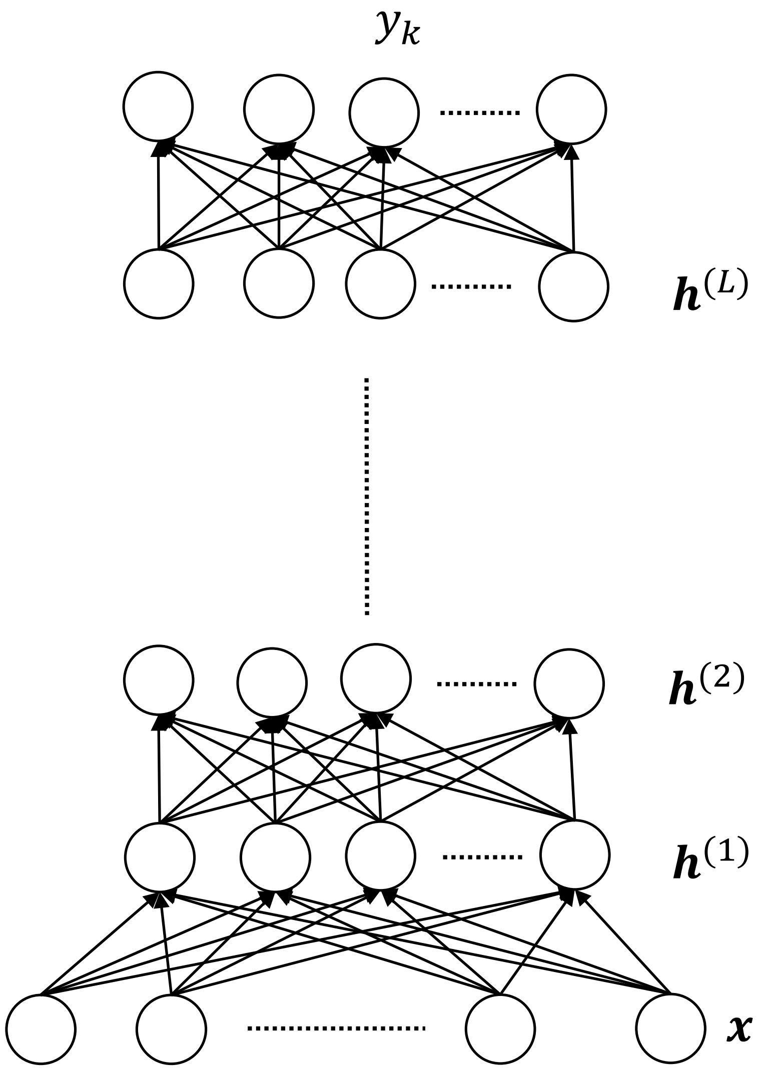


그림 5.10. ReLU 활성화 함수



$$f(x) = \max(0, x) = \begin{cases} x & \text{if} \quad x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.7.1}$$

$$f'(x) = \begin{cases} 1 & \text{if} \quad x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.7.2}$$

$$y_k = \frac{e^{\hat{y_k}}}{\sum_j e^{\hat{y_j}}} \tag{5.7.3}$$

$$E_{CE} = -\sum_k t_k \log(y_k) \tag{5.7.4}$$

$$\partial y_k / \partial \hat{y_j} = \begin{cases} y_k(1 - y_k) & \text{if} \ j = k \\ -y_k y_j & \text{otherwise} \end{cases} \tag{5.7.5}$$

$$\frac{\partial E_{CE}}{\partial \hat{y_k}} = y_k - t_k \tag{5.7.6}$$

$$\delta_k = -\frac{\partial E_{CE}}{\partial \hat{y_k}} = t_k - y_k \tag{5.7.7}$$

참조: SoftMax의 미분

식 (5.7.3)으로 주어진 SoftMax 함수의 미분을 구해보자. 먼저

$$\frac{\partial y_k}{\partial \hat{y}_k} = \frac{\partial \dfrac{e^{\hat{y}_k}}{\sum_j e^{\hat{y}_j}}}{\partial \hat{y}_k} = \frac{e^{\hat{y}_k}\sum_j e^{\hat{y}_j} - e^{\hat{y}_k}e^{\hat{y}_k}}{\left[\sum_j e^{\hat{y}_j}\right]^2} \tag{5.7.8}$$

이 된다. 여기서,

$$\frac{d}{dx}\frac{f(x)}{g(x)} = \frac{f'(x)g(x) - f(x)g'(x)}{g^2(x)} \tag{5.7.9}$$

를 사용하였다. 식 (5.7.3)을 이용하여 식 (5.7.8)을 정리하면

$$\frac{\partial y_k}{\partial \hat{y}_k} = y_k(1-y_k) \tag{5.7.10}$$

이다. 또한, $k \neq i$ 일때

$$\frac{\partial y_k}{\partial \hat{y}_i} = \frac{\partial \dfrac{e^{\hat{y}_k}}{\sum_j e^{\hat{y}_j}}}{\partial \hat{y}_i} = \frac{-e^{\hat{y}_k}e^{\hat{y}_i}}{\left[\sum_j e^{\hat{y}_j}\right]^2} = -y_k y_i \tag{5.7.11}$$

이다. 이를 정리하면,

$$\frac{\partial y_k}{\partial \hat{y}_i} = \begin{cases} y_k(1-y_k) & \text{if } k=i \\ -y_k y_i & \text{otherwise} \end{cases} = y_i(\delta_{ki} - y_k) \tag{5.7.12}$$

이다. 여기서,

$$\delta_{ki} = \begin{cases} 1 & \text{if } k=i \\ 0 & \text{otherwise} \end{cases} \tag{5.7.13}$$

는 Kronecker delta 이다.

참조: 교차 엔트로피 오차함수의 미분

SoftMax 함수를 지닌 출력층에서 CE 오차함수가 식 (5.7.4)로 주어진 경우, $\hat{y_k}$에 대한 미분을 계산하면

$$\frac{\partial E_{CE}}{\partial \hat{y_k}} = -\sum_j t_j \frac{\partial \log(y_j)}{\partial \hat{y_k}} = -\sum_j t_j \frac{\partial \log(y_j)}{\partial y_j} \frac{\partial y_j}{\partial \hat{y_k}} \tag{5.7.14}$$

이며, 미분에서 연쇄법칙(Chain Rule)을 이용하면 마지막 항을 얻게 된다. 이를 $j \neq k$일 때와 $j = k$ 때를 구분하여 정리하면

$$\begin{aligned} \frac{\partial E_{CE}}{\partial \hat{y_k}} &= -\sum_j \frac{t_j}{y_j} \frac{\partial y_j}{\partial \hat{y_k}} = -\frac{t_k}{y_k} y_k (1 - y_k) - \sum_{j \neq k} \frac{t_j}{y_j} (-y_j y_k) \\ &= \sum_j t_j y_k - t_k \end{aligned} \tag{5.7.15}$$

이다. $\sum_j t_j = 1$이므로,

$$-\frac{\partial E_{CE}}{\partial \hat{y_k}} = t_k - y_k \tag{5.7.16}$$

이 된다.

예제 5.7-1

그림 5.10과 같이 심층신경회로망이 주어졌다. $h_j^{(l)}\,(l=,1,2,,,,L)$과 아래층 사이의 연결 가중치를 $w_{ji}^{(l)}\,(l=,1,2,,,,L)$이라 하고 마지막층 출력노드 y_k와 아래층 $h_j^{(L)}$ 사이의 연결 가중치를 v_{kj} 라고 할 때, 정방향 전파의 계산 과정을 적어보아라. 또한, 출력노드의 오류신호가 $\delta_k = t_k - y_k$로 주어지면 역방향 전파에 의한 은닉노드의 오류신호를 적어보아라.

풀이

먼저 입력노드 x_i와 첫 은닉층 노드 $h_j^{(1)}$ 사이의 가중치는 $w_{ji}^{(1)}$이므로 첫 은닉층 노드에 입력되는 가중치 합은 $\hat{h}_j^{(1)} = \sum_i w_{ji}^{(1)} x_i$ 이 되고, 은닉노드의 활성화 함수는 식 (5.7.1)과 같이 ReLU 함수로 주어졌다고 하면, $h_j^{(1)} = f(\hat{h}_j^{(1)})$이 된다. $l = 2, 3, 4, \ldots, L$에 위치한 은닉노드들은 가중치 합이 $\hat{h}_j^{(l)} = \sum_i w_{ji}^{(l)} h_i^{(l-1)}$와 같이 계산된 후, ReLU 함수에 의해 $h_j^{(l)} = f(\hat{h}_j^{(l)})$ 로 주어진다. 마지막 층 출력노드는 가중치 합이 $\hat{y}_k = \sum_j v_{kj} h_j^{(L)}$ 로 계산되고 SoftMax 활성화 함수 식 (5.7.3)과 같이 $y_k = Softmax(\hat{y}_k)$로 주어진다.

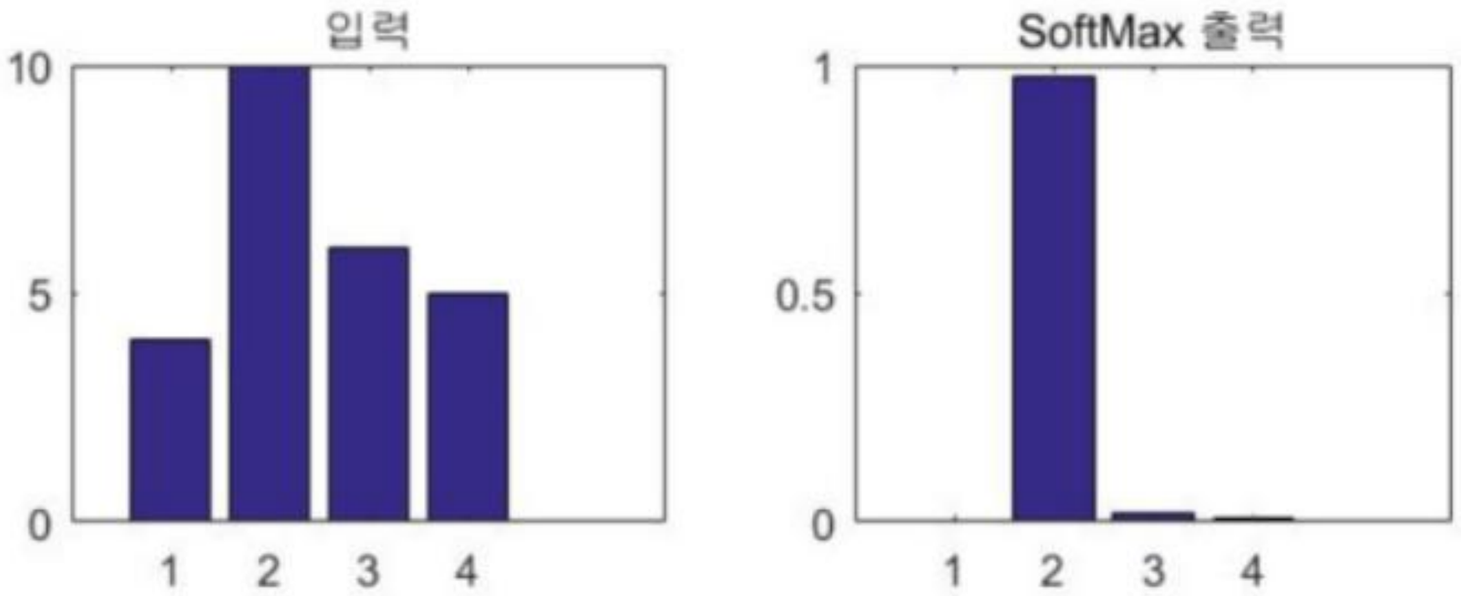
역전파 계산 과정은 출력노드의 오류신호가 $\delta_k = t_k - y_k$ 로 주어졌으므로 L 번째 은닉층의 오류신호는 $\delta_j^{(L)} = f'(\hat{h}_j^{(L)}) \sum_k v_{kj} \delta_k = f'(\hat{h}_j^{(L)}) \sum_k v_{kj} (t_k - y_k)$이 된다. $l = 1, 3, 4, \ldots, L-1$ 층 은닉노드들의 오류신호는 윗 층의 오류신호로부터 $\delta_j^{(l)} = f'(\hat{h}_j^{(l)}) \sum_k w_{kj}^{(l+1)} \delta_k^{(l+1)}$와 같이 계산된다.

예제 5.7-2

SoftMax 함수는 벡터 요소들 중에서 큰 값은 더 크게, 작은 값은 더 작게 상대적으로 조정하며 그 값들의 합은 1이 되게 한다. SoftMax 함수에 4차원 벡터 [4 10 6 5]가 입력되었을 때, SoftMax 함수를 통과한 후의 4차원 벡터를 구하여 보아라.

풀이

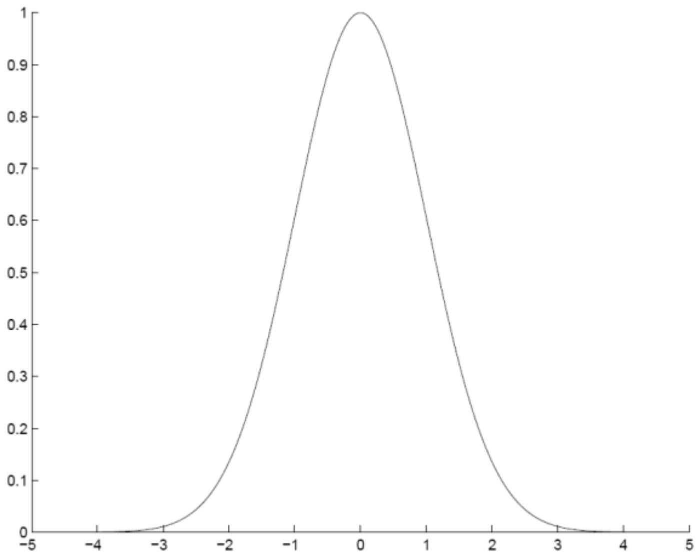
먼저 $\widehat{y_1}=4$, $\widehat{y_2}=10$, $\widehat{y_3}=6$, $\widehat{y_4}=5$로 두고서 식 (5.7.3)에 따라 y_k, $(k=1,2,3,4)$를 계산하면 $y_1=0.0024$, $y_2=0.9732$, $y_3=0.0178$, $y_4=0.0066$이 되며, $y_1+y_2+y_3+y_4=1$이다. SoftMax에 입력되는 값들과 SoftMax 출력 값을 그림으로 비교하여 보면 아래와 같다. 이 그림에서 SoftMax 함수에 의해 요소들 간의 차이가 확연하게 변한 것을 볼 수 있다.



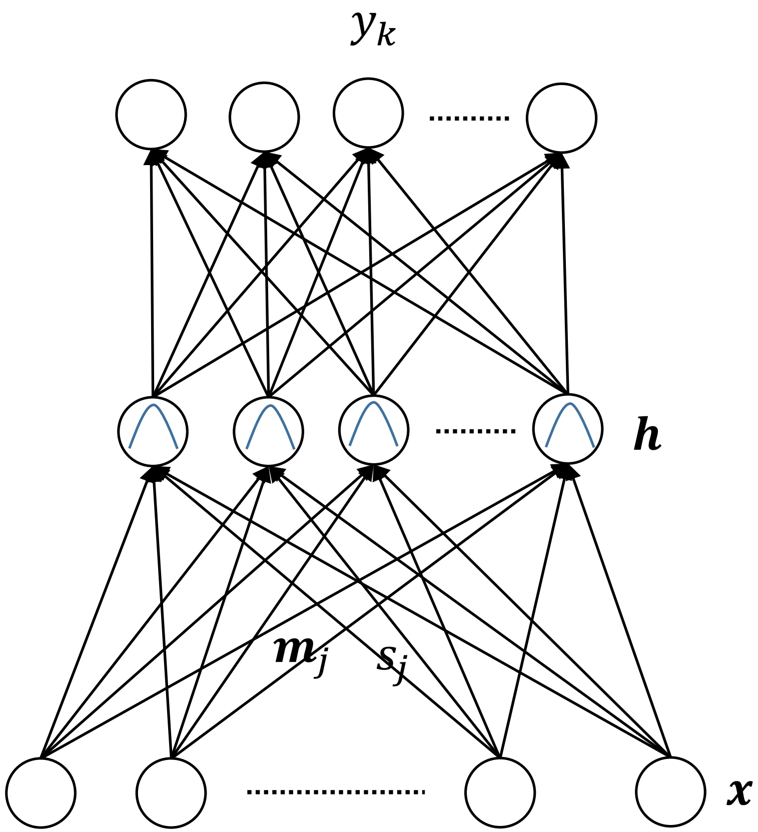
5.8. Radial Basis Function Networks

- Locally-tuned units:

$$h_j = \exp\left[-\frac{\|\boldsymbol{x} - \boldsymbol{m}_j\|^2}{2s_j^2}\right] \qquad (5.8.1)$$



$$y_k = \sum_j w_{kj} h_j \qquad (5.8.2)$$



Local vs. Distributed Representation

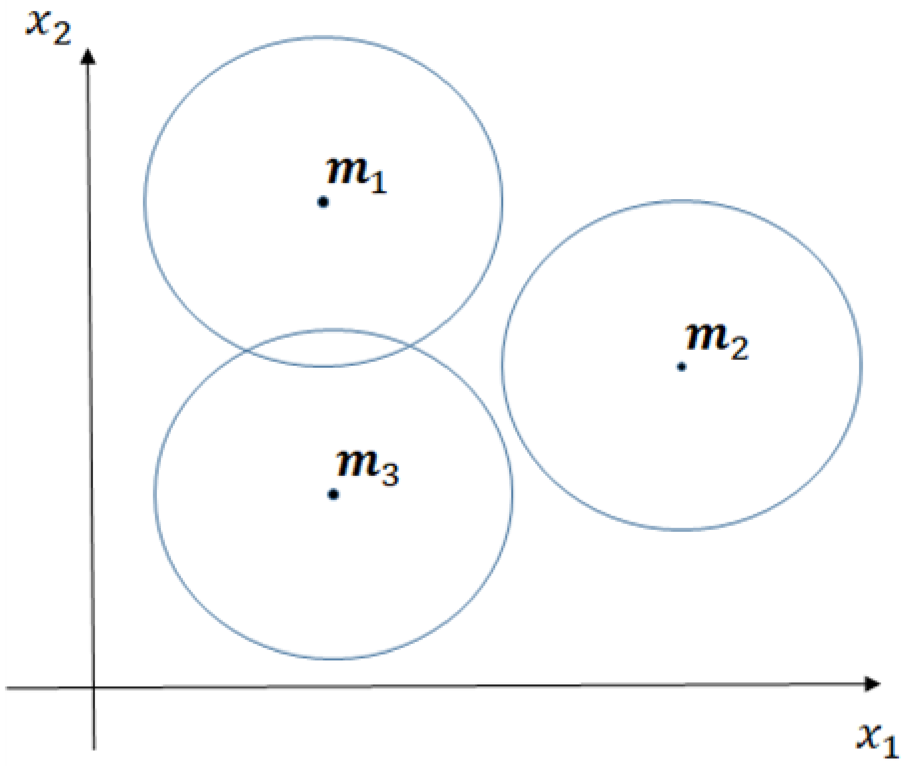


그림 5.12. RBFN의 국소 표현

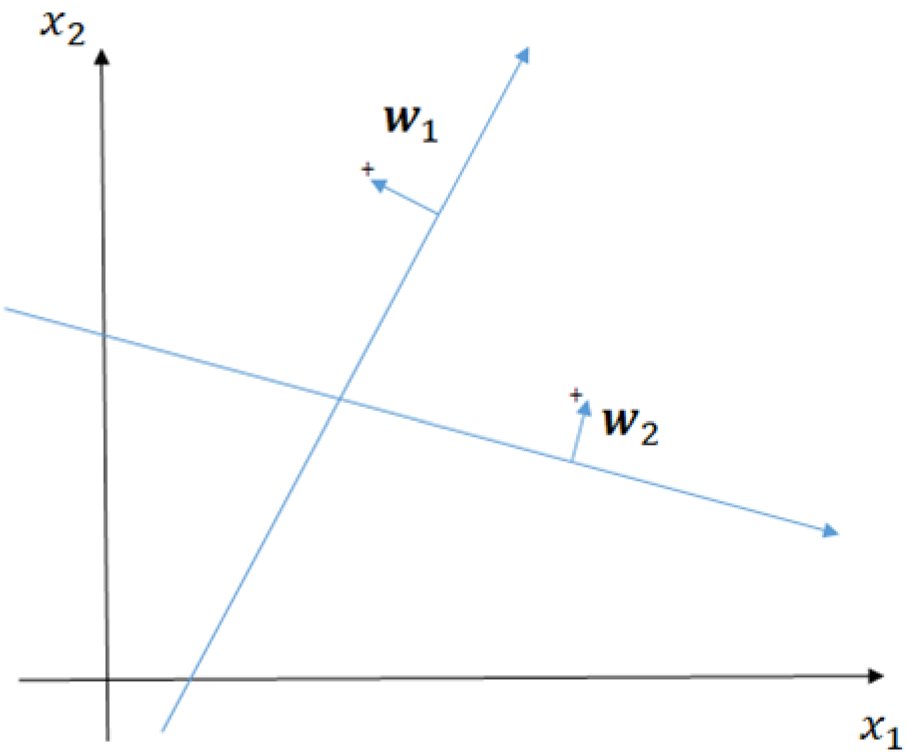


그림 5.13. 다층퍼셉트론의 분포된 표현

Training RBF Network

- Hybrid learning
    - First layer centers and spreads (Unsupervised k-means)
    - Second layer weights (Supervised gradient descent)
- Fully supervised
    - Similar to backpropagation in MLP, gradient descent for all parameters

RBF Network: Fully Supervised Method

- Similar to backpropagation in MLP

$$E_{MSE} = \frac{1}{2}\sum_{k=1}^{M}(t_k - y_k)^2 \qquad (5.8.3)$$

$$\frac{\partial E_{MSE}}{\partial w_{kj}} = -(t_k - y_k)h_j \qquad (5.8.4)$$

$$\Delta w_{kj} = -\eta\frac{\partial E_{MSE}}{\partial w_{kj}} = \eta(t_k - y_k)h_j \qquad (5.8.5)$$

$$\begin{aligned} -\frac{\partial E_{MSE}}{\partial m_{jh}} &= -\frac{1}{2}\sum_k \frac{\partial(t_k - y_k)^2}{\partial m_{jh}} \\ &= \sum_k -\frac{1}{2}\frac{\partial(t_k - y_k)^2}{\partial y_k}\frac{\partial y_k}{\partial h_j}\frac{\partial h_j}{\partial m_{jh}} \\ &= \sum_k (t_k - y_k)w_{kj}\frac{\partial h_j}{\partial m_{jh}} \end{aligned} \qquad (5.8.6)$$

$$\frac{\partial h_j}{\partial m_{jh}} = \frac{\partial \exp\left(-\frac{\sum_i (x_i - m_{ji})^2}{2s_j^2}\right)}{\partial m_{jh}} = h_j\frac{x_h - m_{jh}}{s_j^2} \qquad (5.8.7)$$

$$\Delta m_{jh} = \eta h_j\frac{x_h - m_{jh}}{s_j^2}\sum_k w_{kj}(t_k - y_k) \qquad (5.8.8)$$

RBF Network: Fully Supervised Method

- Similar to backpropagation in MLP

$$-\frac{\partial E_{MSE}}{\partial s_j} = -\frac{1}{2}\sum_k \frac{\partial (t_k - y_k)^2}{\partial s_j} \tag{5.8.9}$$
$$= \sum_k -\frac{1}{2}\frac{\partial (t_k - y_k)^2}{\partial y_k}\frac{\partial y_k}{\partial h_j}\frac{\partial h_j}{\partial s_j}$$
$$= \sum_k (t_k - y_k) w_{kj} \frac{\partial h_j}{\partial s_j}$$

$$\frac{\partial h_j}{\partial s_j} = \frac{\partial \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{m}_j\|^2}{2s_j^2}\right)}{\partial s_j} = h_j \frac{\partial\left(-\frac{\|\boldsymbol{x} - \boldsymbol{m}_j\|^2}{2s_j^2}\right)}{\partial s_j} = h_j \frac{\|\boldsymbol{x} - \boldsymbol{m}_j\|^2}{s_j^3} \tag{5.8.10}$$

$$\Delta s_j = \eta h_j \frac{\|\boldsymbol{x} - \boldsymbol{m}_j\|^2}{s_j^3} \sum_k (t_k - y_k) w_{kj} \tag{5.8.11}$$