Introduction to Machine Learning

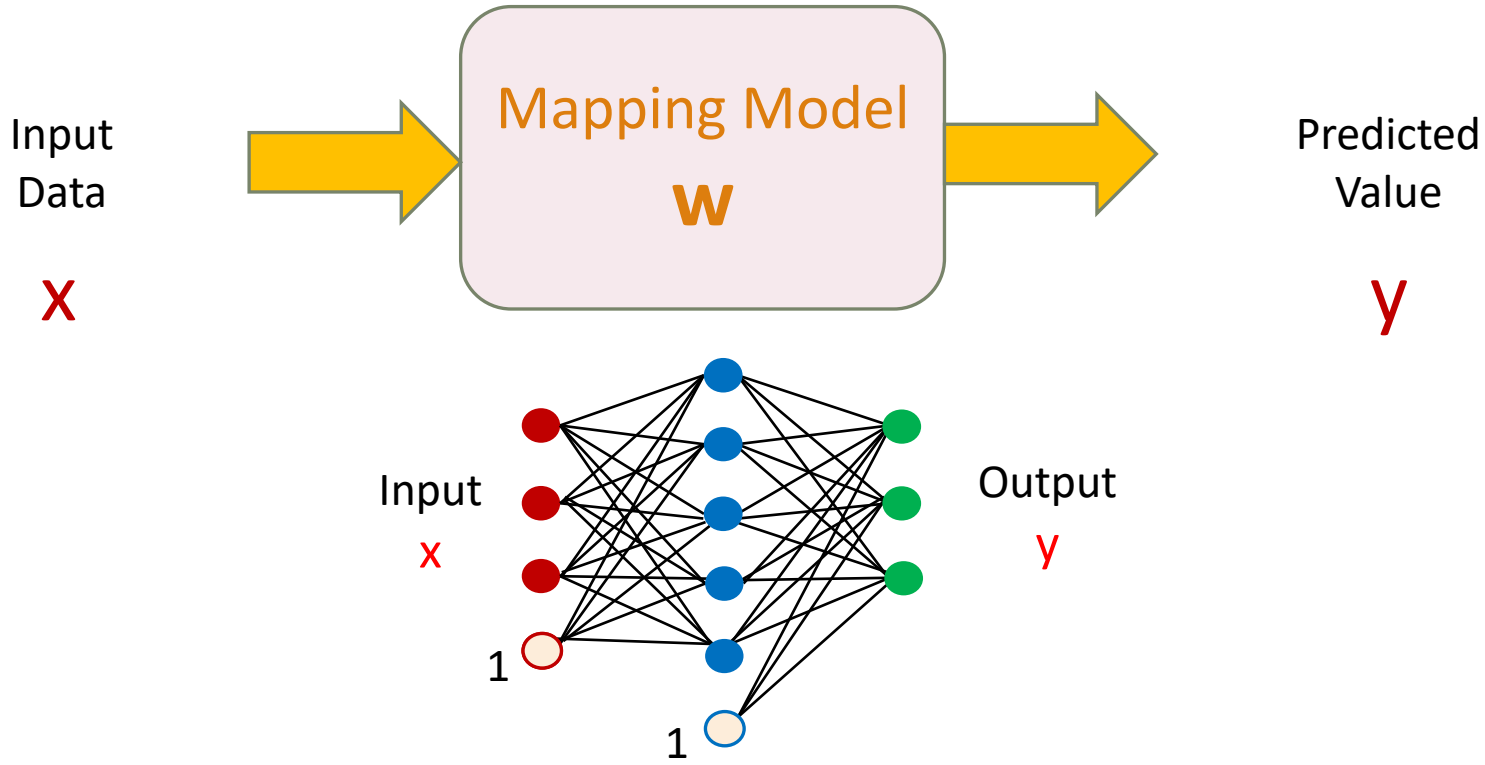
Neural Networks

SHENG-JYH WANG

NATIONAL YANG MING CHIAO TUNG UNIVERSITY, TAIWAN

SPRING, 2024

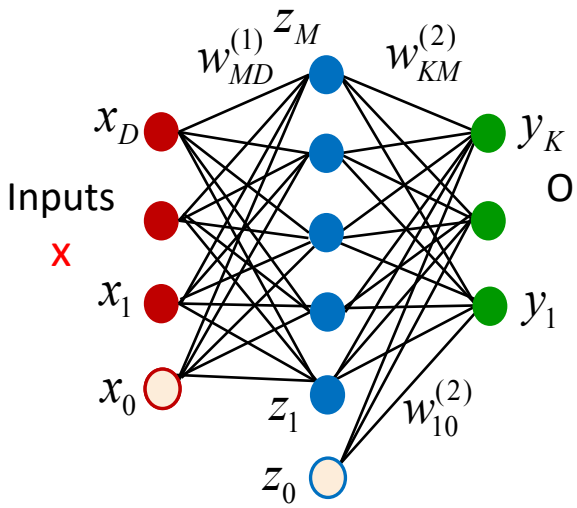
Feed-forward Neural Network (1/10)



Remark: Neural Networks are also known as ***Multilayer Perceptron***

Feed-forward Neural Network (2/10)

Two-layer Neural Network



Hidden Layer

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

$$z_j = h(a_j) \qquad j = 1, \ldots, M$$

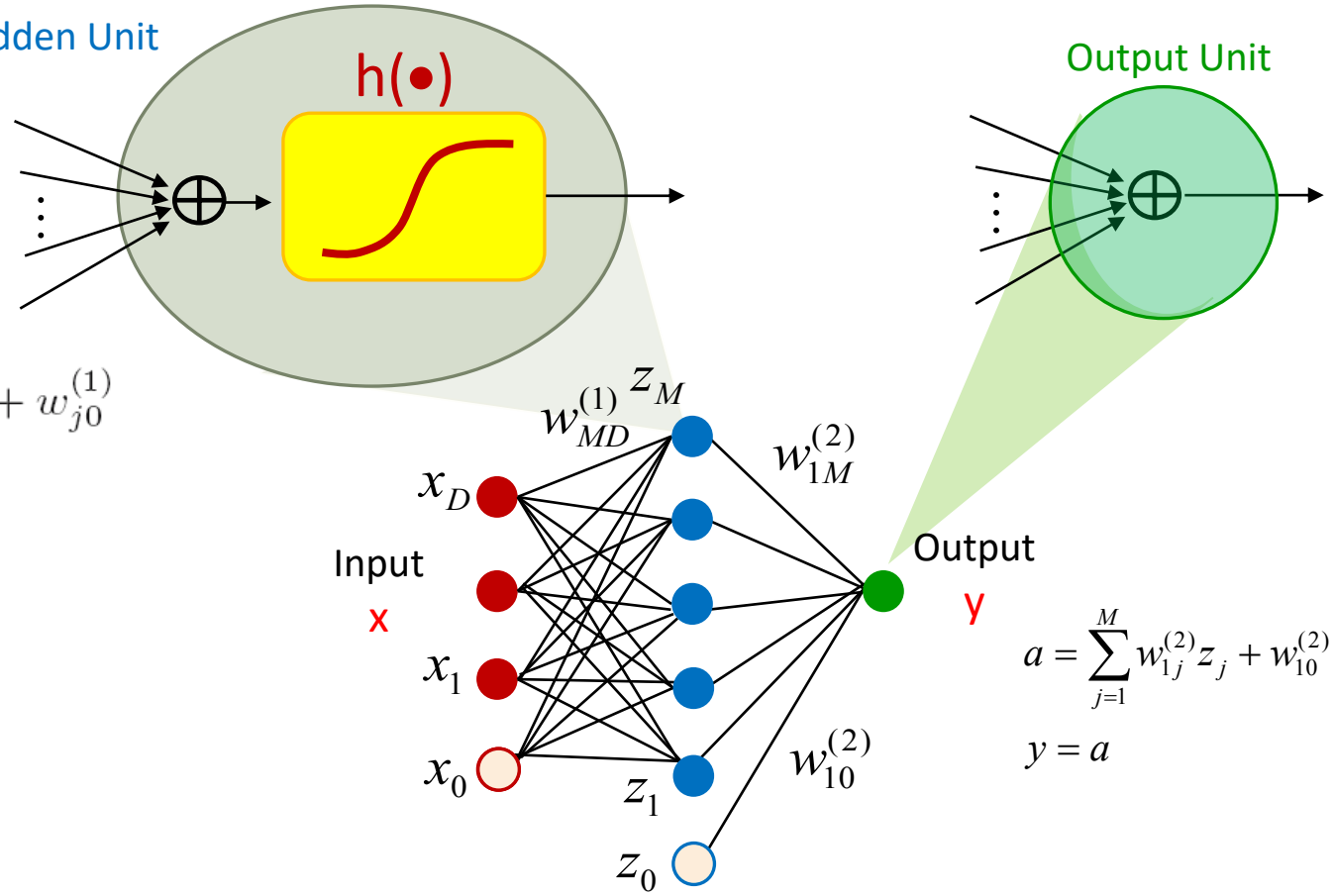
$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \qquad k = 1, \ldots, K$$

$$y_k = \sigma(a_k)$$

Output Layer

Feed-forward Neural Network (3/10)

Regression Model



$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

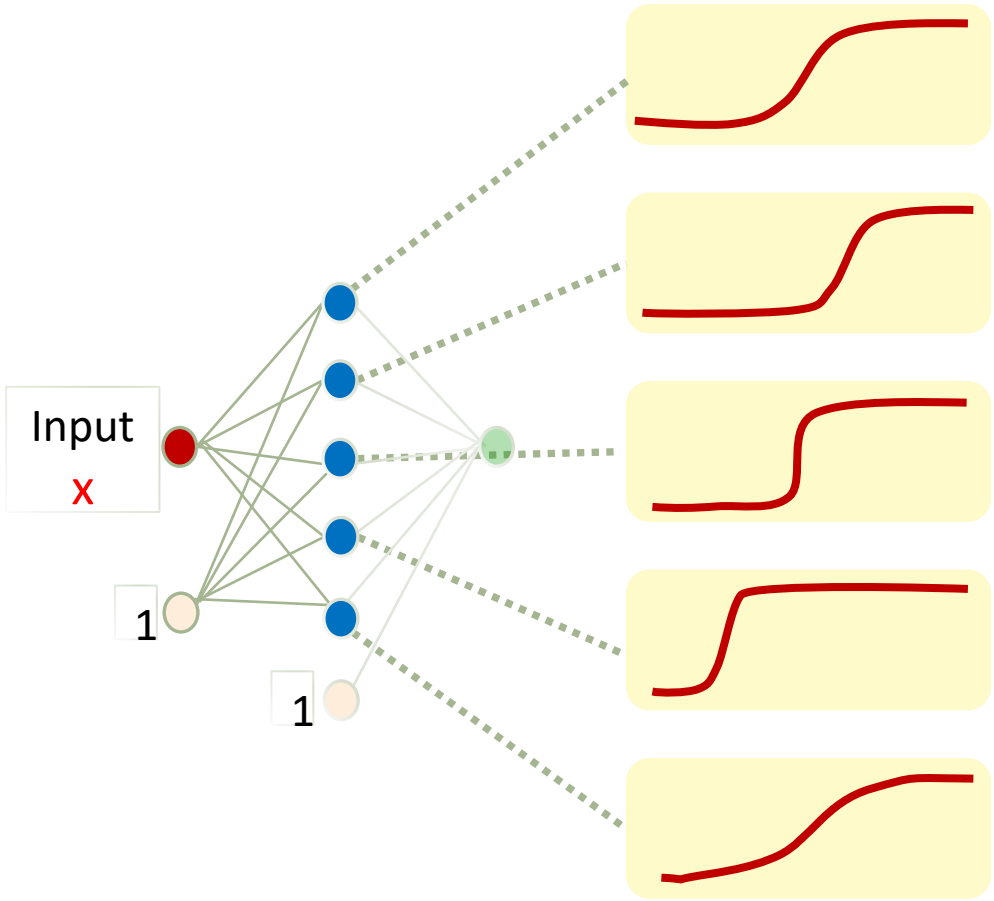
$j = 1, \ldots, M$

$$z_j = h(a_j)$$

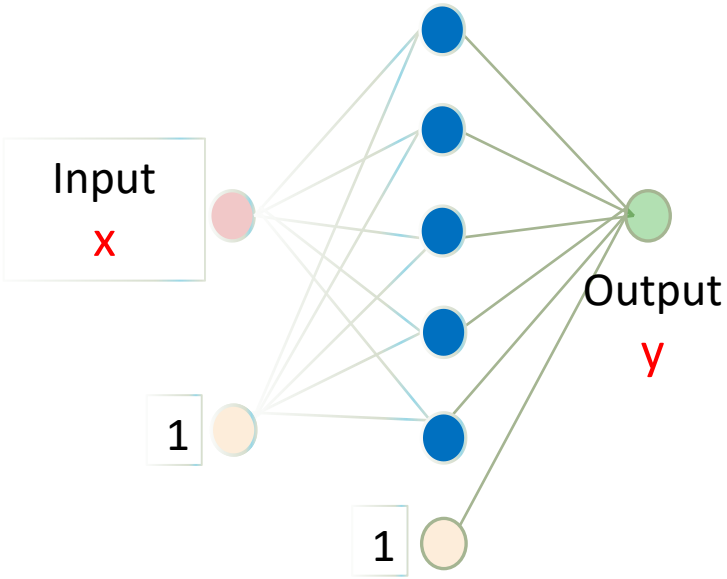
$$a = \sum_{j=1}^{M} w_{1j}^{(2)} z_j + w_{10}^{(2)}, \quad k = 1,\ldots,K$$

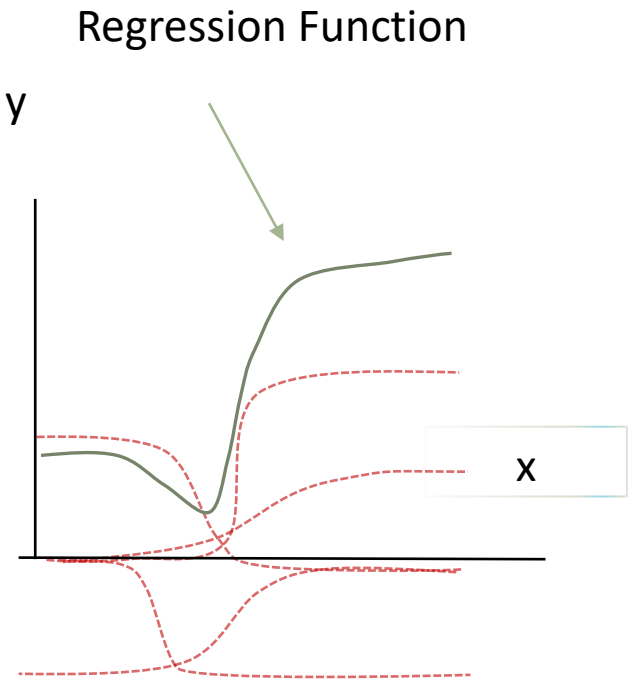
$$y = a$$

Feed-forward Neural Network (4/10)



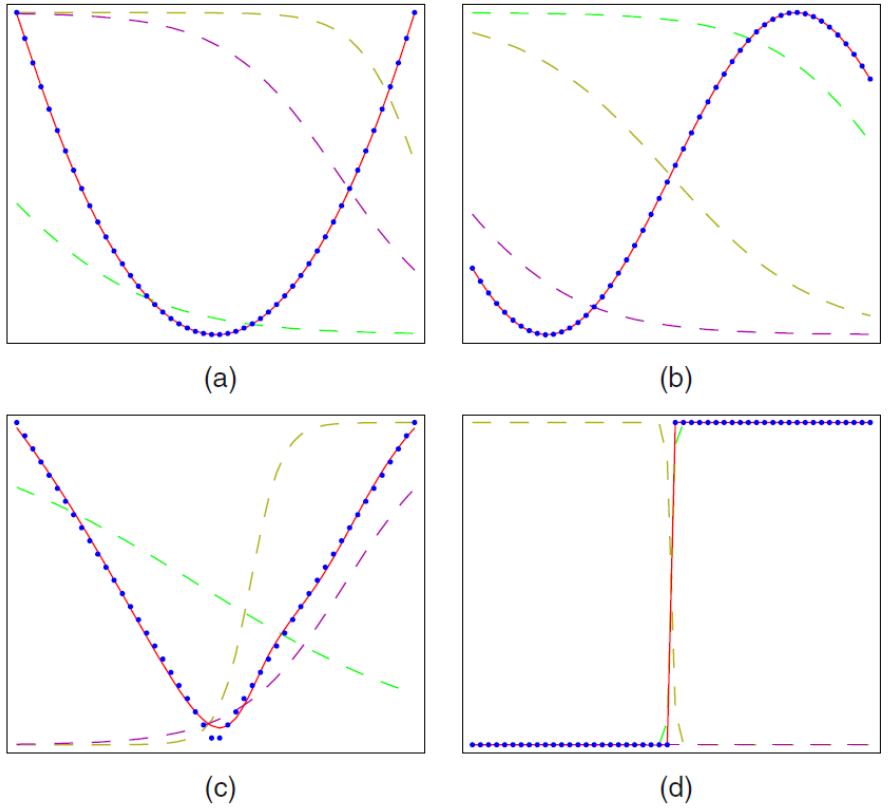
Feed-forward Neural Network (5/10)





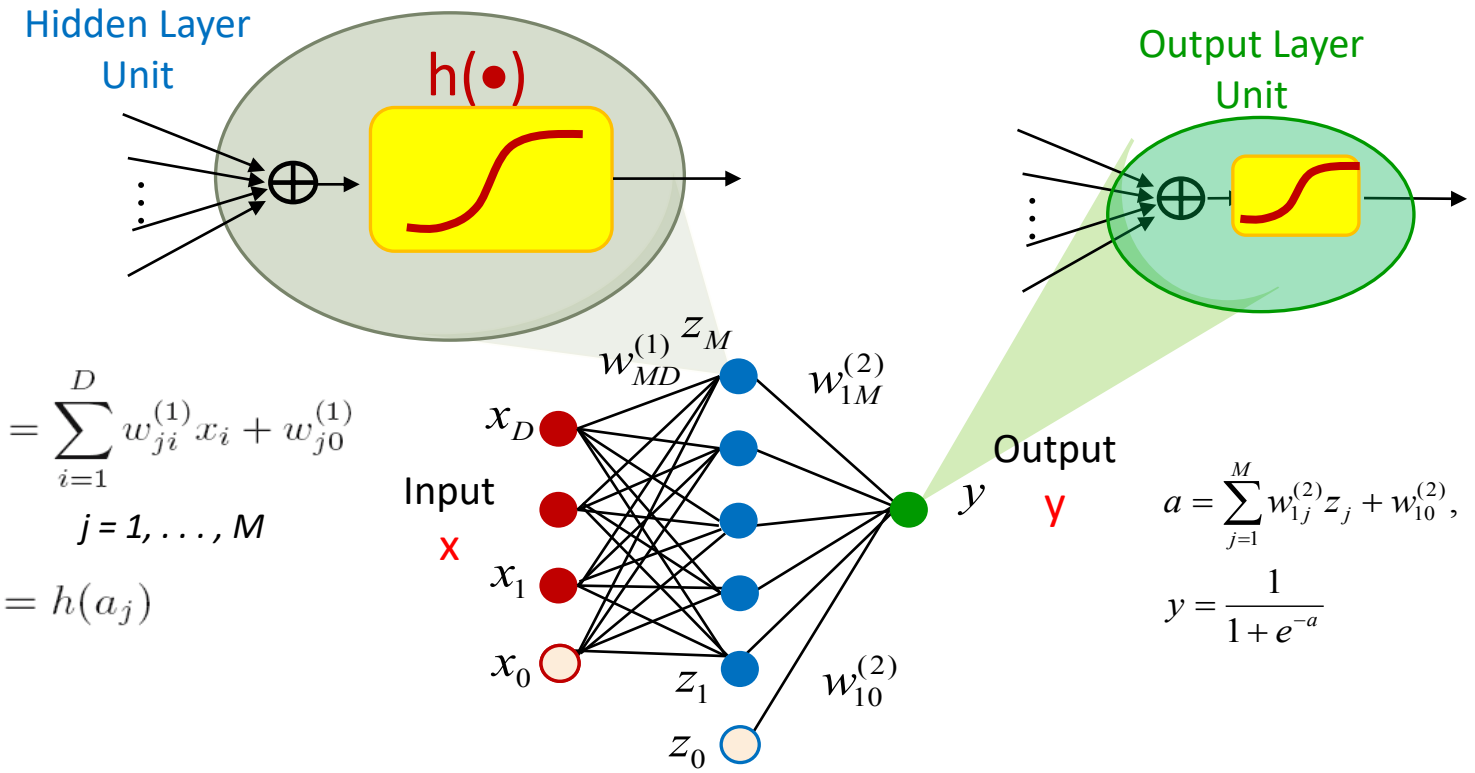
Feed-forward Neural Network (6/10)

Figure 5.3 Illustration of the capability of a multilayer perceptron to approximate four different functions comprising (a) $f(x) = x^2$, (b) $f(x) = \sin(x)$, (c), $f(x) = |x|$, and (d) $f(x) = H(x)$ where $H(x)$ is the Heaviside step function. In each case, $N = 50$ data points, shown as blue dots, have been sampled uniformly in x over the interval $(-1, 1)$ and the corresponding values of $f(x)$ evaluated. These data points are then used to train a two-layer network having 3 hidden units with 'tanh' activation functions and linear output units. The resulting network functions are shown by the red curves, and the outputs of the three hidden units are shown by the three dashed curves.



Feed-forward Neural Network (7/10)

Binary Classification Model



$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

$j = 1, \ldots, M$

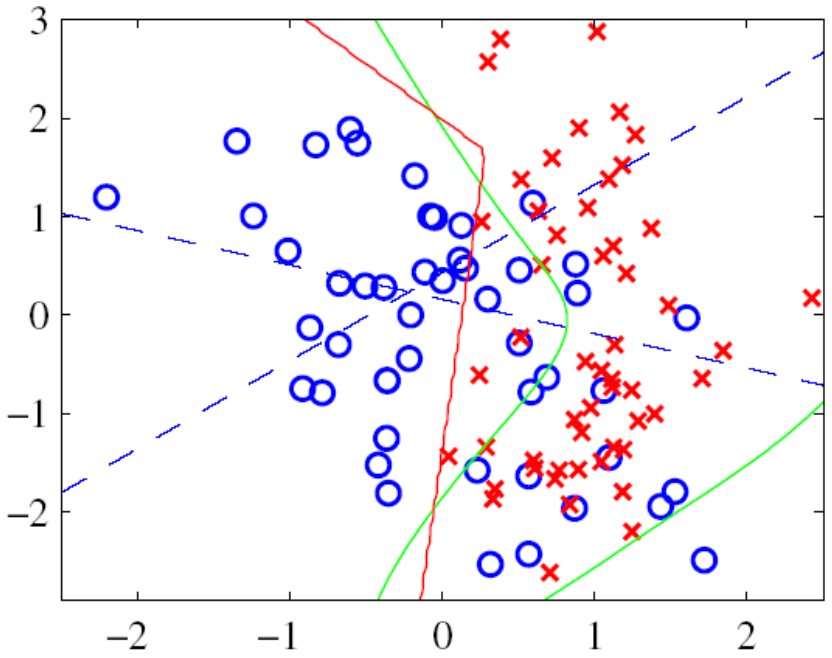
$$z_j = h(a_j)$$

$$a = \sum_{j=1}^{M} w_{1j}^{(2)} z_j + w_{10}^{(2)}, \quad k = 1,\ldots,K$$

$$y = \frac{1}{1+e^{-a}}$$

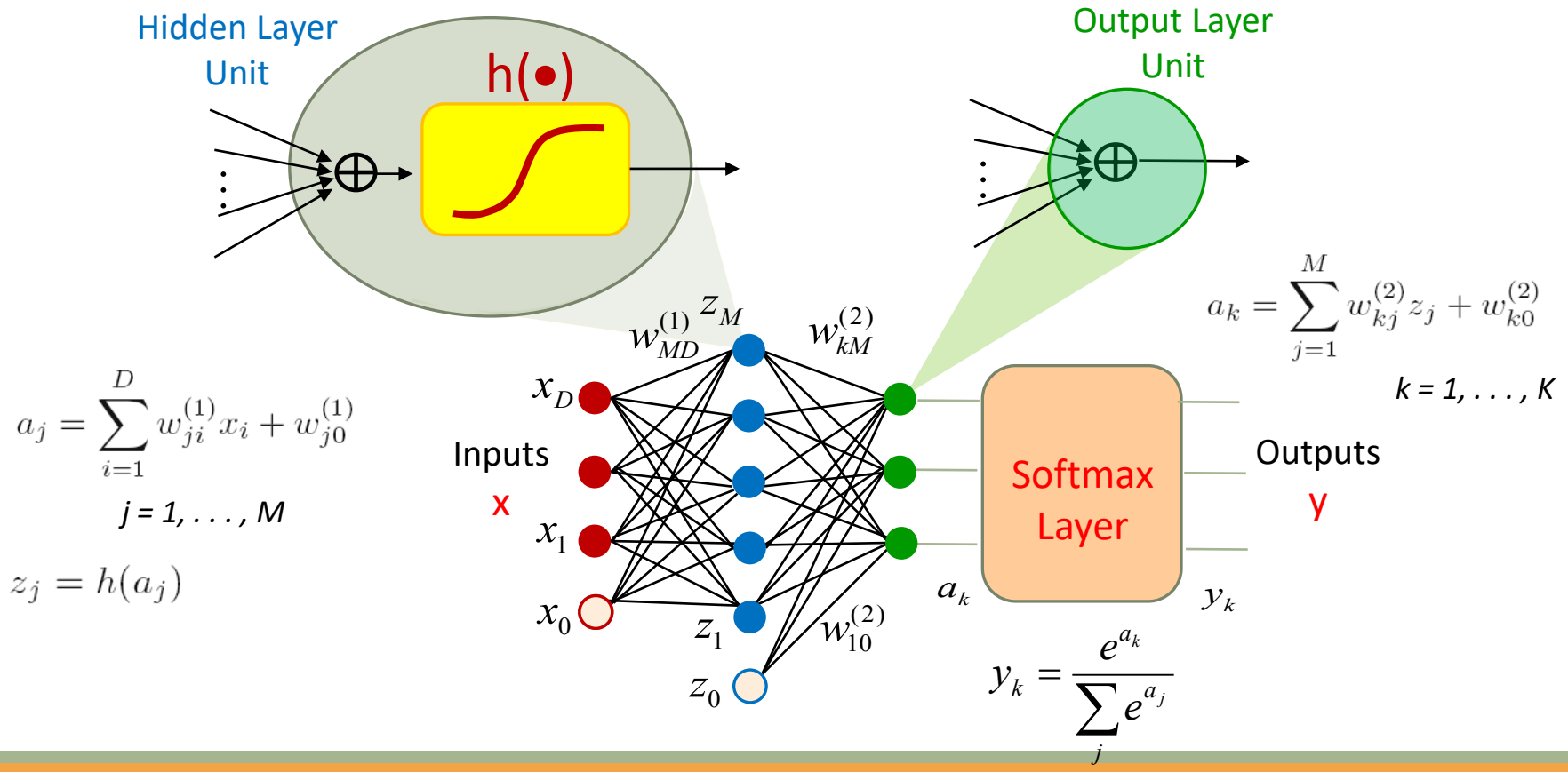
Feed-forward Neural Network (8/10)

Figure 5.4 Example of the solution of a simple two-class classification problem involving synthetic data using a neural network having two inputs, two hidden units with 'tanh' activation functions, and a single output having a logistic sigmoid activation function. The dashed blue lines show the $z = 0.5$ contours for each of the hidden units, and the red line shows the $y = 0.5$ decision surface for the network. For comparison, the green line denotes the optimal decision boundary computed from the distributions used to generate the data.



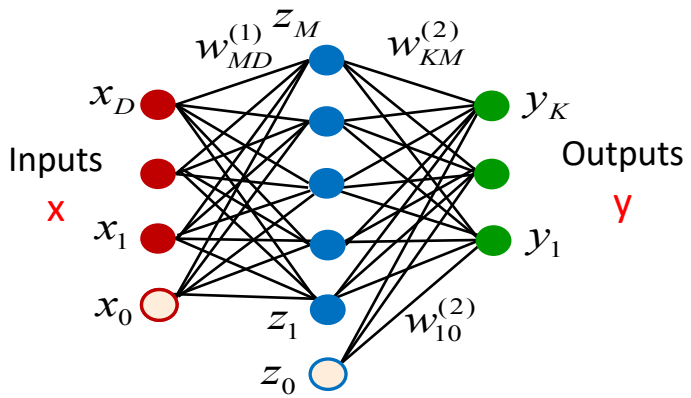
Feed-forward Neural Network (9/10)

Multi-Class Classification Model



Feed-forward Neural Network (10/10)

Two-layer Neural Network



$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

$$\sigma(a) = \begin{cases} a & \text{regression} \\ \dfrac{1}{1+e^{-a}} & \text{binary classification} \end{cases}$$

$$\sigma(a_k) = \frac{e^{a_k}}{\sum_j e^{a_j}} \qquad \text{multi-class classification}$$

Network Learning (1/7)

For regression problems (1-D Output)

Assume $p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}\left(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}\right)$

Given a data set of N independent, identically distributed observations $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, along with corresponding target values $\mathbf{t} = \{t_1, \ldots, t_N\}$, we have

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{w}, \beta).$$

Taking the negative logarithm, we obtain the error function

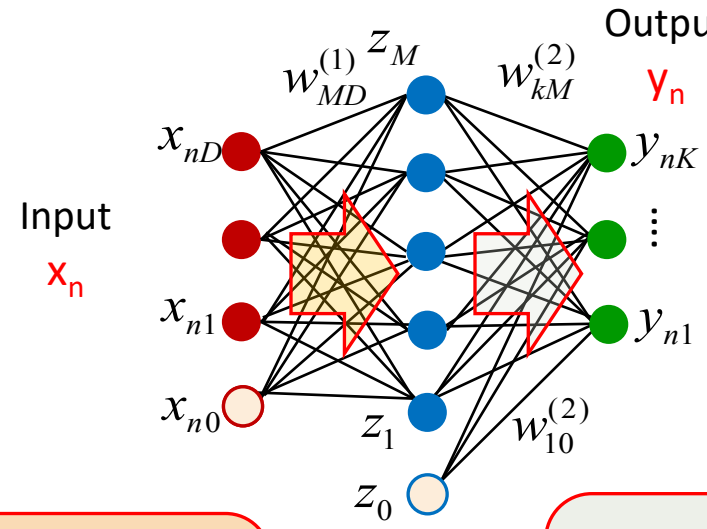
$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2\pi)$$

Maximizing the likelihood function is equivalent to minimizing the sum-of-squares error function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2$$ ***nonconvex***

Network Learning (2/7)

Regression (K-Dimension Output)



Expected Output

t_n

t_{nK}

t_{n1}

$$E_n(\mathbf{w}) = \frac{1}{2}\|\mathbf{y}_n - \mathbf{t}_n\|^2 = \frac{1}{2}\sum_{k=1}^{K}(y_{nk} - t_{nk})^2$$

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}, \quad j = 1,\ldots,M$$

$$z_j = h(a_j)$$

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}, \quad k = 1,\ldots,K$$

$$y_k = a_k$$

Forward Propagation

Network Learning (3/7)

For binary classification problems

$$y = \sigma(a) \equiv \frac{1}{1 + \exp(-a)}$$

$$p(t|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t \left\{1 - y(\mathbf{x}, \mathbf{w})\right\}^{1-t}$$

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

For multi-class classification problems

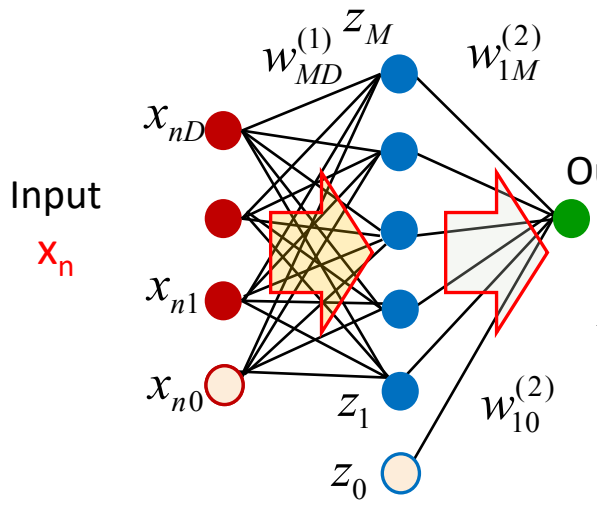
$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x}, \mathbf{w}))}$$

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w})$$

Network Learning (4/7)

Binary Classification

Forward Propagation



$$E_n(\mathbf{w}) = -\{t_n \ln y(\mathbf{x}_n, \mathbf{w}) + (1 - t_n)\ln(1 - y(\mathbf{x}_n, \mathbf{w}))\}$$

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}, \qquad j = 1,\ldots,M$$

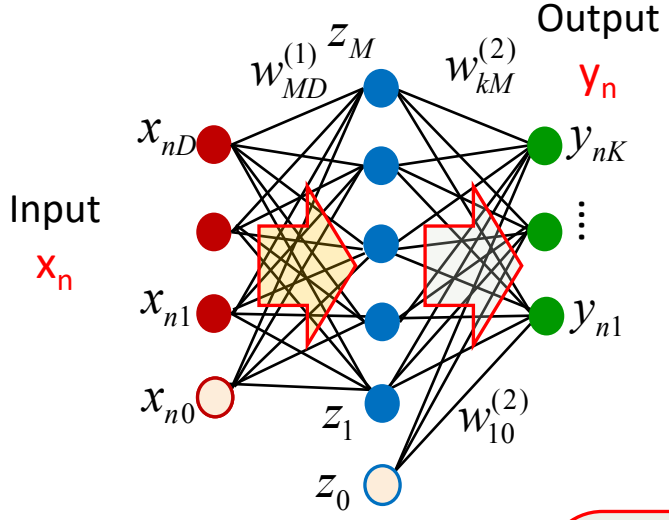
$$z_j = h(a_j)$$

$$a = \sum_{j=1}^{M} w_{1j}^{(2)} z_j + w_{10}^{(2)}, \quad k = 1,\ldots,K$$

$$y = \frac{1}{1 + e^{-a}}$$

Network Learning (5/7)

Multi-Class Classification



Expected Output

t_n

t_{nK} 1

t_{n1} 0

$$E_n(\mathbf{w}) = -\sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w})$$

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}, \quad j = 1,...,M$$

$$z_j = h(a_j)$$

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}, \quad k = 1,...,K$$

$$y_k = \frac{e^{a_k}}{\sum_j e^{a_j}}$$

Network Learning (6/7)

Parameter Optimization

Quadratic Approximation of the error function

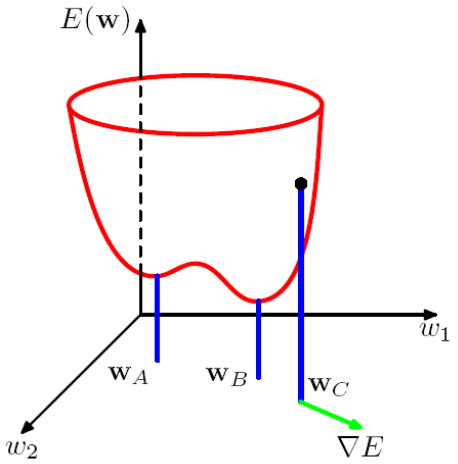
Consider the Taylor expansion of E(**w**) around some point $\hat{\mathbf{w}}$ in weight space

$$E(\mathbf{w}) \simeq E(\widehat{\mathbf{w}}) + (\mathbf{w} - \widehat{\mathbf{w}})^{\mathrm{T}}\mathbf{b} + \frac{1}{2}(\mathbf{w} - \widehat{\mathbf{w}})^{\mathrm{T}}\mathbf{H}(\mathbf{w} - \widehat{\mathbf{w}})$$

$$\mathbf{b} \equiv \nabla E|_{\mathbf{w}=\widehat{\mathbf{w}}}$$

$$\mathbf{H} = \nabla\nabla E$$

$$(\mathbf{H})_{ij} \equiv \left.\frac{\partial E}{\partial w_i \partial w_j}\right|_{\mathbf{w}=\widehat{\mathbf{w}}}$$



Network Learning (7/7)

Gradient Descent Optimization

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)}) \qquad E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

learning rate: $\eta > 0$

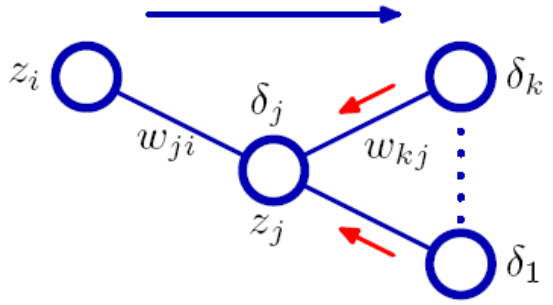
Sequential Gradient Descent (Stochastic Gradient Descent)

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)})$$

Error Back-Propagation (1/5)

An efficient technique for evaluating the gradient of an error function E(**w**) for a feed-forward neu

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)})$$



Error Backward-Propagation

Regression

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}, \quad j = 1,\ldots,M$$
$$z_j = h(a_j)$$

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}, \quad k = 1,\ldots,K$$
$$y_k = a_k$$

$$E_n(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{K}(y_{nk} - t_{nk})^2$$

Error function

Hidden layer $$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} = h'(a_j)\sum_k w_{kj}\delta_k$$

Output layer $$\delta_k \equiv \frac{\partial E_n}{\partial a_k} = y_{nk} - t_{nk}$$

Error Back-Propagation (2/5)

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}, \quad j = 1,\ldots,M$$

$$z_j = h(a_j)$$

Hidden layer

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}^{(1)}} = \delta_j z_i$$

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}, \quad k = 1,\ldots,K$$

$$y_k = a_k$$

Output layer

$$\frac{\partial E_n}{\partial w_{kj}^{(2)}} = \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

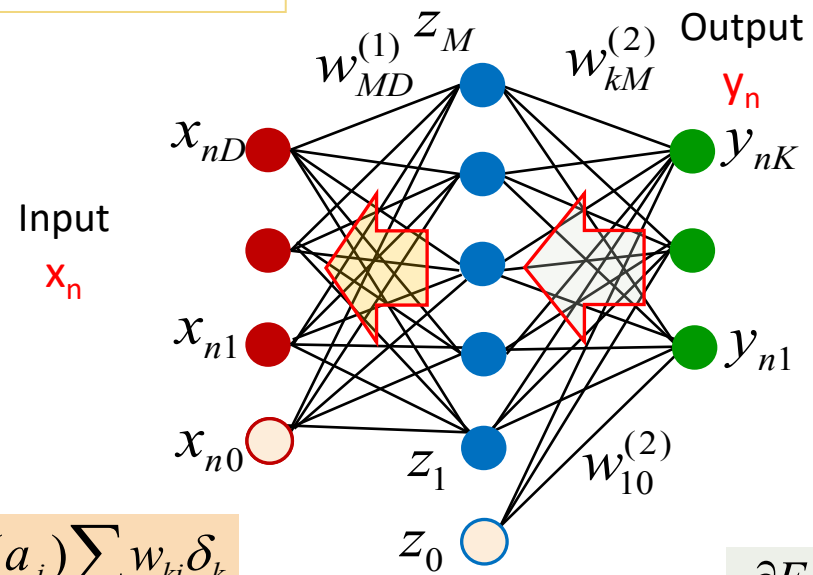
$$E_n(\mathbf{w}) = \frac{1}{2} \sum_{k=1}^{K} (y_{nk} - t_{nk})^2$$

Error function

$$\delta_k = y_{nk} - t_{nk}$$

Error Back-Propagation (3/5)

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)})$$



$$\delta_k = y_{nk} - t_{nk}$$

1

$$\frac{\partial E_n}{\partial w_{kj}^{(2)}} = \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

2

$$\delta_j = h'(a_j)\sum_k w_{kj}\delta_k$$

3

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}^{(1)}} = \delta_j z_i$$

Error Back-Propagation (4/5)

1

$$\delta_k \equiv \frac{\partial E_n}{\partial a_k} = \frac{\partial E_n}{\partial y_k}\frac{\partial y_k}{\partial a_k} = y_k - t_k$$

$$E_n(\mathbf{w}) = \begin{cases} \frac{1}{2}\sum_{k=1}^{K}(y_{nk} - t_{nk})^2 & \text{regression} \\ -\{t_n \ln y(\mathbf{x}_n, \mathbf{w}) + (1 - t_n)\ln(1 - y(\mathbf{x}_n, \mathbf{w}))\} & \text{binary classification} \\ -\sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w}) & \text{multi-calss classification} \end{cases}$$

$y_k = a_k$ regression

$y = \frac{1}{1 + e^{-a}}$ binary classification

$y_k = \frac{e^{a_k}}{\sum_j e^{a_j}}$ multi-class classification

2

$$\frac{\partial E_n}{\partial w_{kj}^{(2)}} = \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

3

$$\delta_j = h'(a_j)\sum_k w_{kj}\delta_k$$

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}^{(1)}} = \delta_j z_i$$

Error Back-Propagation (5/5)

Remarks:

1. Efficiency of backpropagation

   For sufficiently large W, a single evaluation of the error function (for a given input pattern) would require O(W) operations.

   In comparison, directly computing

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{E_n(w_{ji}+\epsilon) - E_n(w_{ji})}{\epsilon} + O(\epsilon)$$

   or

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{E_n(w_{ji}+\epsilon) - E_n(w_{ji}-\epsilon)}{2\epsilon} + O(\epsilon^2)$$

   would require $O(W^2)$ operations.

2. The technique of backpropagation can be applied to the calculation of other derivatives, like Jacobian matrix and Hessian matrix.

Issues for Training Neural Networks (1/2)

Theoretically, a neural network can be used to build any complicated mapping model if we have enough nodes and connections.

A deep network is more efficient than a shallow network.

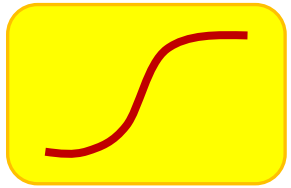
However, it is usually very hard to train a neural network with more than two hidden layers.

- Too many parameters
- Nonlinear optimization problem
- Vanishing gradient phenomenon

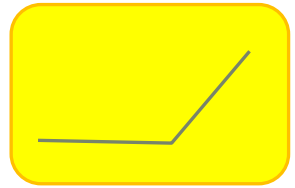
Issues for Training Neural Networks (2/2)

To train a neural network with several hidden layers, we can

- Apply weight sharing (Deep Convolutional Neural Network)
- Use a good initial guess (Restricted Boltzmann Machine)
- Use a huge amount of training data
- Use rectified linear function, instead of sigmoid function
- GPU acceleration
- Use some empirical training techniques
  - Dropout
  - Momentum
  - Weight Decay…



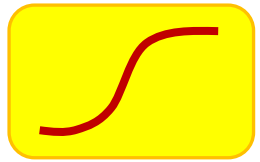
sigmoid



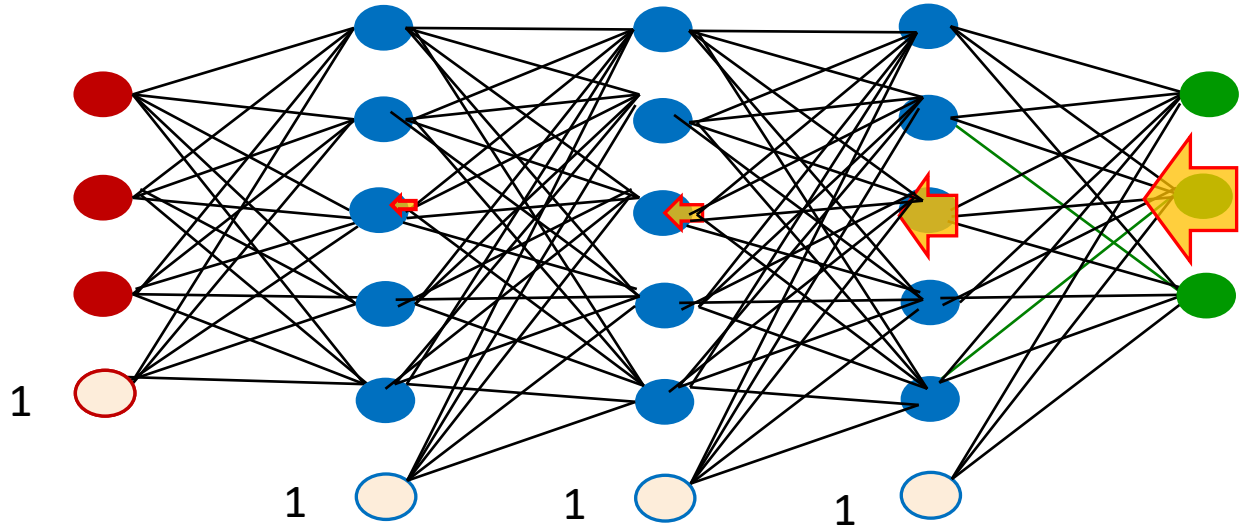
rectified linear function

Vanishing Gradients

$$\delta_j = h'(a_j)\sum_k w_{kj}\delta_k$$

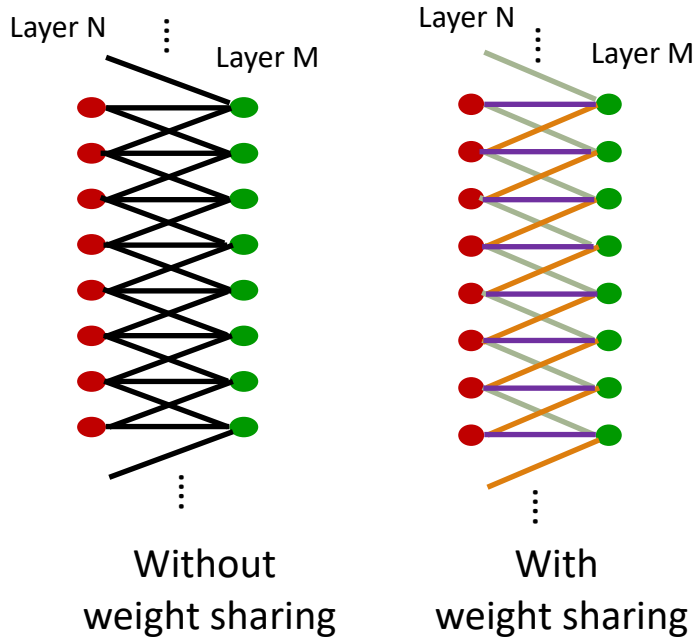


h(•)



Weight Sharing

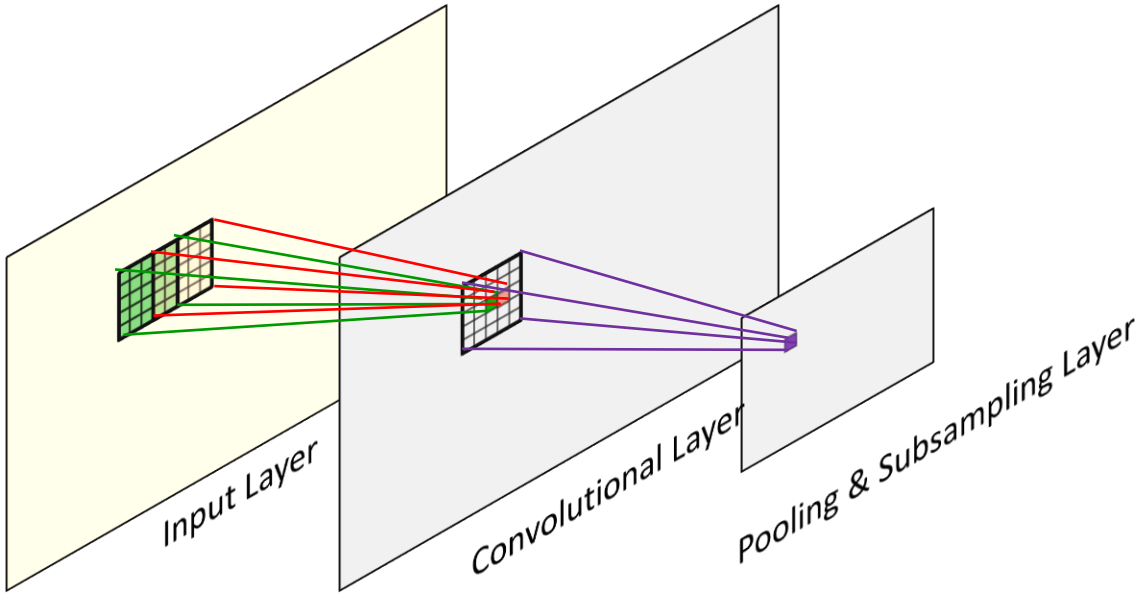
Assume each cell on Layer M connects to k cells on Layer N



Without weight sharing: $k \times N_M$ weights

With weight sharing: k weights

Convolutional Neural Networks (1/4)



Three major mechanisms:

1. local receptive fields
2. weight sharing
3. subsampling.

Convolutional Neural Networks (2/4)

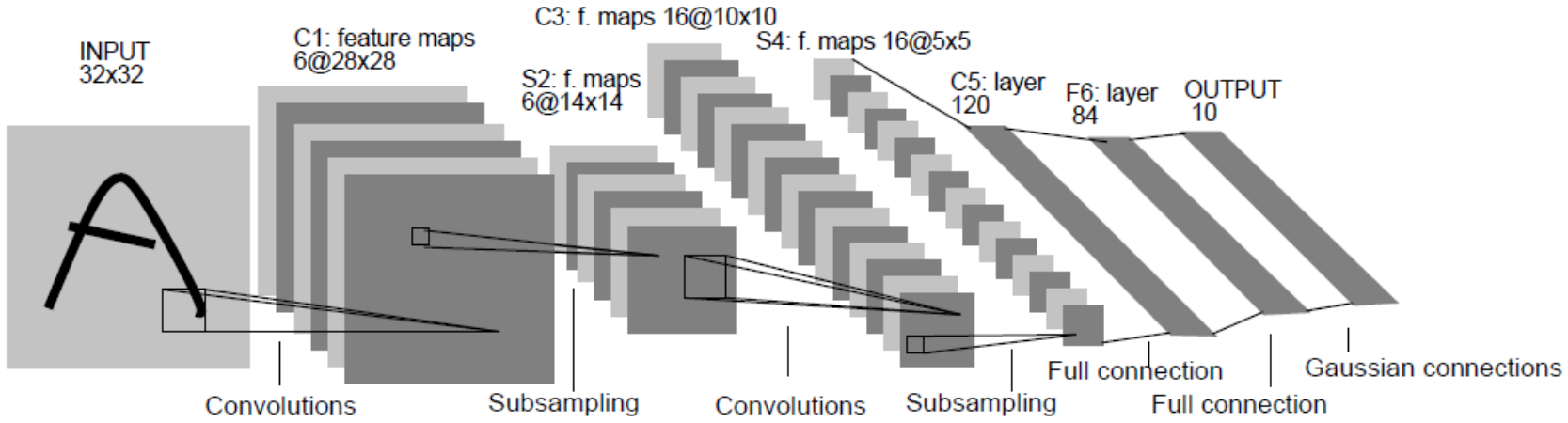
Units in the convolutional layer are organized into planes, named a *feature map*. Each unit in a feature map takes inputs from a small subregion of the image. All of the units in a feature map are constrained to share the same weight values.

Generally, there are multiple feature maps in the convolutional layer, each having its own set of weight and bias parameters.

For each feature map in the convolutional layer, there is a plane of units in the subsampling layer. Each unit in the subsampling layer takes inputs from a small receptive field in the corresponding feature map.

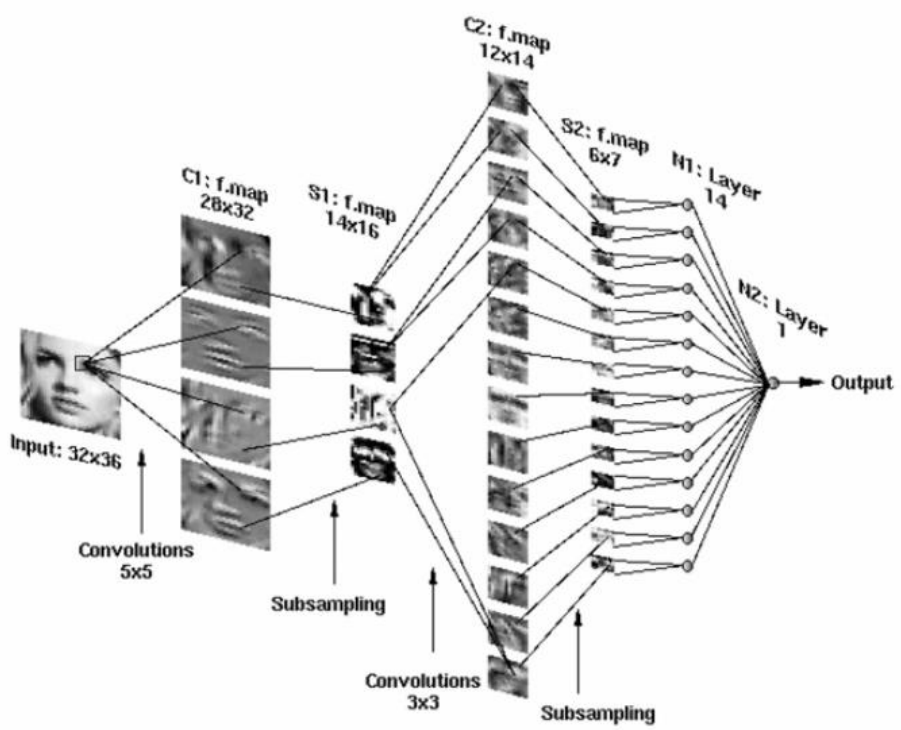
Convolutional Neural Networks (3/4)

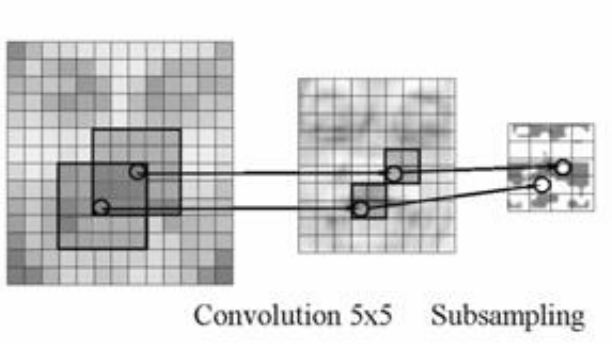
LeCun's Convolutional Neural Network



(Ref: Yann LeCun, "Gradient-Based Learning Applied to Document Recognition", Proceedings of the IEEE, 1998)

Convolutional Neural Networks (4/4)



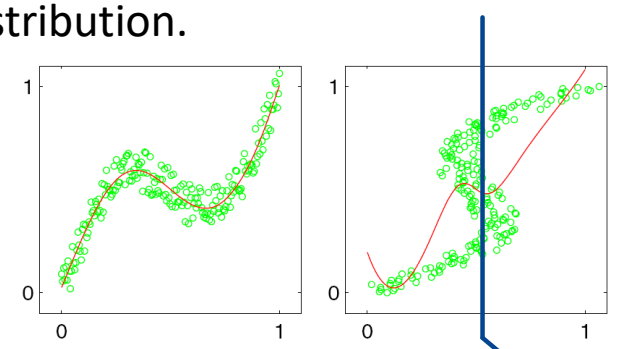


(Ref: Garcia & Delakis, Convolutional Face Finder: A Neural Architecture for Fast and Robust Face Detection, IEEE PAMI 2004.)

Mixture Density Networks (1/4)

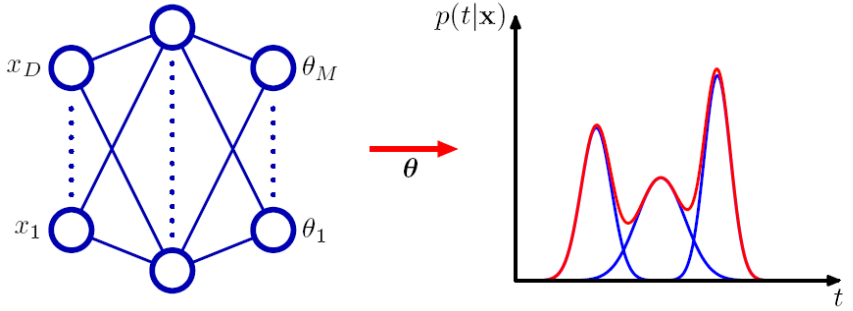
In practical machine learning problems, the conditional distribution p(t|x), often has a significantly non-Gaussian distribution.

Figure 5.19 On the left is the data set for a simple 'forward problem' in which the red curve shows the result of fitting a two-layer neural network by minimizing the sum-of-squares error function. The corresponding inverse problem, shown on the right, is obtained by exchanging the roles of x and t. Here the same network trained again by minimizing the sum-of-squares error function gives a very poor fit to the data due to the multimodality of the data set.



$$p(\mathbf{t}\mid\mathbf{x})=\sum_{k=1}^{K}\pi_k(\mathbf{x})N(\mathbf{t}\mid\mu_k(\mathbf{x}),\sigma_k^2(\mathbf{x})\mathbf{I})$$

heteroscedastic model



Output: parameters of the mixture model

Mixture Density Networks (2/4)

- ✓ To satisfy the constraints

$$\sum_{k=1}^{K} \pi_k(\mathbf{x}) = 1, \qquad 0 \leqslant \pi_k(\mathbf{x}) \leqslant 1$$

we define

$$\pi_k(\mathbf{x}) = \frac{\exp(a_k^{\pi})}{\sum_{l=1}^{K} \exp(a_l^{\pi})}$$

- ✓ To keep $\sigma_k^2(\mathbf{x}) \geqslant 0$,

we define

$$\sigma_k(\mathbf{x}) = \exp(a_k^{\sigma})$$

Mixture Density Networks (3/4)

$$\mu_{kj}(\mathbf{x}) = a_{kj}^{\mu}$$

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \ln\{\sum_{k=1}^{K} \pi_k(\mathbf{x}_n, \mathbf{w}) N(\mathbf{t}_n \mid \mu_k(\mathbf{x}_n, \mathbf{w}), \sigma_k^2(\mathbf{x}_n, \mathbf{w})\mathbf{I})\}$$

$$\frac{\partial E_n}{\partial a_k^{\pi}} = \pi_k - \gamma_{nk}$$

where $$\gamma_{nk} = \gamma_k(\mathbf{t}_n \mid \mathbf{x}_n) = \frac{\pi_k N_{nk}}{\sum_{l=1}^{K} \pi_l N_{nl}}$$

$$\frac{\partial E_n}{\partial a_{kl}^{\mu}} = \gamma_{nk}\{\frac{\mu_{kl} - t_l}{\sigma_k^2}\}$$

$$\frac{\partial E_n}{\partial a_{kl}^{\sigma}} = -\gamma_{nk}\{L - \frac{\|t_n - \mu_k\|^2}{\sigma_k^2}\}$$

Mixture Density Networks (4/4)

Figure 5.21 (a) Plot of the mixing coefficients $\pi_k(x)$ as a function of x for the three kernel functions in a mixture density network trained on the data shown in Figure 5.19. The model has three Gaussian components, and uses a two-layer multilayer perceptron with five 'tanh' sigmoidal units in the hidden layer, and nine outputs (corresponding to the 3 means and 3 variances of the Gaussian components and the 3 mixing coefficients). At both small and large values of x, where the conditional probability density of the target data is unimodal, only one of the kernels has a high value for its prior probability, while at intermediate values of x, where the conditional density is trimodal, the three mixing coefficients have comparable values. (b) Plots of the means $\mu_k(x)$ using the same colour coding as for the mixing coefficients. (c) Plot of the contours of the corresponding conditional probability density of the target data for the same mixture density network. (d) Plot of the approximate conditional mode, shown by the red points, of the conditional density.