BSC LAB
Business & Service Computing

PNU Industrial Data Science

Neural Network

인공신경망

Prof. Hyerim Bae

번호판 인식

인공지능, 사람보다 정확하게 자동차 번호판 읽어냈다

고성능 사진 해석기능 제공, 범죄 예방, 주차 관리 등 스마트 치안 및 생활 안전에 도움

책 읽어주는 조명


전자신문

책 읽어주는 조명 '램프' 일주일만에 완판…네이버 AI 대중화 선봉에

발행일 : 2020.10.27 15:18

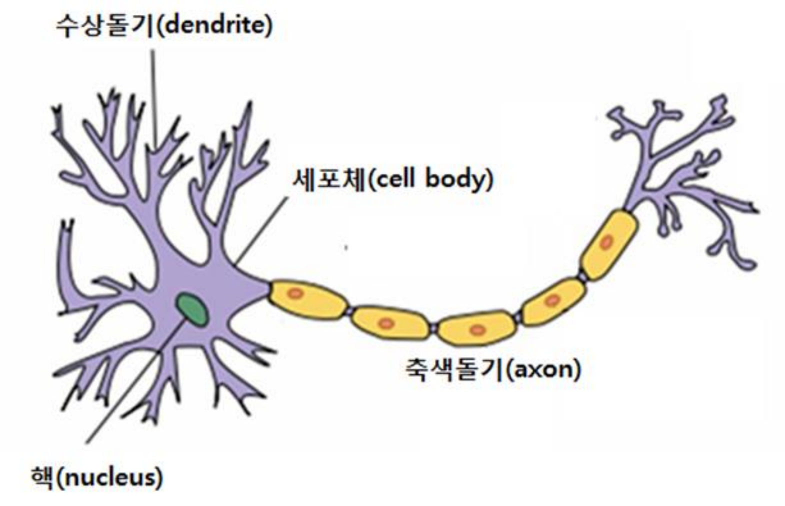
URL

<클로바 램프, 사진=네이버>

Contents

산업데이터과학은 산업현장에서 수집된 데이터를 분석하는데 필요한 기초 소양을 강의합니다.

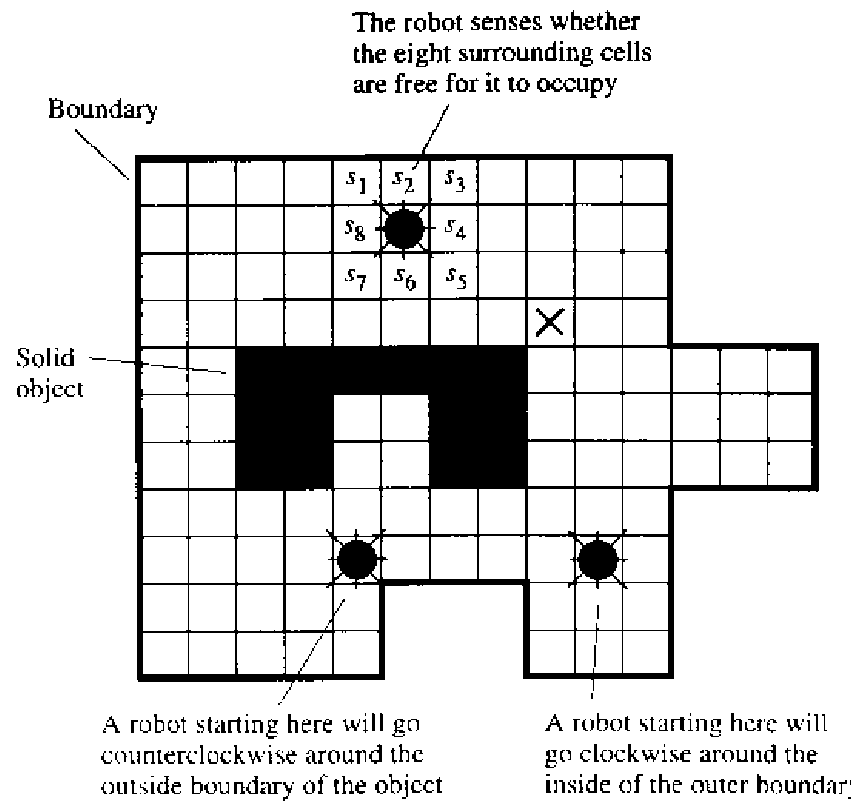
Stimulus-Response Agents



Perception and action

- S-R agents
  - Machines that have no internal state
  - Simply react to immediate stimuli

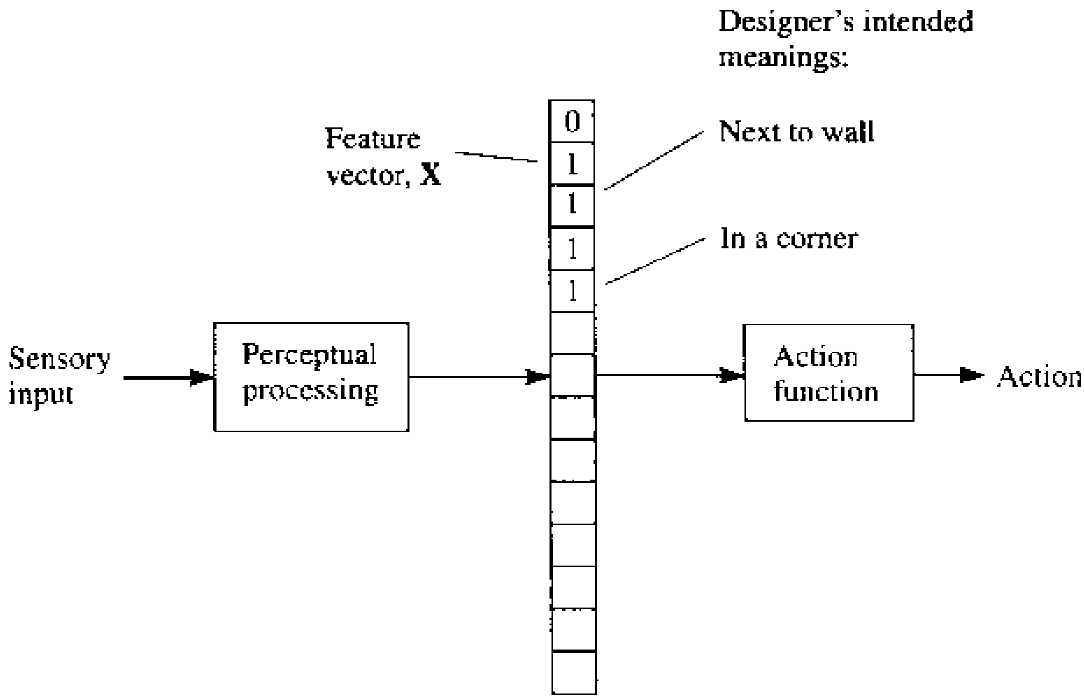
- 2-dimensional grid-space world
  - Enclosed by boundaries
  - Unmovable objects
  - Has no “tight spaces”
  - Action
    - Go to a cell adjacent to a boundary or object
    - follow that boundary along its perimeter



- Sensory inputs: $s_1, \ldots s_8$
  - Have value 0 whenever the corresponding cell can be occupied by the root;
  - Otherwise, have value 1
- Robot movements
  - *north* moves the robot one cell up in the cellular grid
  - *east* moves the robot one cell to the right
  - *south* moves the robot one cell down
  - *west* moves the robot one cell to the left
- Designer's job
  - Specify a function of the sensory input
  - Selects actions appropriate for the task.

- Division of processes
  - Perception processing and action computation

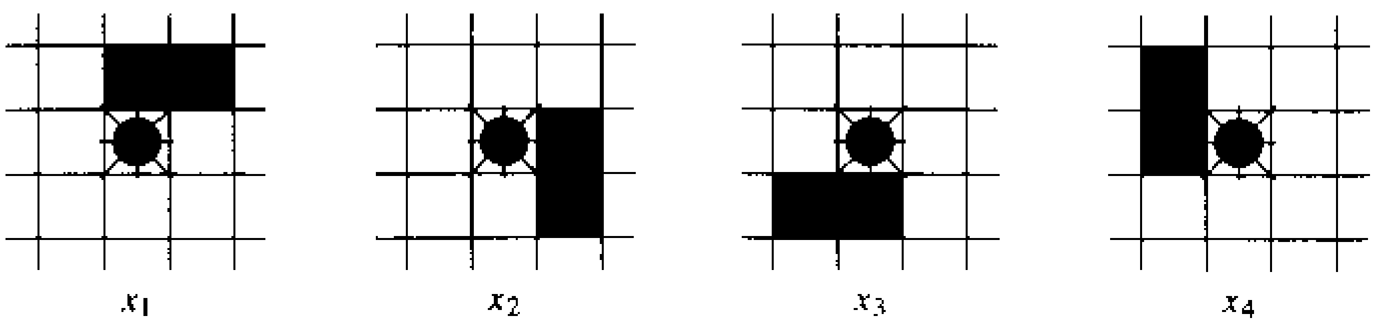
- Perceptual processing
  - produces feature vector **X**
    - *numeric features*: real number
    - *categorical features:* categories
  - **Action computation**
    - Selects an action based on feature vector



- The split between perception and action is arbitrary
  - The split is made in such a way that the same features would be used repeatedly in a variety of tasks to be performed
- The computation of features from sensory signals can be regarded as often used library routines
  - needed by many different action functions
- The next problems
  - (1) converting raw sensory data into a feature vector
  - (2) specifying an action function

Perception

- The sensory input consists of the eight values of $(s_1, \ldots, s_8)$
  - There are 2^8=256 combinations of the values
- For the robot task, there are four binary-valued features of the sensory values that are useful for computing an appropriate action $(x_1, \ldots x_4)$
  - For example, $x_1 = 1$ if and only if s_2=1 or s_3=1
- Perceptual processing might occasionally give erroneous, ambiguous, or incomplete information about the robot's environment
  - Such errors might evoke inappropriate actions
- For robots with more complex sensors and tasks, designing appropriate perceptual processing can be challenging



Action

- Specifying a function that selects the appropriate boundary-following action

  If x_1=1 and x_2=0, move **east**

  If x_2=1 and x_3=0, move **south**

  If x_3=1 and x_4=0, move **west**

  If x_4=1 and x_1=0, move **north**

  - None of the features has value 1, the robot can move in *any* direction until it encounters a boundary
  - The robot take the actions happen to be Boolean combination of the features
    - The features themselves are also Boolean combination of the sensory inputs

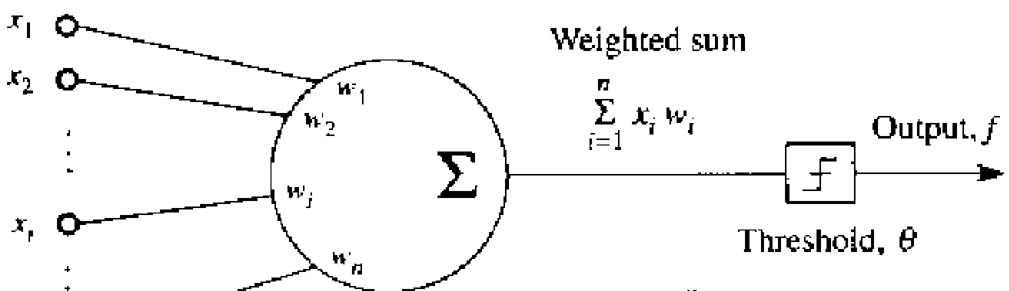
Representing and implementing action function: production system

- Production system comprises an ordered list of rules called *production* rules or *productions*
  - $c_i \rightarrow a_i$, where c_i is the condition part and a_i is the action part
  - Production system consists of a list of such rules
  - Condition part
    - Can be any binary-valued function of the features
    - Often a monomial
  - Action part
    - Primitive action, a call to another productive system, or a set of actions to be executed simultaneously
- Production system representation for the boundary following routine
  - An example of a durative systems-system that never ends

$$
\begin{aligned}
x_4\overline{x_1} &\rightarrow north \\
x_3\overline{x_4} &\rightarrow west \\
x_2\overline{x_3} &\rightarrow south \\
x_1\overline{x_2} &\rightarrow east \\
1 &\rightarrow north
\end{aligned}
$$

Network

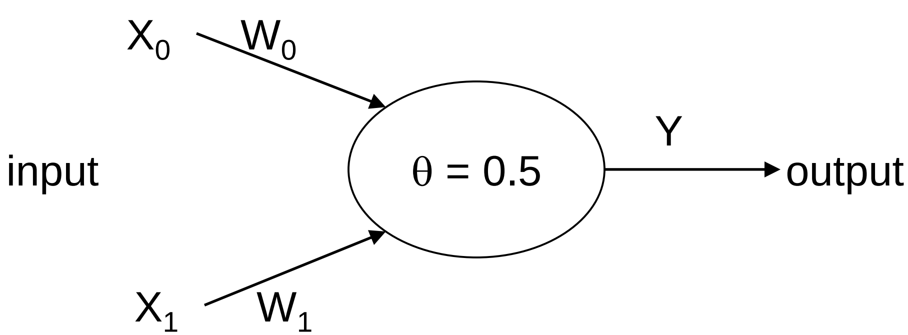
- *Threshold logic unit (TLU)*
  - Circuit consists of networks of threshold elements or other elements that compute a nonlinear function of a weighted sum of their inputs
  - **TLU separates the space of input vectors yielding an above-threshold response from those yielding a below-threshold response by a linear space-called a *hyperplane***



$$f = 1 \text{ if } \sum_{i=1}^{n} x_i w_i \geq \theta$$
$$= 0 \ \textit{otherwise}$$

- *Linearly separable functions*
  - The boolean functions implementable by a TLU
  - Many boolean functions are linearly separable
  - Exclusive-or function of two variables is an example of not linearly separable

– AND, XOR Example



input		output		
X_0	X_1	AND	f	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	0	1
1	1	1	0	0

- AND

$$
\begin{aligned}
0 \times W_0 + 0 \times W_1 &= 0 && < 0.5 \\
0 \times W_0 + 1 \times W_1 &= W_1 && < 0.5 \\
1 \times W_0 + 0 \times W_1 &= W_0 && < 0.5 \\
1 \times W_0 + 1 \times W_1 &= W_0 + W_1 && > 0.5
\end{aligned}
$$

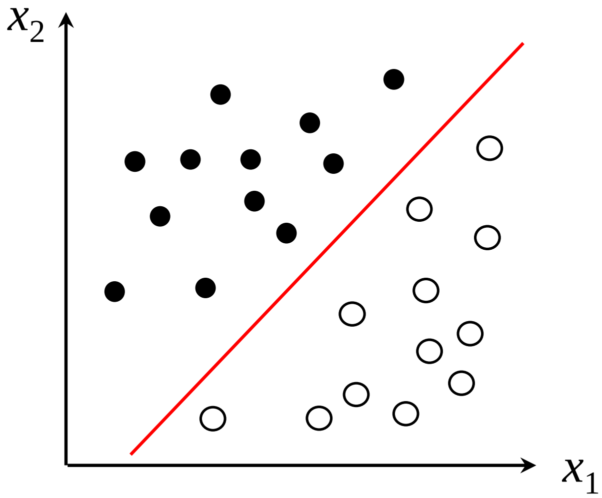
→ W_0, W_1 : 0.3 or 0.4

- XOR

$$
\begin{aligned}
0 \times W_0 + 0 \times W_1 &= 0 & < 0.5 \\
0 \times W_0 + 1 \times W_1 &= W_1 & > 0.5 \\
1 \times W_0 + 0 \times W_1 &= W_0 & > 0.5 \\
1 \times W_0 + 1 \times W_1 &= W_0 + W_1 & < 0.5
\end{aligned}
$$

→ W_0, W_1 do not exist that satisfy above

→ cannot solve XOR

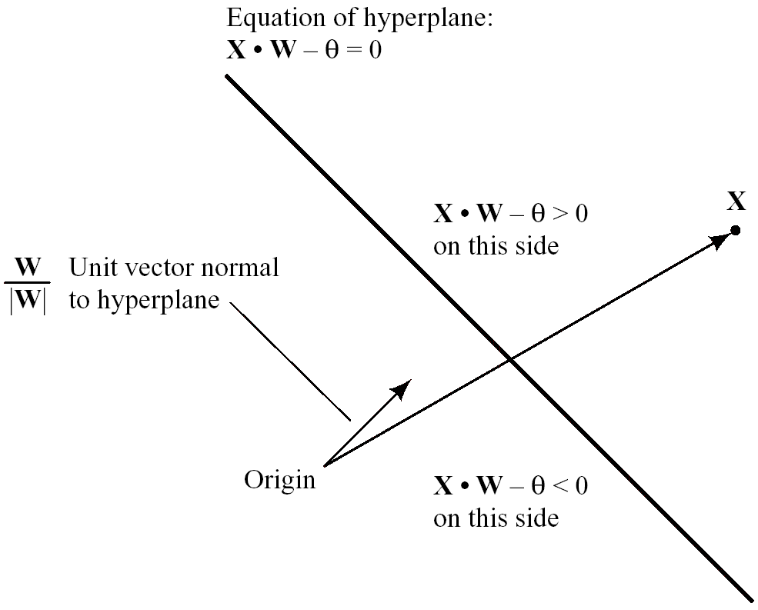


Neural Network

- TLU (threshold logic unit): Basic units for neural networks
  - Based on some properties of biological neurons
- Training set
  - Input: real value, boolean value, …

    N-dimensional vector: $X=(x_1, x_2, \dots x_n)$
  - Output:
    - d_i: associated actions (Label, Class …)
- Target of training
  - Finding $f(\mathbf{X})$ corresponds "acceptably" to the members of the training set.
  - Supervised learning: Labels are given along with the input vectors.

- Training TLU: Adjusting variable weights
- A single TLU: Perceptron, Adaline (*ada*ptive *lin*ear *e*lement) [Rosenblatt 1962, Widrow 1962]
- Elements of TLU
  - Weight: $\mathbf{W} = (w_1, \ldots, w_n)$
  - Threshold: θ
- Output of TLU: Using weighted sum $s = \mathrm{W} \cdot \mathrm{X}$
  - 1 if $\mathbf{s} - \theta > 0$
  - 0 if $\mathbf{s} - \theta < 0$
- Hyperplane
  - $\mathbf{W} \cdot \mathbf{X} - \theta = 0$



Augmented vectors

- Adopting the convention that threshold is fixed to 0.
- Arbitrary thresholds: $(n + 1)$-dimensional vector
  - $X = (x_1, \ldots, x_n, 1)$
  - $\mathbf{W} = (w_1, \ldots, w_n, -\theta)$
- Output of TLU
  - 1 if $\mathbf{W}\cdot\mathbf{X} \geq 0$
  - 0 if $\mathbf{W}\cdot\mathbf{X} < 0$

Gradient Descent Methods

- Training TLU: minimizing the *error function* by adjusting weight values.
- Batch learning v.s. incremental learning
- Commonly used error function: squared error $\varepsilon = (d - f)^2$
  - Gradient: $$\frac{\partial \varepsilon}{\partial \mathbf{W}} \stackrel{def}{=} \left[\frac{\partial \varepsilon}{\partial w_1}, \ldots, \frac{\partial \varepsilon}{\partial w_i}, \ldots, \frac{\partial \varepsilon}{\partial w_{n+1}} \right]$$
  - Chain rule: $$\frac{\partial \varepsilon}{\partial \mathbf{W}} = \frac{\partial \varepsilon}{\partial s} \frac{\partial s}{\partial \mathbf{W}} = \frac{\partial \varepsilon}{\partial s} \mathbf{X} = -2(d - f) \frac{\partial f}{\partial s} \mathbf{X}$$
- *f* is not continuously differentiable with respect to s ($\partial f / \partial s$) :
  - Ignoring threshold function: $f = s$
  - Replacing threshold function with differentiable nonlinear function

The Widrow-Hoff Procedure

- Weight update procedure:
  - Using $f = s = \mathbf{W} \cdot \mathbf{X}$
  - Data labeled $1 \rightarrow 1$, Data labeled $0 \rightarrow -1$
- Gradient: if $\mathrm{f} = \mathrm{s}$,

$$\frac{\partial \varepsilon}{\partial \mathbf{W}} = -2(d - f)\frac{\partial f}{\partial s}\mathbf{X} = -2(d - f)\mathbf{X}$$

- New weight vector

$$\mathbf{W} \leftarrow \mathbf{W} + c(d - f)\mathbf{X}$$

- Widrow-Hoff (delta) rule
  - $(d - f) > 0 \rightarrow$ increasing $s \rightarrow$ decreasing $(d - f)$
  - $(d - f) < 0 \rightarrow$ decreasing $s \rightarrow$ increasing $(d - f)$

The generalized delta procedure

- Sigmoid function (differentiable): [Rumelhart, et al. 1986]

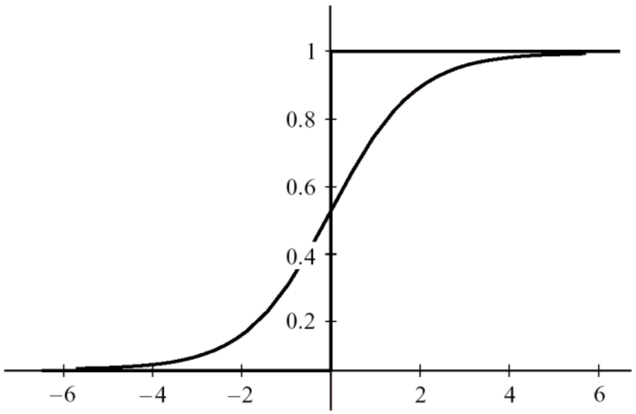
$$f(s) = 1/(1+e^{-s})$$

- Gradient:

$$\frac{\partial \varepsilon}{\partial \mathbf{W}} = -2(d-f)\frac{\partial f}{\partial s}\mathbf{X} = -2(d-f)f(1-f)\mathbf{X}$$

- Generalized delta procedure:

$$\mathbf{W} \leftarrow \mathbf{W} + c(d-f)f(1-f)\mathbf{X}$$



  - Target output: 1, 0
  - Output f = output of sigmoid function
  - $f(1-f) = 0$, where $f = 0$ or 1
  - Weight change can occur only within 'fuzzy' region surrounding the hyperplane (near the point $f(s) = ½$).

The error-correction procedure

- Using threshold unit: $(d-f)$ can be either 1 or -1.

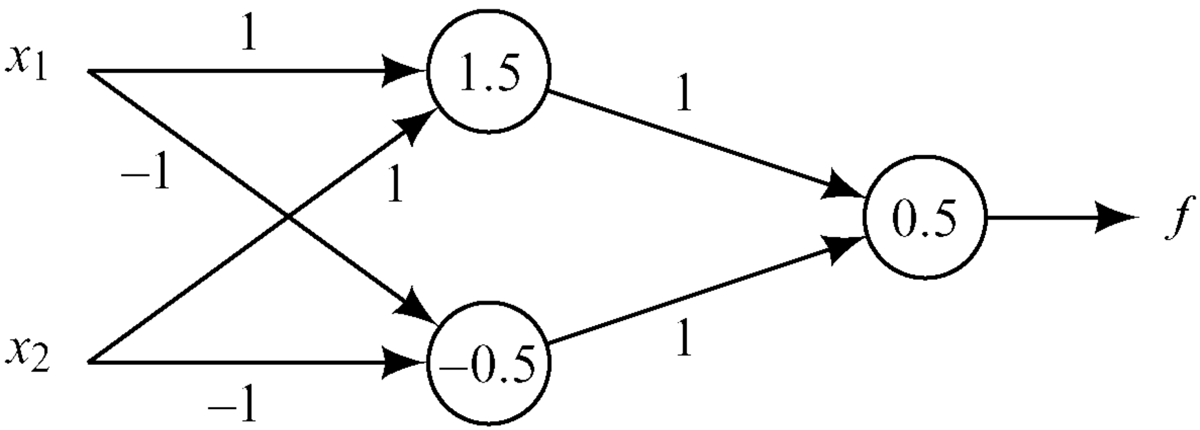
$$\mathbf{W} \leftarrow \mathbf{W} + c(d-f)\mathbf{X}$$

  - In the linearly separable case, after finite iteration, **W** will be converged to the solution.
    - If there is some weight vector, that produces a correct output for all of the input vectors, then after a finite number of input vector presentation, the procedure will find such a weight vector and thus make no more weight changes
  - In the nonlinearly separable case, **W** will never be converged.
  - The Widrow-Hoff and generalized delta procedures will find minimum squared error solutions even when the minimum error is not zero.

Neural network: motivation

- Need for use of multiple TLUs
  - Feedforward network: no cycle
  - Recurrent network: cycle (treated in a later chapter)
  - Layered feedforward network
    - j_{th} layer can receive input only from $j-1_{\text{th}}$ layer.
- Example :

$$f = x_1 x_2 + \bar{x}_1 \bar{x}_2$$



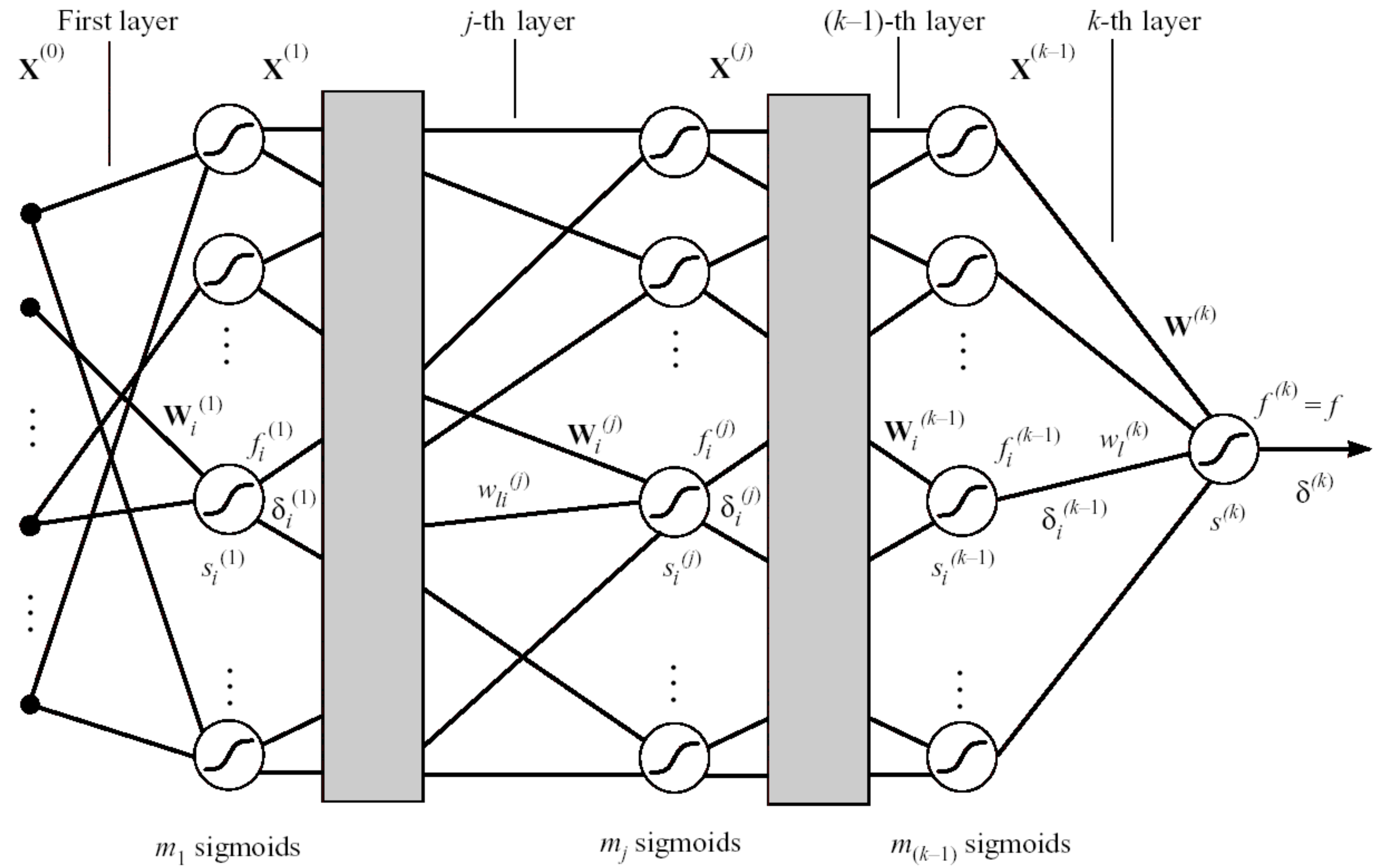
Notation

- Hidden unit: neurons in all but the last layer
- Output of j-th layer: $X^{(j)} \rightarrow$ input of $(j+1)$-th layer
- Input vector: $X^{(0)}$
- Final output: f
- The weight of i-th sigmoid unit in the j-th layer: $W_i^{(j)}$
- Weighted sum of i-th sigmoid unit in the j-th layer: $s_i^{(j)}$

$$s_i^{(j)} = \mathbf{X}^{(j-1)} \cdot \mathbf{W}_i^{(j)}$$

- Number of sigmoid units in j-th layer: m_j

$$W_l^{(j)} = (w_{1,i}^{(j)}, \dots, w_{l,i}^{(j)}, \dots, w_{m_{(j-1)}+1,i}^{(j)})$$



First layer
j-th layer
(k–1)-th layer
k-th layer
$\mathbf{X}^{(0)}$
$\mathbf{X}^{(1)}$
$\mathbf{X}^{(j)}$
$\mathbf{X}^{(k-1)}$
$\mathbf{W}_i^{(1)}$
$f_i^{(1)}$
$\delta_i^{(1)}$
$s_i^{(1)}$
$w_{li}^{(j)}$
$\mathbf{W}_i^{(j)}$
$f_i^{(j)}$
$\delta_i^{(j)}$
$s_i^{(j)}$
$\mathbf{W}_i^{(k-1)}$
$f_i^{(k-1)}$
$\delta_i^{(k-1)}$
$s_i^{(k-1)}$
$\mathbf{W}^{(k)}$
$w_l^{(k)}$
$f^{(k)} = f$
$s^{(k)}$
$\delta^{(k)}$
m_1 sigmoids
m_j sigmoids
$m_{(k-1)}$ sigmoids

The backpropagation method

- Gradient of $W_i^{(j)}$:

$$s_i^{(j)} = \mathbf{X}^{(j-1)} \cdot \mathbf{W}_i^{(j)}$$

$$\frac{\partial \varepsilon}{\partial \mathbf{W}_i^{(j)}} = \frac{\partial \varepsilon}{\partial s_i^{(j)}} \frac{\partial s_i^{(j)}}{\partial \mathbf{W}_i^{(j)}} = \frac{\partial \varepsilon}{\partial s_i^{(j)}} \mathbf{X}^{(j-1)}$$

$$= -2\boxed{(d - f) \frac{\partial f}{\partial s_i^{(j)}}} \mathbf{X}^{(j-1)} = -2\delta_i^{(j)} \mathbf{X}^{(j-1)}$$

Local gradient

$$\frac{\partial \varepsilon}{\partial s_i^{(j)}} = \frac{\partial (d - f)^2}{\partial s_i^{(j)}} = -2(d - f) \frac{\partial f}{\partial s_i^{(j)}}$$

- Weight update:

$$\mathbf{W}_i^{(j)} \leftarrow \mathbf{W}_i^{(j)} + c_i^{(j)} \delta_i^{(j)} \mathbf{X}^{(j-1)}$$

Computing weight changes in the final layer

- Local gradient:

$$\delta^{(k)} = (d - f)\frac{\partial f}{\partial s_i^{(k)}}$$

$$= (d - f) f (1 - f)$$

- Weight update:

$$\mathbf{W}^{(k)} \leftarrow \mathbf{W}^{(k)} + c^{(k)}(d - f) f (1 - f)\mathbf{X}^{(k-1)}$$

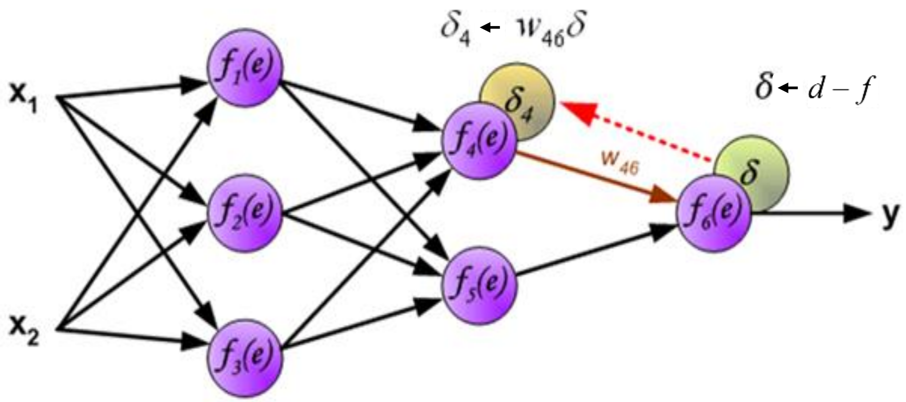
- Attention to recursive equation of local gradient!

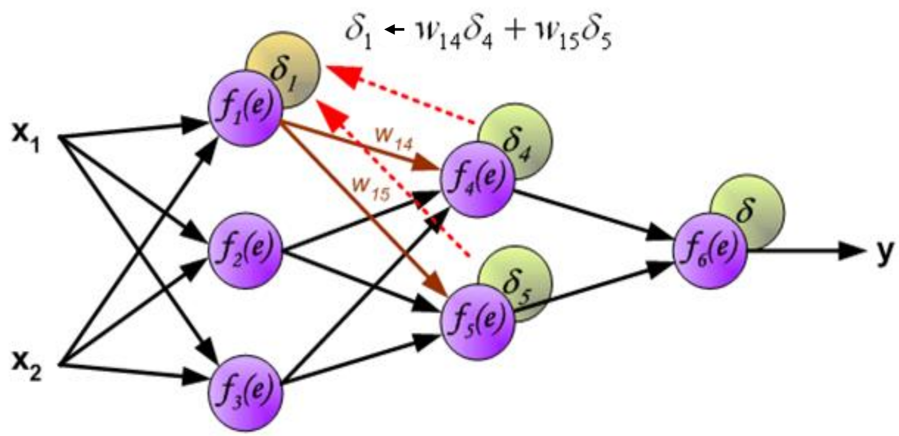
$$\delta^{(k)} = f(1-f)(d-f)$$

$$\delta_i^{(j)} = f_i^{(j)}(1-f_i^{(j)})\sum_{l=1}^{m_{j+1}}\delta_i^{(j+1)}w_{il}^{(j+1)}$$

- Backpropagation:
  - Error is back-propagated from the output layer to the input layer
  - Local gradient of the latter layer is used in calculating local gradient of the former layer.

$$\delta_i^{(j)} = f_i^{(j)}(1 - f_i^{(j)})\sum_{l=1}^{m_{j+1}} \delta_l^{(j+1)} w_{il}^{(j+1)}$$





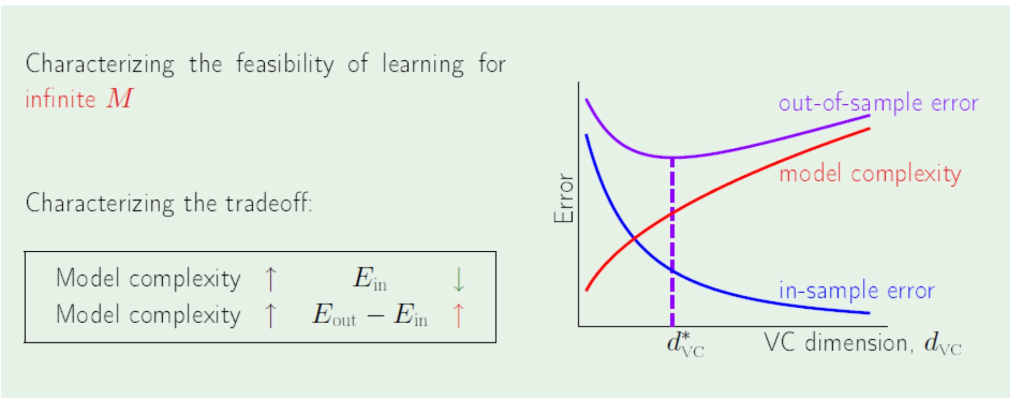
Generalization, Accuracy, and overfitting

- Generalization ability:
  - NN appropriately classifies vectors not in the training set.
  - Measurement = accuracy
- Curve fitting
  - Number of training input vectors ≥ number of degrees of freedom of the network.
  - In the case of m data points, is $(m\text{-}1)$-degree polynomial best model? **No, it can not capture any special information.**
- Overfitting
  - Extra degrees of freedom are essentially just fitting the noise.
  - Given sufficient data, the *Occam's Razor* principle dictates to choose the lowest-degree polynomial that adequately fits the data.

Validation of NN

- Out-of-sample-set error rate
  - Error rate on data drawn from the same underlying distribution of training set.
- Dividing available data into a t*raining* set and a *validation* set
  - Usually use 2/3 for training and 1/3 for validation
- k-fold cross validation
  - k disjoint subsets (called folds).
  - Repeat training k times with the configuration: one validation set, k-1 (combined) training sets.
  - Take average of the error rate of each validation as the out-of-sample error.
  - Empirically 10-fold is preferred.

Error in NN



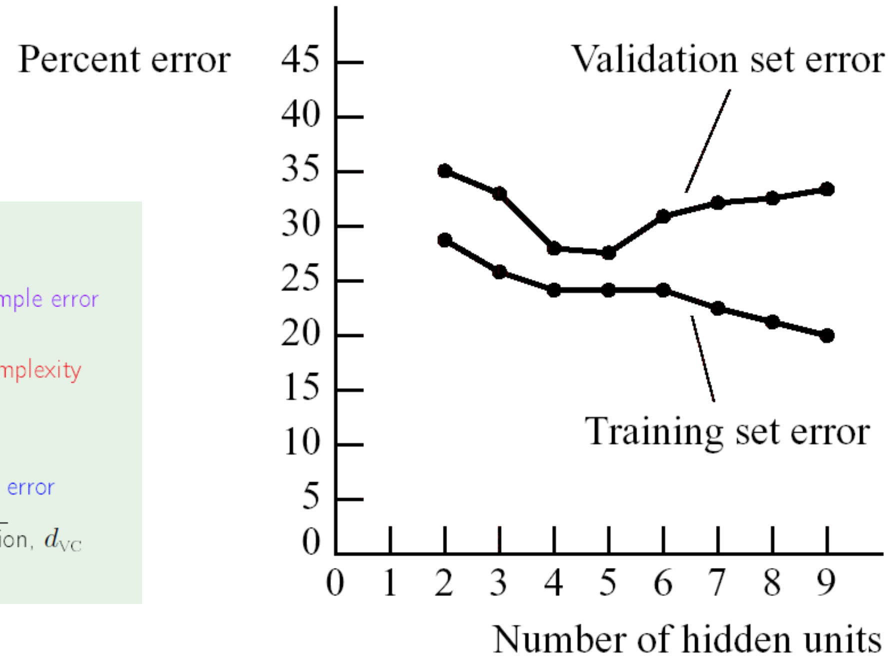


Fig 3.9 Estimate of Generalization Error Versus Number of Hidden Units

Appendix: Computing changes to the weights in intermediate layers

- Local gradient:

$$
\begin{aligned}
\delta_i^{(j)} &= (d-f)\frac{\partial f}{\partial s_i^{(j)}} \\
&= (d-f)\left[\frac{\partial f}{\partial s_1^{(j+1)}}\frac{\partial s_1^{(j+1)}}{\partial s_i^{(j)}} + \ldots \frac{\partial f}{\partial s_l^{(j+1)}}\frac{\partial s_l^{(j+1)}}{\partial s_i^{(j)}} + \ldots + \frac{\partial f}{\partial s_{m_{j+1}}^{(j+1)}}\frac{\partial s_{m_{j+1}}^{(j+1)}}{\partial s_i^{(j)}}\right] \\
&= \sum_{l=1}^{m_{j+1}} (d-f)\frac{\partial f}{\partial s_l^{(j+1)}}\frac{\partial s_l^{(j+1)}}{\partial s_i^{(j)}} = \sum_{l=1}^{m_{j+1}} \delta_l^{(j+1)}\frac{\partial s_l^{(j+1)}}{\partial s_i^{(j)}}
\end{aligned}
$$

  - The final ouput f, depends on $s_i^{(j)}$ through of the summed inputs to the sigmoids in the (j+1)-th layer.

- Need for computation of $\dfrac{\partial s_l^{(j+1)}}{\partial s_i^{(j)}}$

$$s_l^{(j+1)} = \mathbf{X}^{(j)} \cdot \mathbf{W}_l^{(j+1)} = \sum_{v=1}^{m_j+1} f_v^{(j)} w_{vl}^{(j+1)}$$

$$\frac{\partial s_l^{(j+1)}}{\partial s_i^{(j)}} = \frac{\partial \left[\sum_{v=1}^{m_j+1} f_v^{(j)} w_{vl}^{(j+1)} \right]}{\partial s_i^{(j)}} = \sum_{v=1}^{m_j+1} w_{vl}^{(j+1)} \frac{\partial f_v^{(j)}}{\partial s_i^{(j)}}$$

$$= w_{il}^{(j+1)} f_i^{(j)} (1 - f_i^{(j)})$$

- $v \neq i$: $$\frac{\partial f_v^{(j)}}{\partial s_i^{(j)}} = 0$$ $v = i$: $$\frac{\partial f_v^{(j)}}{\partial s_i^{(j)}} = f_v^{(j)} (1 - f_v^{(j)})$$
- Consequently,

$$\delta_i^{(j)} = f_i^{(j)} (1 - f_i^{(j)}) \sum_{l=1}^{m_{j+1}} \delta_l^{(j+1)} w_{il}^{(j+1)}$$