Artificial Intelligence
Chapter 3
Neural Networks

Biointelligence Lab
School of Computer Sci. & Eng.
Seoul National University

Outline

3.1 Introduction

3.2 Training Single TLUs

- Gradient Descent
- Widrow-Hoff Rule
- Generalized Delta Procedure

3.3 Neural Networks

- The Backpropagation Method
- Derivation of the Backpropagation Learning Rule

3.4 Generalization, Accuracy, and Overfitting

3.5 Discussion

3.1 Introduction

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

$$\mathrm{X}: \text{ n-dim vector}, \mathrm{X} = (x_1, ..., x_n)$$

  - 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.

3.2 Training Single TLUs
3.2.1 TLU Geometry

- 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 $s = \mathbf{W} \cdot \mathbf{X}$
  - 1 if $\mathbf{s} - \theta > 0$
  - 0 if $\mathbf{s} - \theta < 0$
- Hyperplane
  - $\mathbf{W} \cdot \mathbf{X} - \theta = 0$

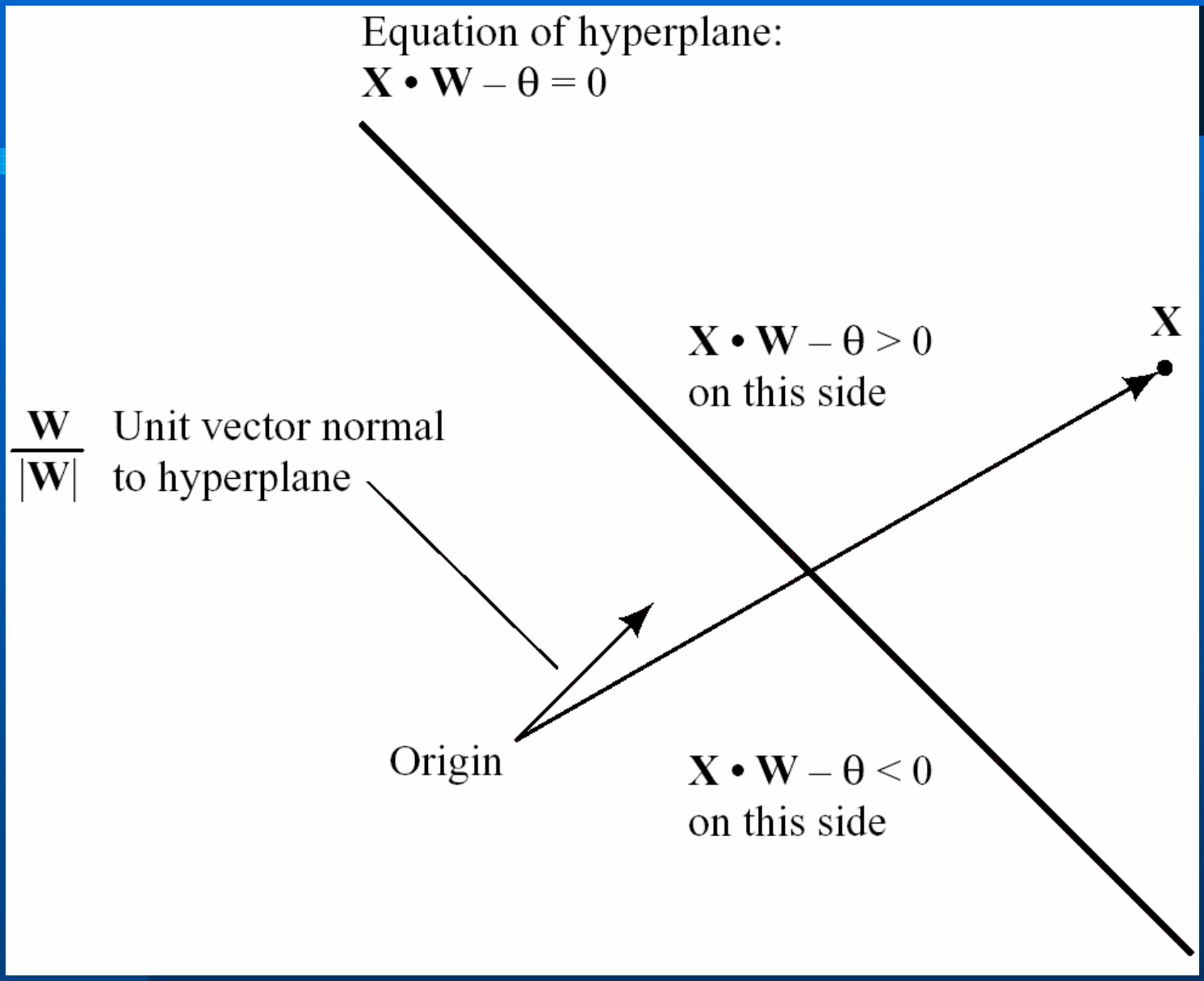


Figure 3.1 TLU Geometry

3.2.2 Augmented Vectors

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

3.2.3 Gradient Decent 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: $\frac{\partial \varepsilon}{\partial \mathbf{W}} \overset{def}{=} \left[\frac{\partial \varepsilon}{\partial w_1}, \ldots, \frac{\partial \varepsilon}{\partial w_i}, \ldots, \frac{\partial \varepsilon}{\partial w_{n+1}} \right]$
  - Chain rule: $\frac{\partial \varepsilon}{\partial \mathbf{W}} = \frac{\partial \varepsilon}{\partial s} \frac{\partial s}{\partial \mathbf{W}} = \frac{\partial \varepsilon}{\partial s} \mathbf{X} = -2(d - f) \frac{\partial f}{\partial s} \mathbf{X}$

- Solution of nonlinearity of $\partial f / \partial s$:
  - Ignoring threshold function: $f = s$
  - Replacing threshold function with differentiable nonlinear function

3.2.4 The Widrow-Hoff Procedure: First Solution

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

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

- 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 $(d - f)$

3.2.5 The Generalized Delta Procedure: Second Solution

- 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 $f(s) = ½$).

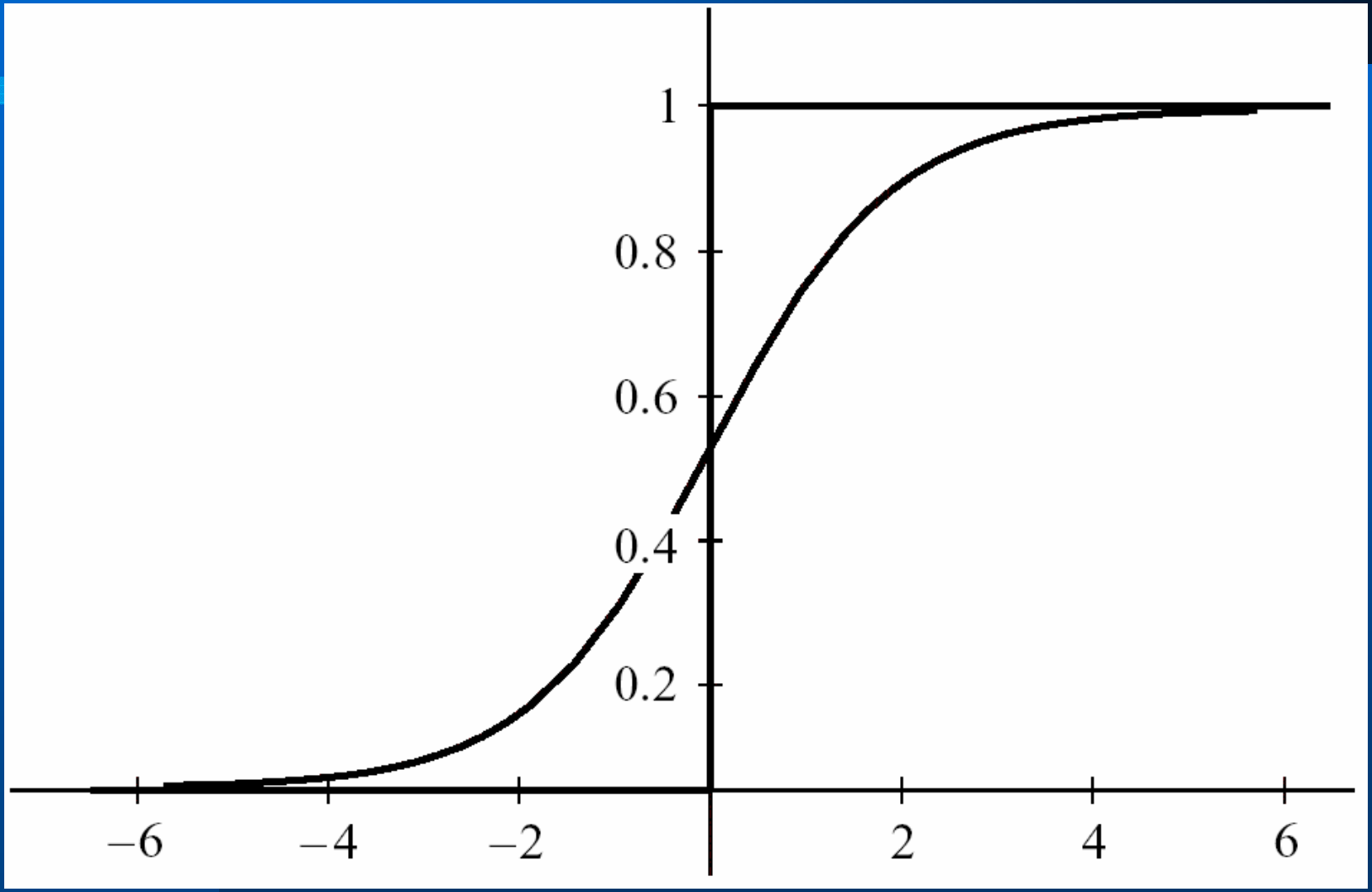


Figure 3.2 A Sigmoid Function

3.2.6 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.
- 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.

3.3 Neural Networks
3.3.1 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 $j - 1_{th}$ layer.
- Example : $f = x_1 x_2 + \bar{x}_1 \bar{x}_2$

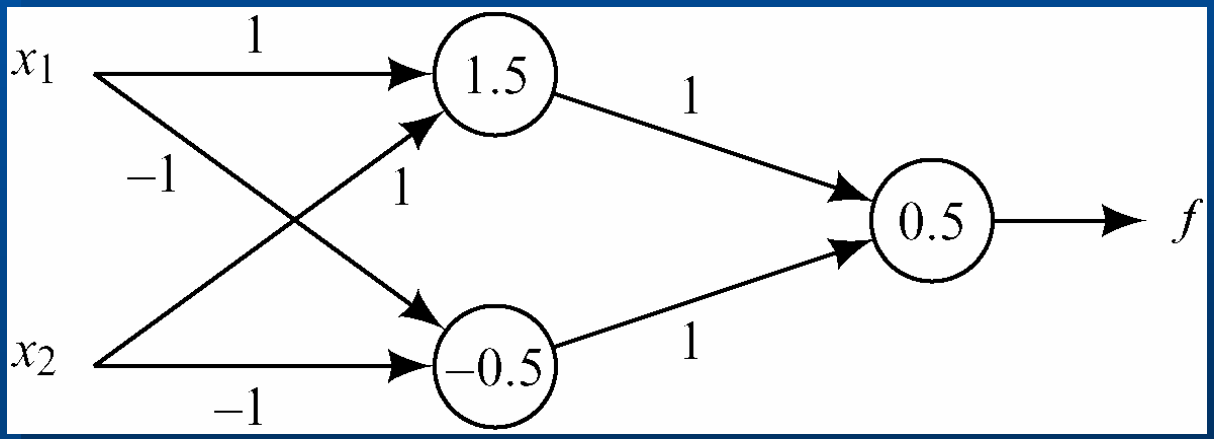


Figure 3.4 A Network of TLUs That Implements the Even-Parity Function

3.3.2 Notation

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

$$\mathbf{W}_i^{(j)} = (w_{1,i}^{(j)}, \ldots, w_{l,i}^{(j)}, \ldots, w_{m_{(j-1)}+1,i}^{(j)})$$

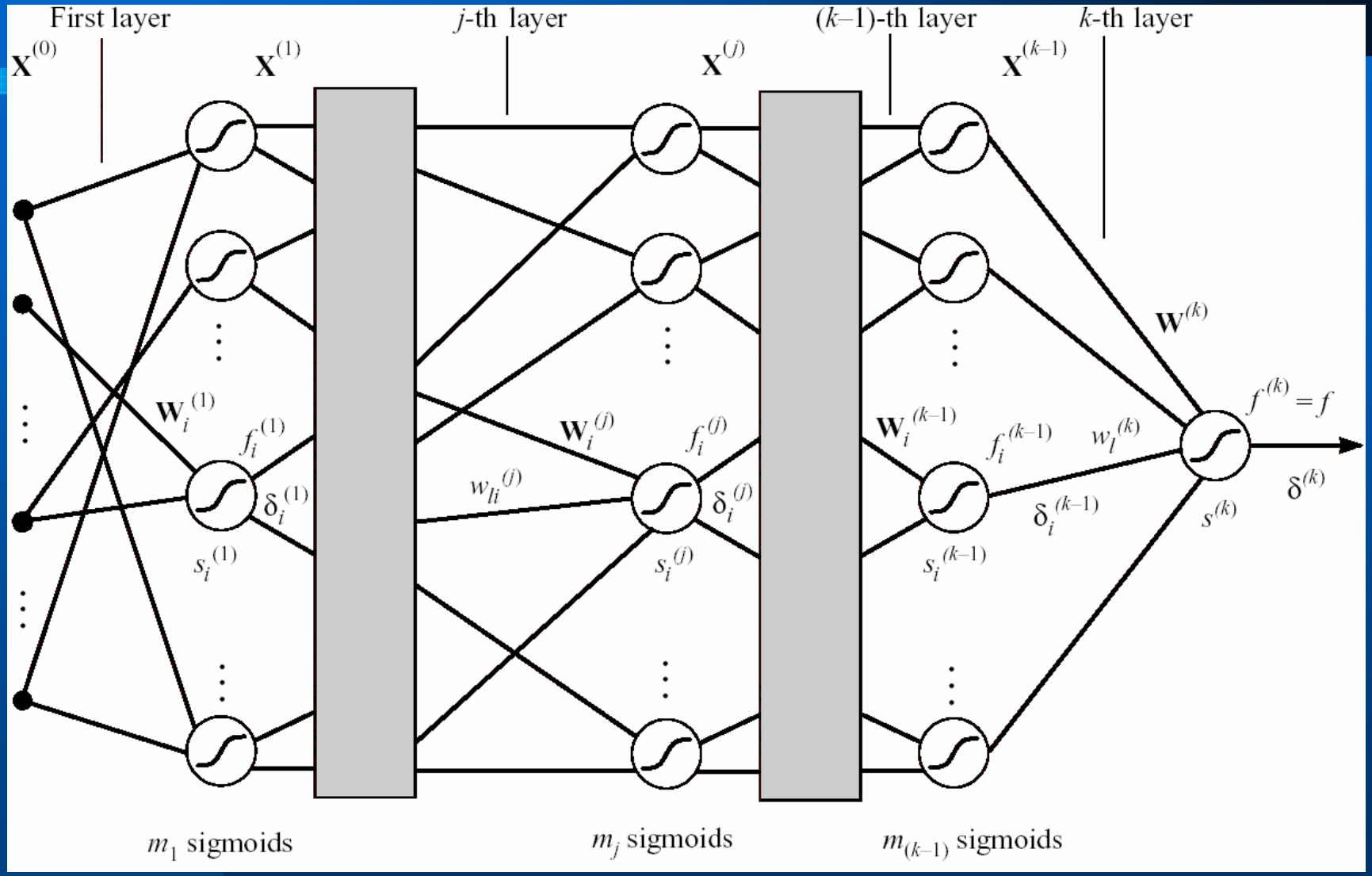


Figure 3.4 A *k*-layer Network of Sigmoid Units

From the Optimization View (1)

- Notations
  - $\mathbf{x}^{(i-1)}$: (augmented) input vector of (i)-th layer
  - $\mathbf{f}^{(i)}$: multivariate activation function of (i)-th layer
  - $m_{(i)}$: number of elements in (i)-th layer; length of $\mathbf{x}^{(i)} = m_{(i)} + 1$, length of $\mathbf{f}^{(i)} = m_{(i)}$
  - $\mathbf{W}^{(i)}$: weight matrix between (i-1)-th and (i)-th layers
    - $m_{(i-1)}+1 \times m_{(i)}$ matrix
    - $w_{jk}^{(i)}$: the value of weight between node j of (i-1)-th layer and node k of (i)-th layer
  - $\mathbf{s}^{(j)} = \mathbf{W}^{(j)T}\mathbf{x}^{(j-1)}$, $s_i^{(j)}$: $\mathbf{W}_i^{(j)} \cdot \mathbf{x}^{(j-1)}$

$$\mathbf{f}^{(i)} = (f_1^{(i)}, \cdots, f_{m_{(i)}}^{(i)})^T \qquad \mathbf{x}^{(i)} = (\mathbf{f}^{(i)}(\mathbf{s}^{(i)})^T, 1)^T \qquad \mathbf{x}^{(i)} = (\tilde{\mathbf{x}}^T, 1)^T$$

$\tilde{\mathbf{x}}$: original input vector

From the Optimization View (2)

- the generic form of optimizations: $\arg\min_{\mathbf{w}} obj$
- An objective function of feed-forward neural network
  - Squared error between output and desired output

$$E = \frac{1}{2}\sum_l \|\mathbf{d}(l) - \mathbf{f}^{(k)}(l)\|^2 \qquad \|\mathbf{x}\|^2 = \sum_j x_j^2$$

- In the optimization view, learning NN is just one of minimization problem.

$$\arg\min_{\mathcal{W}=\{\mathbf{W}^{(1)},\cdots,\mathbf{W}^{(k)}\}} \frac{1}{2}\sum_l \|\mathbf{d}(l) - \mathbf{f}^{(k)}(l)\|^2$$

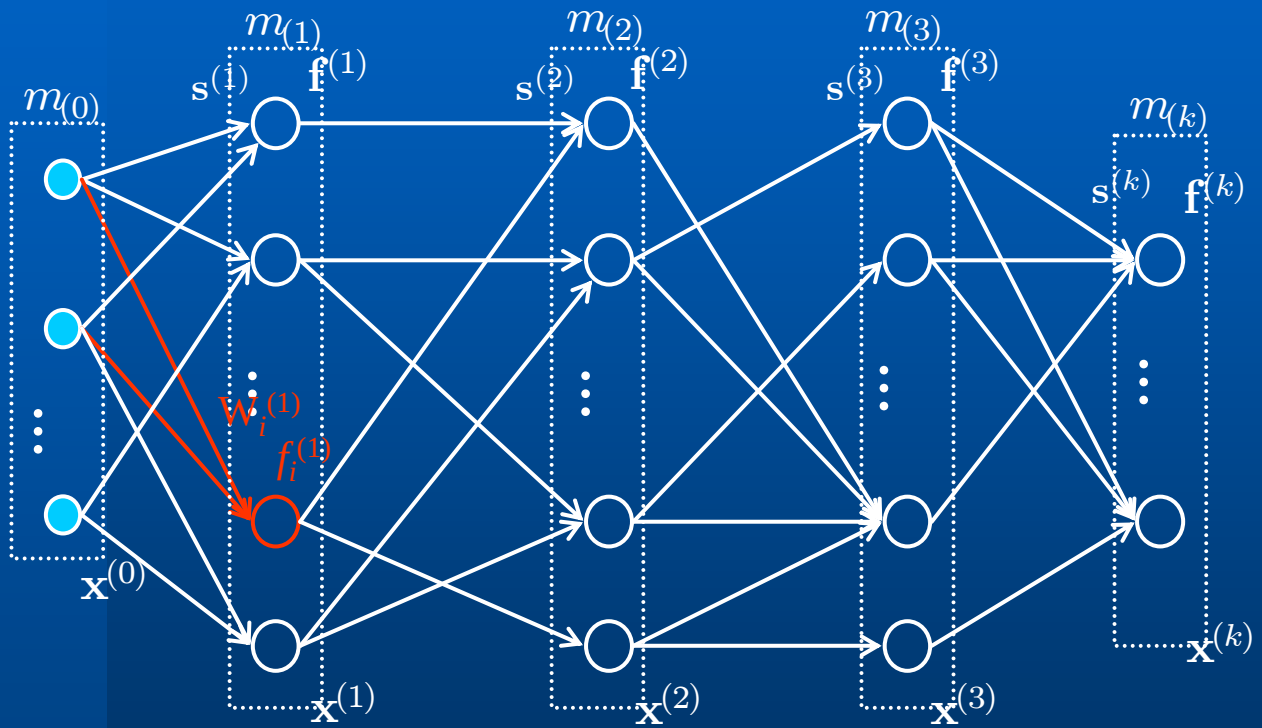
- General procedure to minimize the objective

$$\mathcal{W} \leftarrow \mathcal{W} - \eta\frac{\partial E}{\partial \mathcal{W}}$$

- Only thing we must do is to calculate the gradient: backpropagation does it!

From the Optimization View (3)

- Feed-forward step: calculate $\mathbf{f}^{(k)}$.



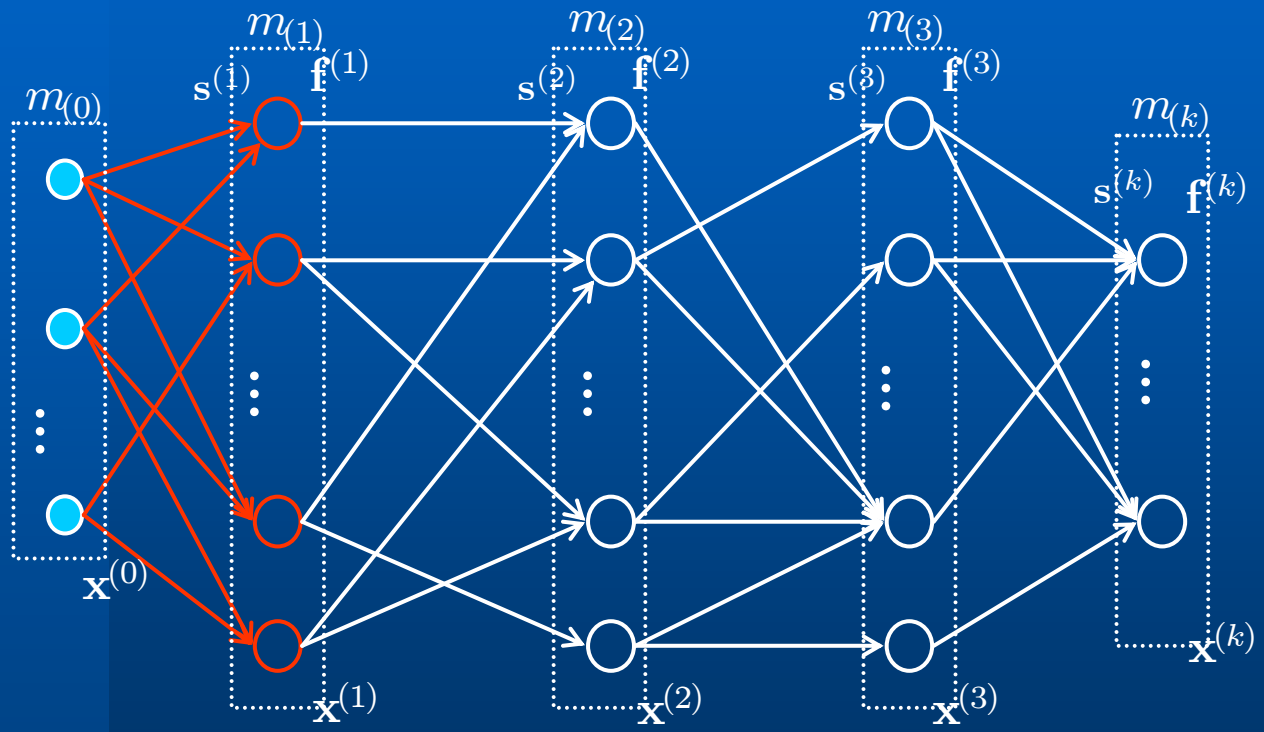
$$\tilde{x}_i^{(1)} = f_i^{(1)}(\mathbf{W}_i^{(1)} \cdot \mathbf{x}^{(0)})$$

$$\mathbf{W}_i^{(j)} = (w_{1,i}^{(j)}, \ldots, w_{l,i}^{(j)}, \ldots, w_{m_{(j-1)}+1,i}^{(j)})$$

Assumption: the activation function of each neuron in a layer is equivalent.

From the Optimization View (3)

- Feed-forward step: calculate $\mathbf{f}^{(1)}$.

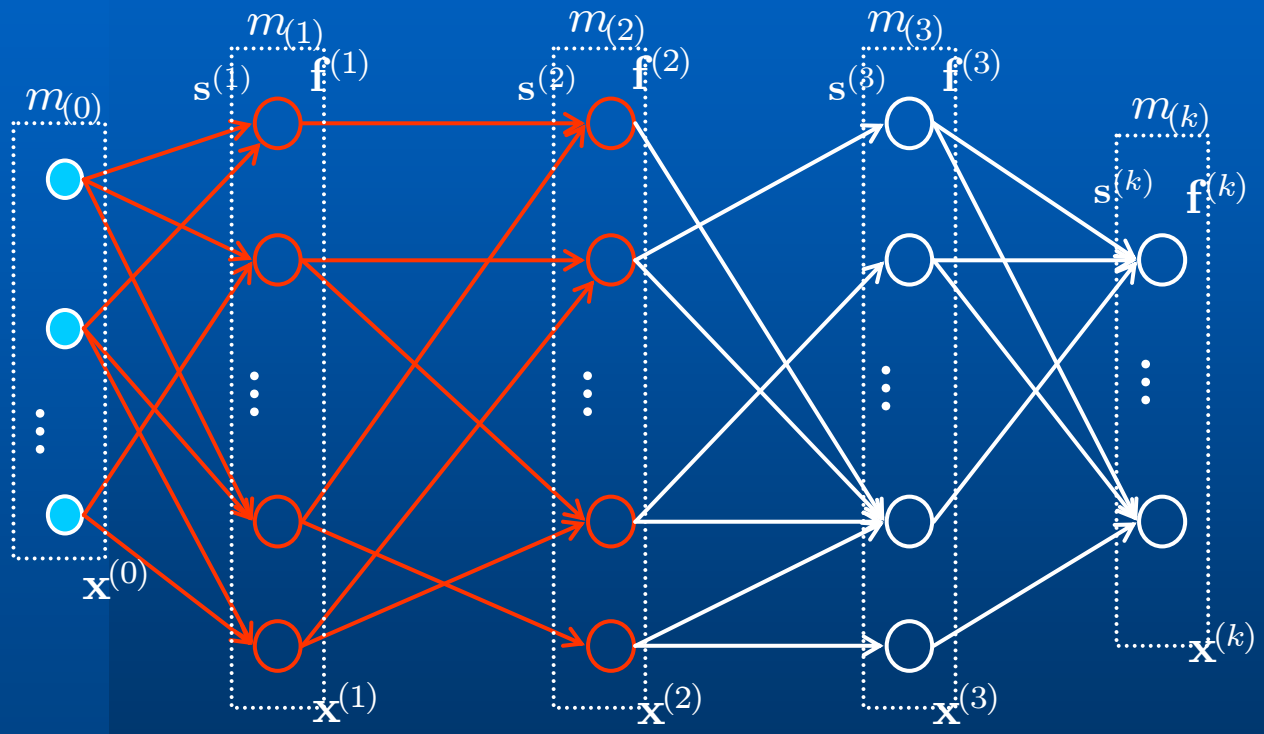


$$\tilde{\mathbf{x}}^{(1)} = \mathbf{f}^{(1)}(\mathbf{W}^{(1)T}\mathbf{x}^{(0)})$$

If we use vectorized activation function: $\mathbf{f}^{(i)}(\mathbf{x}) = (f^{(i)}(x_1), \cdots, f^{(i)}(x_{m_i}))^T$

From the Optimization View (3)

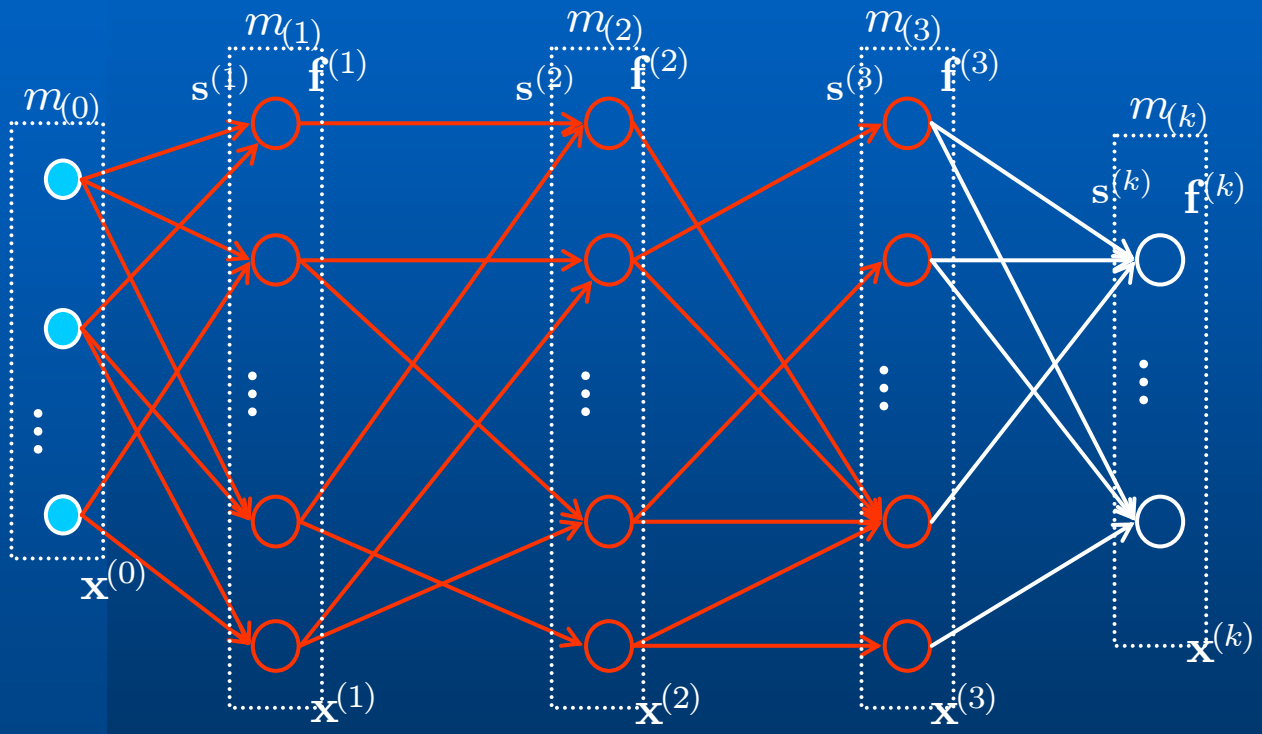
- Feed-forward step: calculate $\mathbf{f}^{(2)}$.



$$\tilde{\mathbf{x}}^{(2)} = \mathbf{f}^{(2)}(\mathbf{W}^{(2)T}\mathbf{x}^{(1)})$$

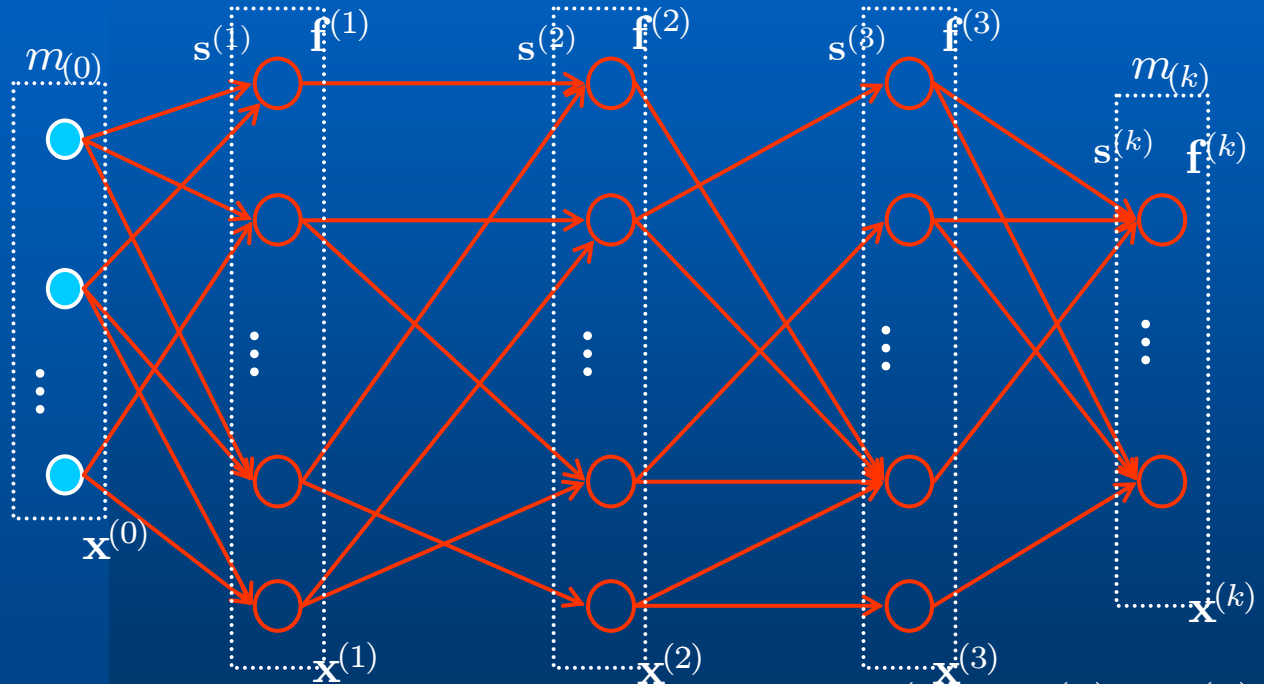
From the Optimization View (3)

- Feed-forward step: calculate $\mathbf{f}^{(3)}$.



From the Optimization View (3)

- Feed-forward step: calculate $\mathbf{f}^{(k)}$.



$$\tilde{\mathbf{x}}^{(k)} = \mathbf{f}^{(k)}\left(\mathbf{W}^{(k)T}\mathbf{x}^{(k-1)}\right)$$

From the Optimization View (4)

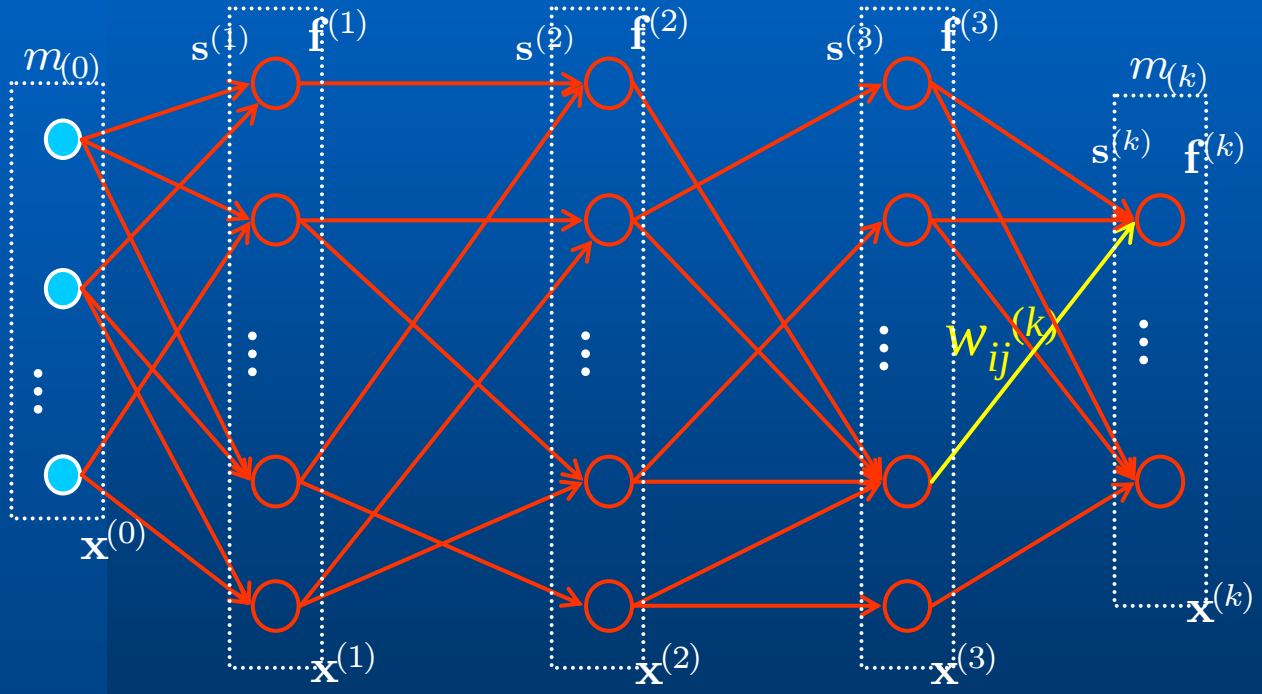
- Completion of feed-forward step
  - For every example-pair, repeat above step to calculate $\mathbf{f}^{(k)}(l)$.
  - We can now calculate error:

$$E = \sum_l E_l = \frac{1}{2} \sum_l \|\mathbf{d}(l) - \mathbf{f}^{(k)}(l)\|^2$$

- Back-propagation step
  - From k-th layer to input layer, the gradient $\frac{\partial E}{\partial \mathbf{W}^{(i)}}$ is calculated using chain rule.

From the Optimization View (4)

- Backpropagation at output layer: $\frac{\partial E_l}{\partial \mathbf{W}^{(k)}}$



- That means to calculate $\frac{\partial E_l}{\partial w_{ij}^{(k)}} = \frac{\partial E_l}{\partial f_j^{(k)}} \frac{\partial f_j^{(k)}}{\partial w_{ij}^{(k)}}$

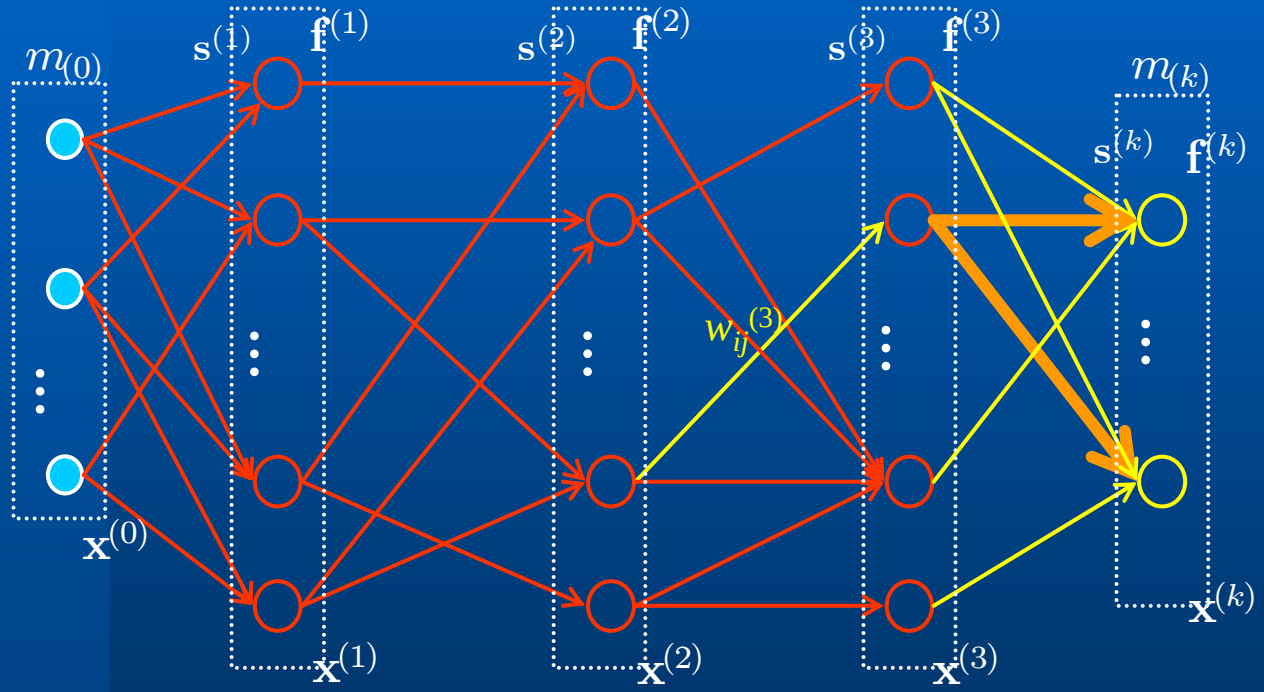
From the Optimization View (4)

- Chain rule
  - E_l is a function of $f^{(k)}$; $f^{(k)}$ is a function of $w_{ij}^{(k)}$.
  - E_l is an indirect function of w of which gradient can be calculated via chain rule.

$$\frac{\partial E_l}{\partial w_{ij}^{(k)}} = \frac{\partial E_l}{\partial f_j^{(k)}} \frac{\partial f_j^{(k)}}{\partial w_{ij}^{(k)}} = -(d_j - f_j^{(k)}) \frac{\partial f_j^{(k)}}{\partial w_{ij}^{(k)}}$$

From the Optimization View (5)

- Backpropagation at hidden layer: $\frac{\partial E_l}{\partial \mathbf{W}^{(3)}}$

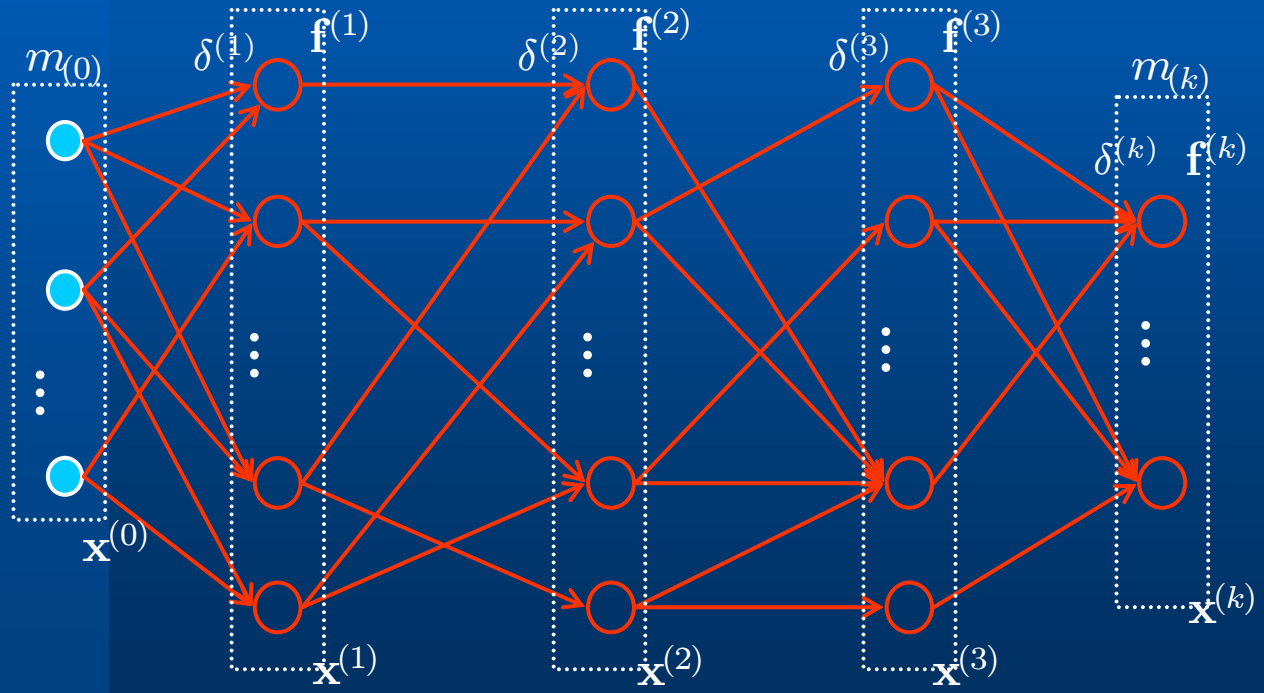


$$\frac{\partial E_l}{\partial w_{ij}^{(3)}} = \frac{\partial E_l}{\partial f_j^{(3)}} \frac{\partial f_j^{(3)}}{\partial w_{ij}^{(3)}} = \sum_h \frac{\partial E_l}{\partial f_h^{(k)}} \frac{\partial f_h^{(k)}}{\partial f_j^{(3)}} \frac{\partial f_j^{(3)}}{\partial w_{ij}^{(3)}}$$

$f_j^{(3)}$ is a function of $w_{ij}^{(3)}$.
Each $f_h^{(k)}$ is a function $f_j^{(3)}$.

From the Optimization View (5)

- Intermediate device for recursive equation
  - $\delta_i^{(j)} = \partial E_l / \partial s_i^{(j)}$
  - $\delta^{(j)} = (\delta_1^{(j)}, \cdots, \delta_{m_j}^{(j)})^T$



From the Optimization View (5)

- Recursive relation

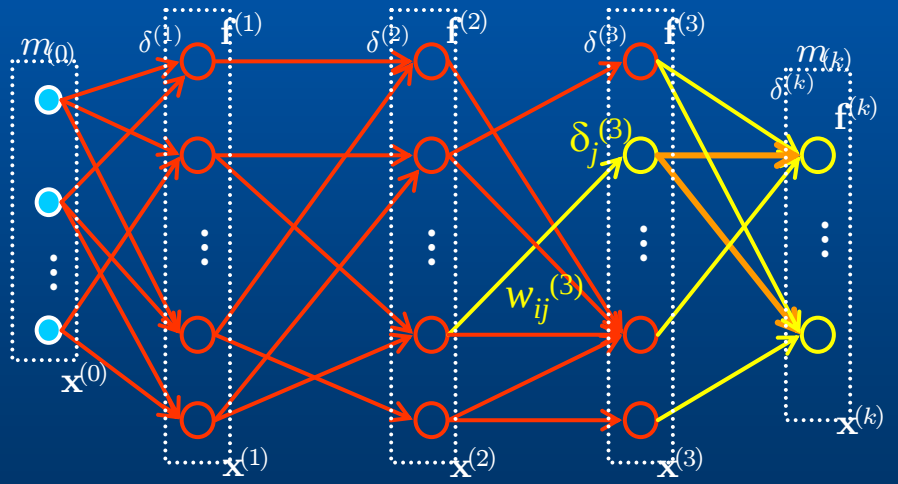
$$\frac{\partial E_l}{\partial w_{ij}^{(3)}} = \frac{\partial E_l}{\partial s_j^{(3)}} \frac{\partial s_j^{(3)}}{\partial w_{ij}^{(3)}} = \sum_h \frac{\partial E_l}{\partial s_h^{(k)}} \frac{\partial s_h^{(k)}}{\partial s_j^{(3)}} \frac{\partial s_j^{(3)}}{\partial w_{ij}^{(3)}}$$

$$\frac{\partial s_h^{(k)}}{\partial s_j^{(3)}} = \frac{\partial s_h^{(k)}}{\partial x_j^{(3)}} \frac{\partial x_j^{(3)}}{\partial s_j^{(3)}} = w_{jh}^{(k)} \frac{\partial f_j^{(3)}}{\partial s_j^{(3)}}$$

$$\frac{\partial s_j^{(3)}}{\partial w_{ij}^{(3)}} = x_i^{(2)}$$

$$\delta_j^{(3)} = \frac{\partial f_j^{(3)}}{\partial s_j^{(3)}} \sum_h \delta_h^{(k)} w_{jh}^{(k)}$$

$$\frac{\partial E_l}{\partial w_{ij}^{(3)}} = \delta_j^{(3)} x_i^{(2)}$$



From the Optimization View (6)

- Output layer

$$\delta_i^{(k)} = \frac{\partial E_l}{\partial s_i^{(k)}} = \frac{\partial E_l}{\partial f_i^{(k)}} \frac{\partial f_i^{(k)}}{\partial s_i^{(k)}} = -(d_i - f_i^{(k)}) \frac{\partial f_i^{(k)}}{\partial s_i^{(k)}}$$

- Hidden layer

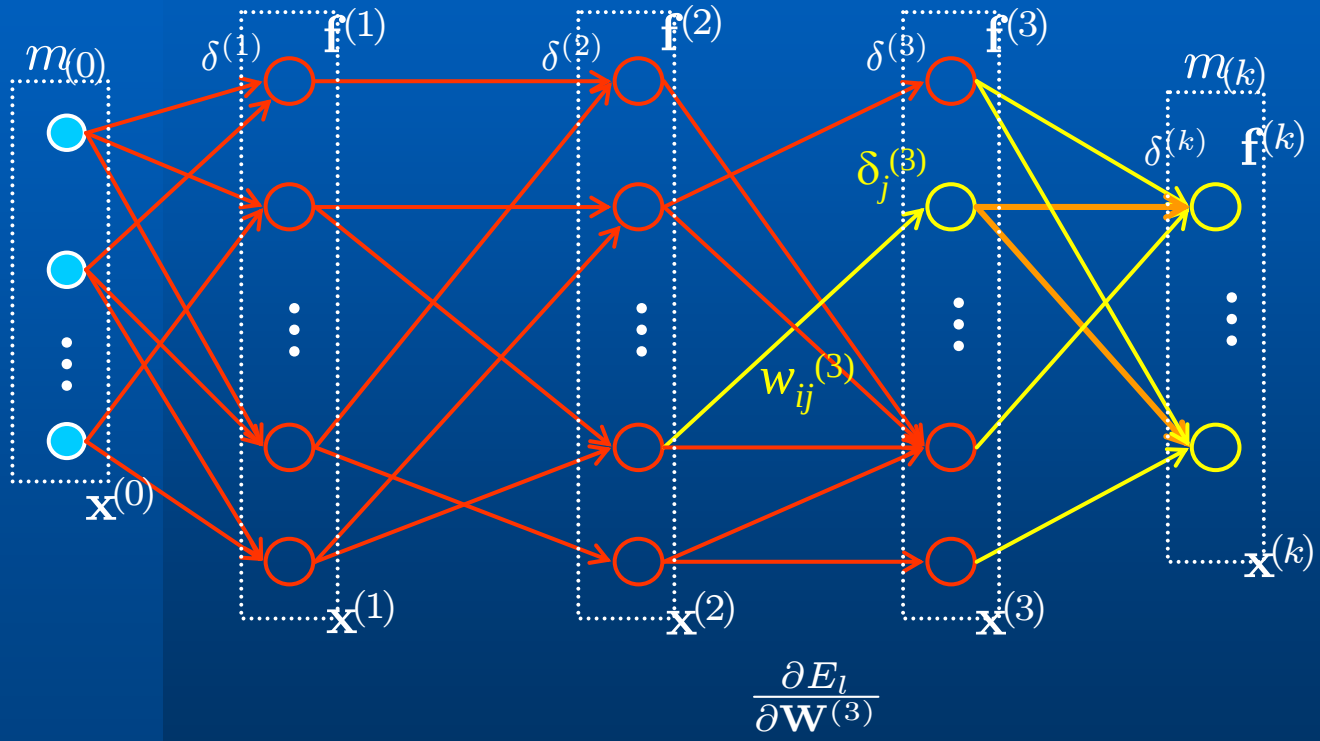
$$\delta_j^{(k)} = \frac{\partial f_j^{(k)}}{\partial s_j^{(k)}} \sum_h \delta_h^{(k+1)} w_{jh}^{(k+1)}$$

- Gradient

$$\frac{\partial E_l}{\partial w_{ij}^{(k)}} = \delta_j^{(k)} x_i^{(k-1)}$$

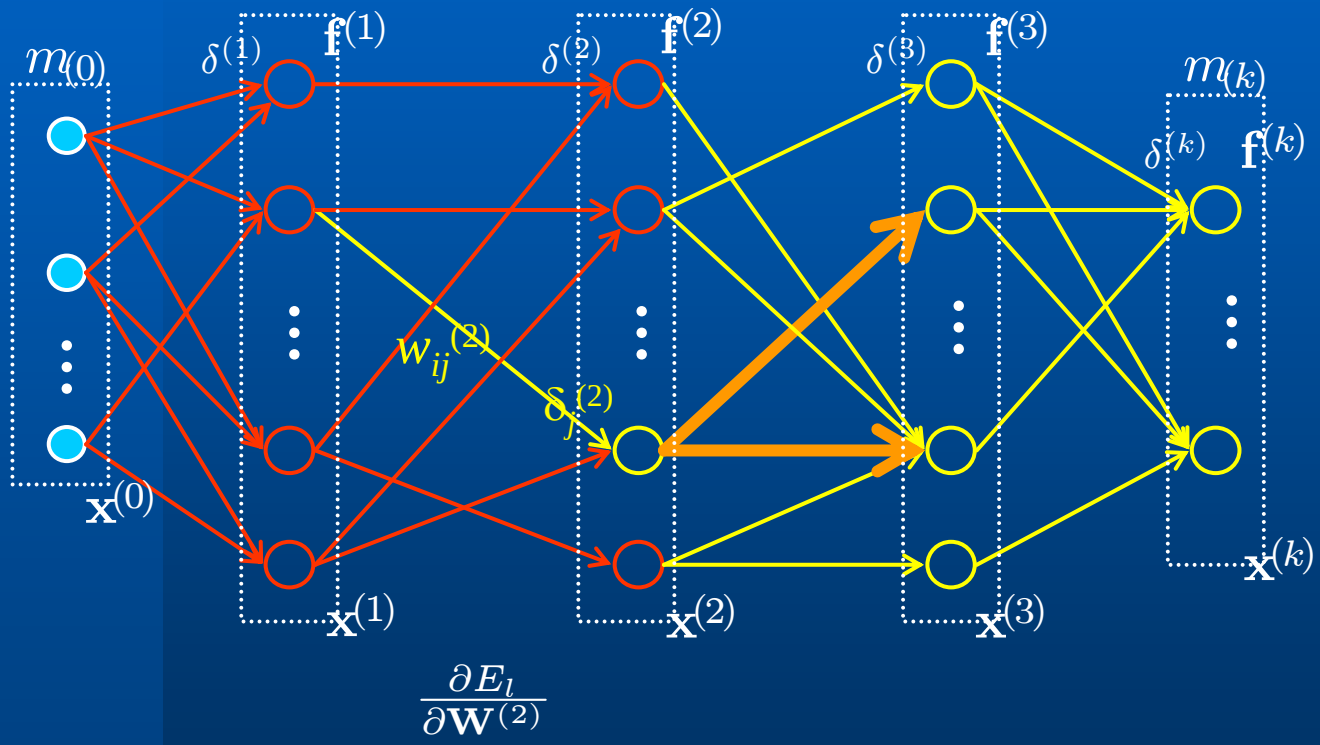
From the Optimization View (6)

- Backpropagation step at 3-th layer



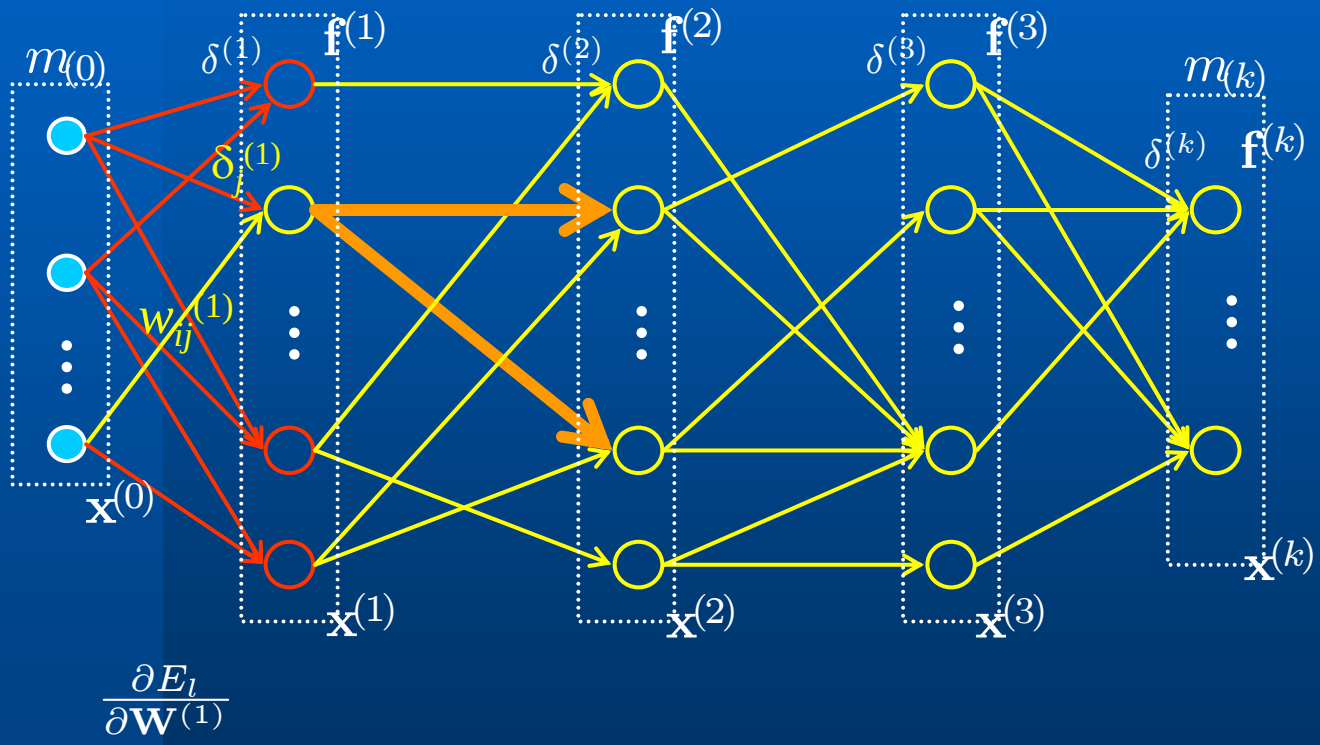
From the Optimization View (6)

- Backpropagation step at 2-nd layer



From the Optimization View (6)

- Backpropagation step at 1-st layer



Summary of backpropagation

- To find a solution of

$$\arg\min_{\mathcal{W}=\{\mathbf{W}^{(1)},\cdots,\mathbf{W}^{(k)}\}} \frac{1}{2}\sum_l \|\mathbf{d}(l)-\mathbf{f}^{(k)}(l)\|^2$$

  - We can use gradient descent method.

$$\mathcal{W} \leftarrow \mathcal{W} - \eta\frac{\partial E}{\partial \mathcal{W}}$$

- The gradient is calculated by backpropagation method.
- Algorithm
  - Feedforward *l*-th input of examples.
  - Backpropagate to get the gradient
  - Compose the gradient $\frac{\partial E_l}{\partial \mathcal{W}}$
  - Update weights. $\frac{\partial E}{\partial \mathcal{W}} = \sum_l \frac{\partial E_l}{\partial \mathcal{W}}$
  - Loop until the error is minimized.

3.3.5 (con't)

- Example (even parity function)

input			target
1	0	1	0
0	0	1	1
0	1	1	0
1	1	1	1

  - Learning rate: 1.0

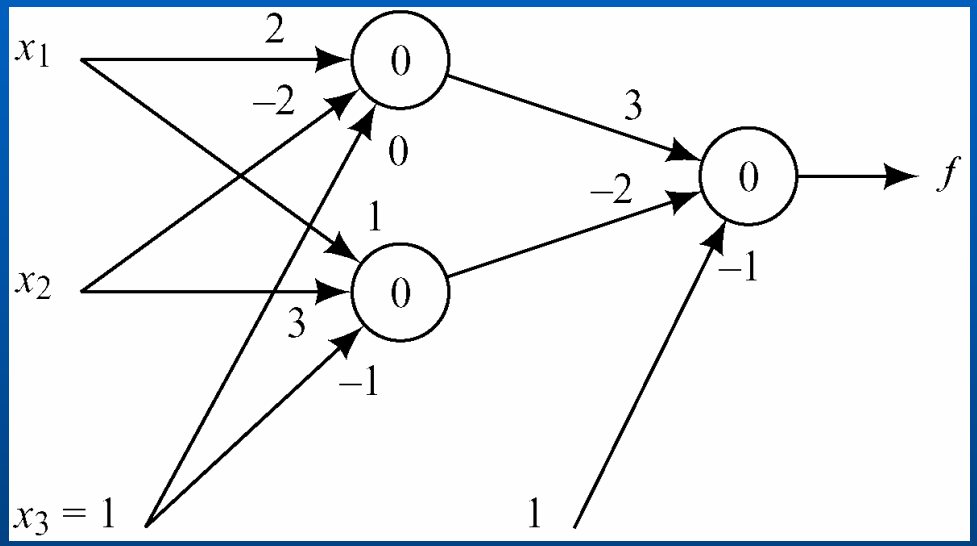


Figure 3.6 A Network to Be Trained by Backprop

3.4 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*-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.

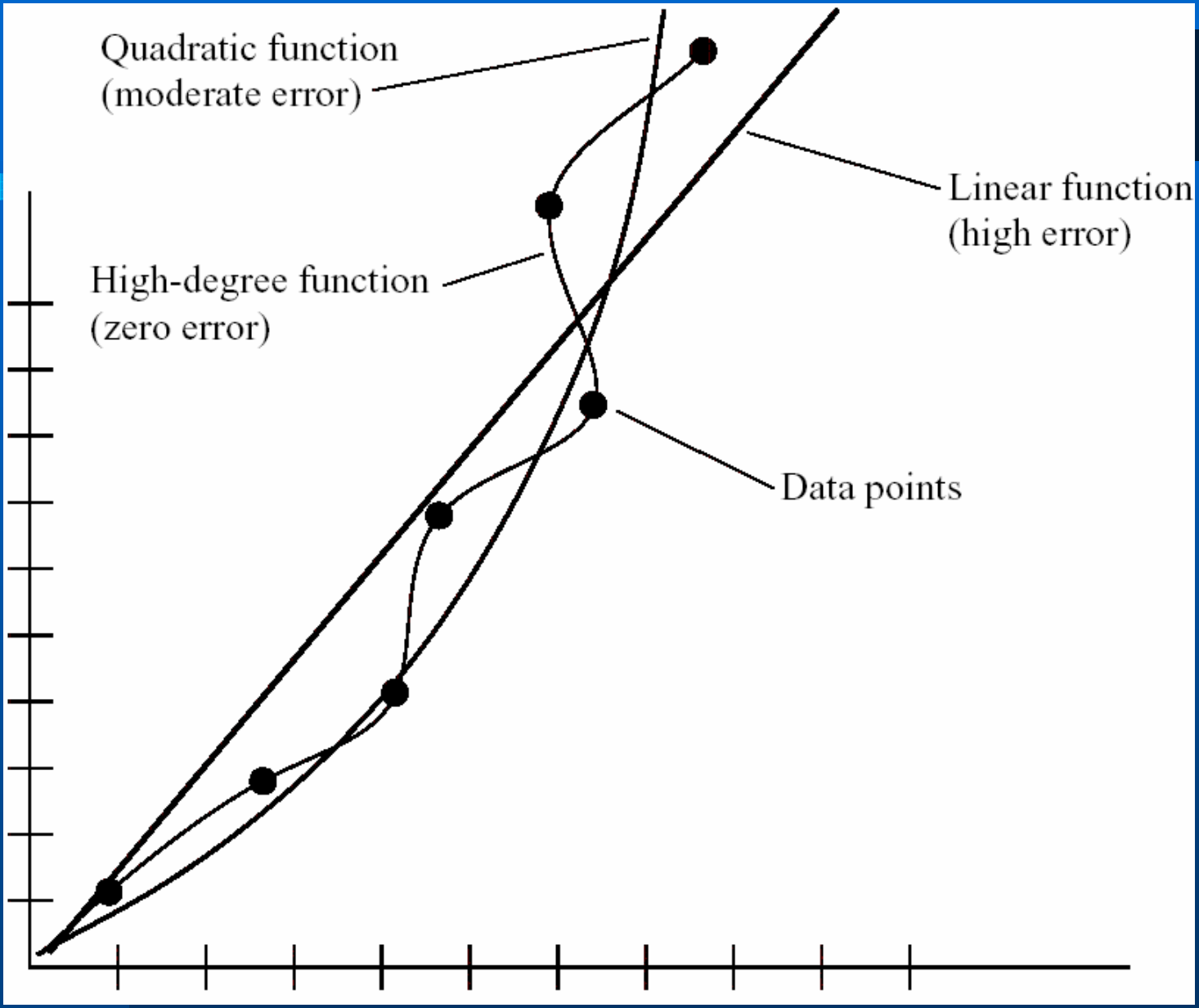


Figure 3.7 Curve Fitting

3.4 (cont'd)

- 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.

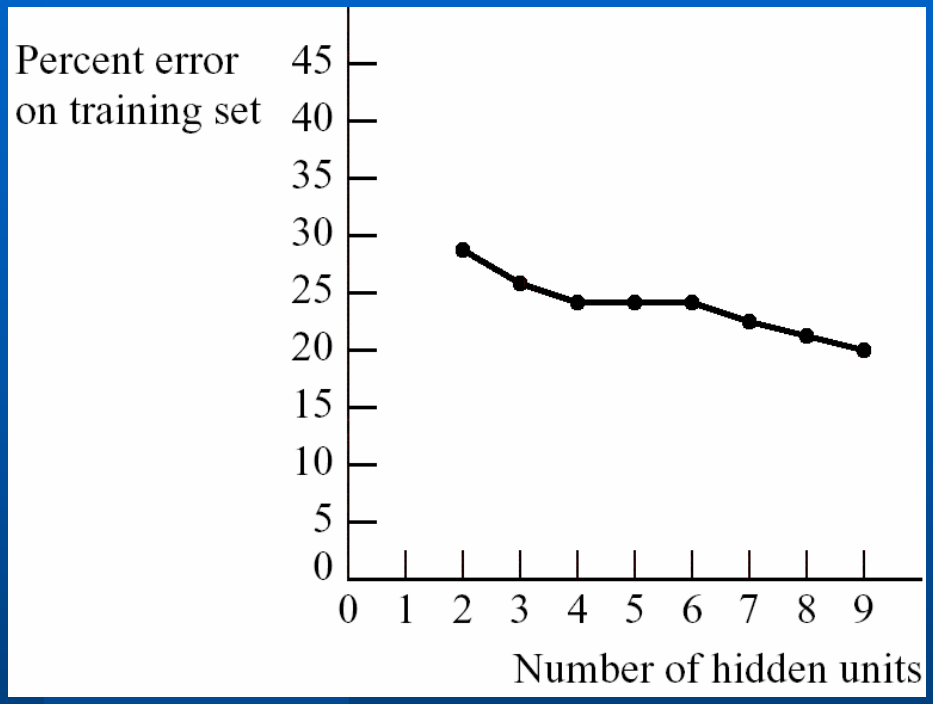


Figure 3.8 Error Versus Number of Hidden Units

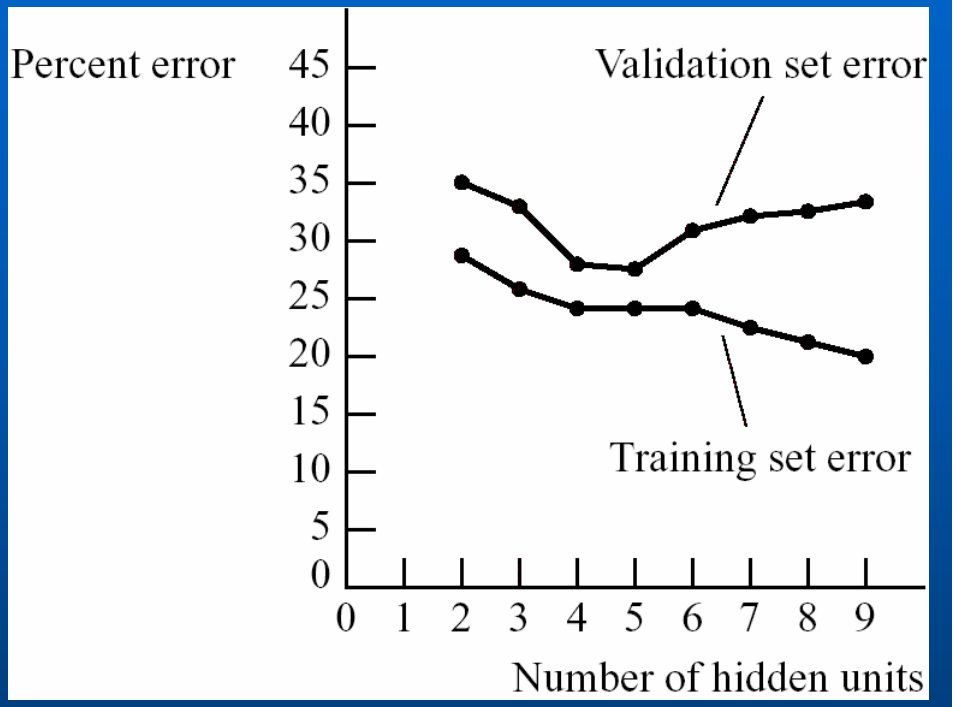


Fig 3.9 Estimate of Generalization Error Versus Number of Hidden Units

3.5 Additional Readings and Discussion

- Applications
  - Pattern recognition, automatic control, brain-function modeling
  - Designing and training neural networks still need experience and experiments.
- Major annual conferences
  - Neural Information Processing Systems (NIPS)
  - International Conference on Machine Learning (ICML)
  - Computational Learning Theory (COLT)
- Major journals
  - Neural Computation
  - IEEE Transactions on Neural Networks
  - Machine Learning