EENG582: Artificial Neural Networks

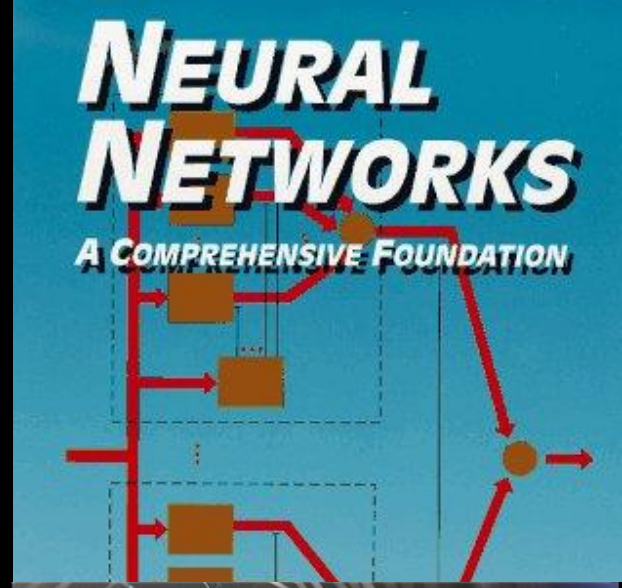
Neural Networks
A Comprehensive Foundation
by S. Haykin

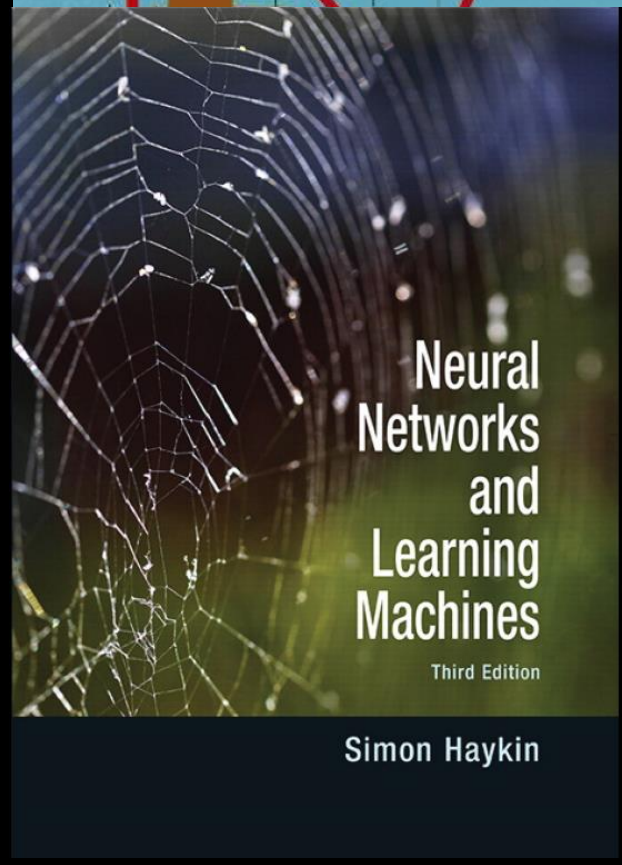
Sections 4.9 – 4.15

4 Multilayer Perceptrons

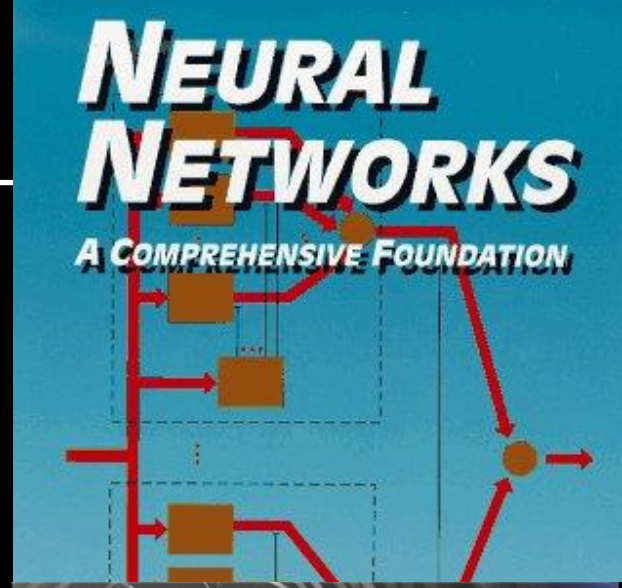
Prof. Dr. Hasan AMCA
Electrical and Electronic Engineering Department
(ee.emu.edu.tr)

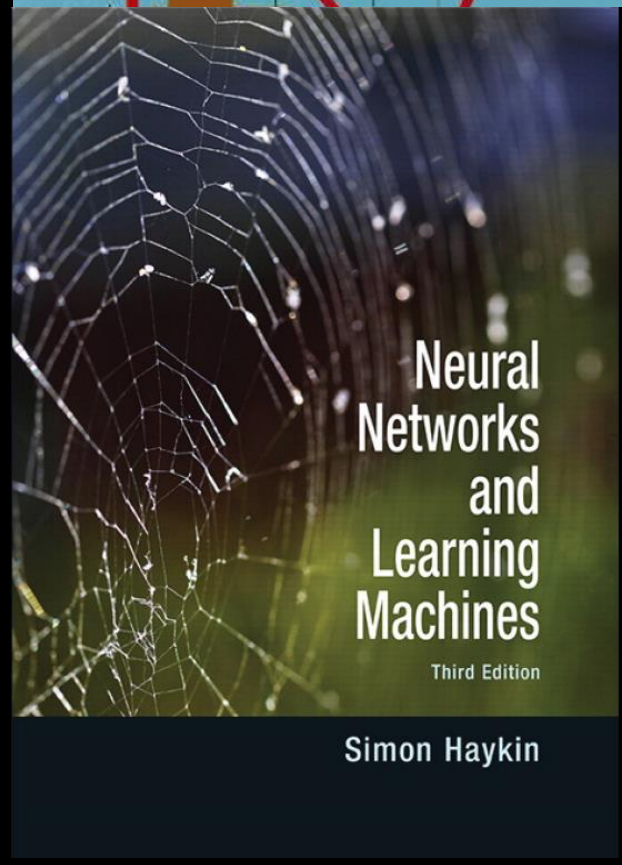
Eastern Mediterranean University
(emu.edu.tr)





4 Multi Layer Perceptrons





4.9 Feature Detection

- *Feature detectors* are neurons in the retina or brain that respond to specific attributes of a stimulus, movement, orientation etc.
- *Hidden neurons* play a critical role in the operation of a multilayer perceptron with back-propagation learning because they act as *feature detectors*.
- They do so by performing a nonlinear transformation on the input data into a new space called the *hidden space*, or *feature space.*
- Considering a multilayer perceptrons with a single nonlinear layer of m_1 hidden neurons and a linear layer of $m_2 = M$ output neurons.

4.9 Feature Detection

- Let the *synaptic weights* of the network be adjusted to minimize the mean square error between the desired output and the actual output of the network produced in response to a m_0-dimensional input vector (pattern), with the ensemble averaging performed over a total of N patterns.
- Let the nonlinear function of the pattern (vector) applied to the input layer of the network $z_j(n)$ denote the output of hidden neuron j due to the presentation of input pattern n.

4.9 Feature Detection

- The output of neuron k in the output layer is

$$y_k(n) = \sum_{j=0}^{m_j} w_{kj} z_j(n), \quad k = 1, 2, \ldots, M \quad \text{and} \quad n = 1, 2, \ldots, N \qquad (4.69)$$

- where w_{kj} represents the bias applied to neuron k.
- The cost function to be minimized is

$$\mathcal{E}_{av} = \frac{1}{2N} \sum_{n=1}^{N} \sum_{k=1}^{M} \big(d_k(n) - y_k(n)\big)^2 \qquad (4.70)$$

- Note the use of a batch mode of operation here.

4.9 Feature Detection

- Using (4.69) and (4.70) to reformulate the cost function $\mathcal{E}_{av}$ in matrix form:

$$\mathcal{E}_{av} = \frac{1}{2N}\left\|\widetilde{\mathbf{D}} - \mathbf{W}\tilde{\mathbf{Z}}\right\|^2 \qquad (4.71)$$

- where $\mathbf{W}$ is the M-by-m_1 matrix of synaptic weights relating to the output layer of the network.
- The matrix $\tilde{\mathbf{Z}}$ is the m_1-by-N matrix of hidden neuronal outputs (with their mean values subtracted off), which are produced by the individual N input patterns applied to the input layer of the network; that is,

$$\tilde{\mathbf{Z}} = \left\{\left(z_j(n) - \mu_{z_j}\right);\quad j = 1, 2, \ldots, m_1:\quad n = 1, 2, \ldots, N\right\}$$

- where μ_{z_j} is the mean value of $z_j(n)$.

4.9 Feature Detection

- The M-by-N matrix $\tilde{\mathbf{D}}$ is the target patterns (desired responses) presented to the output layer of the network;

$$\tilde{\mathbf{D}} = \left\{\left(d_k(n) - \mu_{d_k}\right);\quad k = 1, 2, \ldots, M:\quad n = 1, 2, \ldots, N\right\}$$

- where$\mu_{d_k}$is the mean value of $d_k(n)$.
- The minimization of $\mathcal{E}_{av}$ defined by (4.70) is recognized as a linear least-squares problem given by

$$\mathbf{W} = \tilde{\mathbf{D}}\tilde{\mathbf{Z}}^{+} \tag{4.72}$$

- where $\tilde{\mathbf{Z}}^{+}$ is the pseudo-inverse of matrix $\tilde{\mathbf{Z}}$.

4.9 Feature Detection

- The minimum value of $\mathcal{E}_{av}$ is given by (see Problem 4.7)

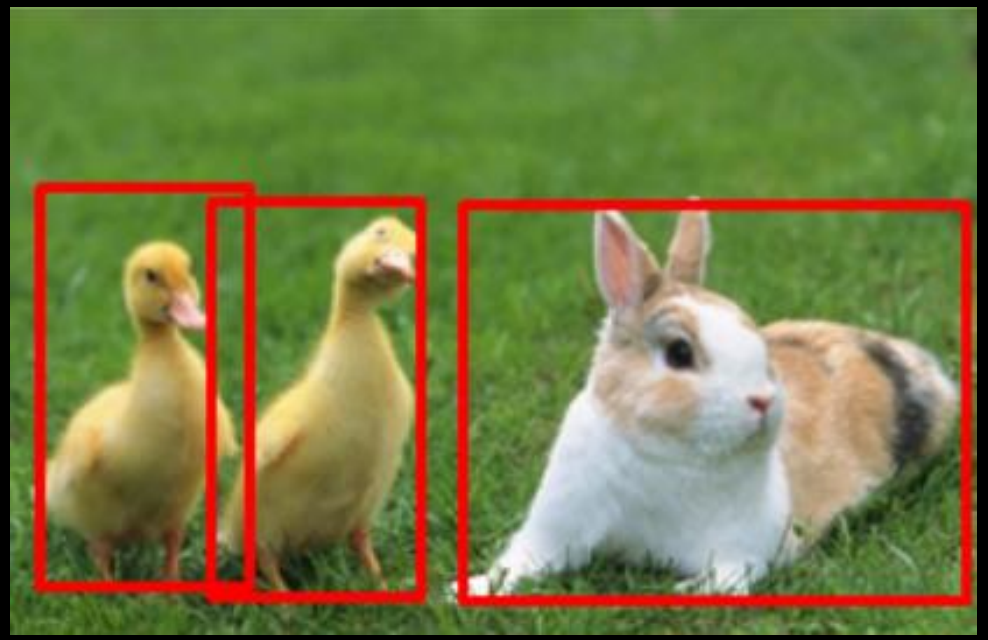
$$\mathcal{E}_{av,\,min} = \frac{1}{2N}\,\mathrm{tr}\left[\tilde{\mathbf{D}}\tilde{\mathbf{D}}^{T}\tilde{\mathbf{D}}\tilde{\mathbf{Z}}^{T}\left(\tilde{\mathbf{Z}}\tilde{\mathbf{Z}}^{T}\right)^{+}\tilde{\mathbf{Z}}\tilde{\mathbf{D}}^{T}\right] \tag{4.73}$$

- Where tr(.) is the trace operator and $\tilde{\mathbf{D}}$ represents the target pattern.
- Problem 4.7. Starting from the cost function defined in (4.70), derive the minimizing solution of (4.72) and the minimum value of the cost function defined in (4.73).

4.9 Feature Detection

- Tasks in Computer vision can be categorized as image classification, object detection or segmentation tasks.
- Feature detection is a low-level image processing operation. That is, it is usually performed as the first operation on an image and examines every pixel to see if there is a feature present at that pixel.

Classification



Object detection

Segmentation

Source: https://ataspinar.com/2017/12/04/using-convolutional-neural-networks-to-detect-features-in-sattelite-images/

4.9 Feature Detection

- Minimization of the cost function $\mathcal{E}_{av}$ with respect to the synaptic weights of the multilayer perceptron is equivalent to maximizing the discriminant

$$\mathcal{D} = tr[\mathbf{C}_b \mathbf{C}_t^+] \tag{4.74}$$

- Where $\mathbf{C}_b$ and $\mathbf{C}_t^+$ are defined as follows:
  - The m_1-by- m_1 matrix $\mathbf{C}_t$ is the total covariance matrix of the *hidden neuronal outputs* due to the presentations of N input patterns:

$$\mathbf{C}_t = \tilde{\mathbf{Z}}\tilde{\mathbf{Z}}^T \tag{4.75}$$

    and $\mathbf{C}_t^+$ is the pseudo inverse of $\mathbf{C}_t$.
  - The m_1-by- m_1 matrix $\mathbf{C}_b$ is defined by

$$\mathbf{C}_b = \tilde{\mathbf{Z}}\tilde{\mathbf{D}}^T\tilde{\mathbf{D}}\tilde{\mathbf{Z}}^T \tag{4.76}$$

- In a multilayer perceptron with more than one hidden layer, matrix $\tilde{\mathbf{Z}}$ refers to entire set of patterns in the space defined by final layer of hidden neurons.

4.9 Feature Detection

- If there are M class $\mathcal{C}_k$, k=1, 2, … , M with class N_k patterns in class $\mathcal{C}_k$ and

$$\sum_{k=1}^{M} N_k = N$$

- we may then expand the matrix $\mathbf{C}_b$ for this coding scheme in the form

$$\mathbf{C}_b = \sum_{k=1}^{M} N_k^2 (\boldsymbol{\mu}_{z,k} - \boldsymbol{\mu}_z)(\boldsymbol{\mu}_{z,k} - \boldsymbol{\mu}_z)^T \qquad (4.77)$$

Where the m_1-by-1 vector $\boldsymbol{\mu}_{z,k}$ is the mean value of the vector of the hidden neuronal outputs over all N_k patterns in class $\mathcal{C}_k$ and the vector $\boldsymbol{\mu}_z$ is the mean value of the vector of hidden neuronal outputs over all N input presentations.

Relation to Fisher's Linear Discriminant

- The discriminant function $\mathcal{D}$ defined in (4.74) is a linear transformation from a multidimensional problem to a one-dimensional problem $y = \mathbf{w}^{\mathrm{T}}\mathbf{x}$.
- *Fisher criterion* for discriminating between these two classes is defined by the *Rayleigh quotient.*

$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{C}_b \mathbf{w}}{\mathbf{w}^T \mathbf{C}_t \mathbf{w}}$$

Where the *between-class covariance matrix* $\mathbf{C}_b = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T$ and the *within-class covariance matrix*

$$\mathbf{C}_t = \sum_{n \in \mathcal{C}_1} (\mathbf{x}_n - \boldsymbol{\mu}_1)(\mathbf{x}_n - \boldsymbol{\mu}_1)^T + \sum_{n \in \mathcal{C}_2} (\mathbf{x}_n - \boldsymbol{\mu}_2)(\mathbf{x}_n - \boldsymbol{\mu}_2)^T$$

Relation to Fisher's Linear Discriminant

- The vector w that maximizes $J(\mathbf{w})$ must satisfy the condition

$$\boldsymbol{C}_b\boldsymbol{w} = \lambda \mathbf{C}_t \mathbf{w} \qquad (4.76)$$

- (4.76) is a generalized eigenvalue problem.
- Recognizing that the matrix product $\boldsymbol{C}_b\boldsymbol{w}$ is always in the direction of the difference vector $\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2$, we find that the solution for (4.76) is simply

$$\mathbf{w} = \mathbf{C}_t^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \qquad (4.77)$$

- which is referred to as Fisher's linear discriminant.

4.10 Back-propagation and Differentiation

- Back-propagation is a specific technique for implementing gradient descent in weight space for a multilayer feedforward network.
- The bask idea is to efficiently compute partial derivatives of an approximating function $F(\mathbf{w}, \mathbf{x})$ realized by the network with respect to all the elements of the adjustable weight vector $\mathbf{w}$ for a given value of input vector $\mathbf{x}$.
- Consider a multilayer perceptron depicted in Fig. 4.18.
- Let $w_{ji}^{(l)}$ denote synaptic weight from neuron i to neuron j in layer l = 0, 1, 2, ...
- We wish to evaluate derivatives of the function $F(\mathbf{w}, \mathbf{x})$ with respect to all the elements of the weight vector $\mathbf{w}$, for a specified input vector $\mathbf{x} = [x_1, x_2, \dots, x_{m_0}]^T$.

4.10 Back-propagation and Differentiation

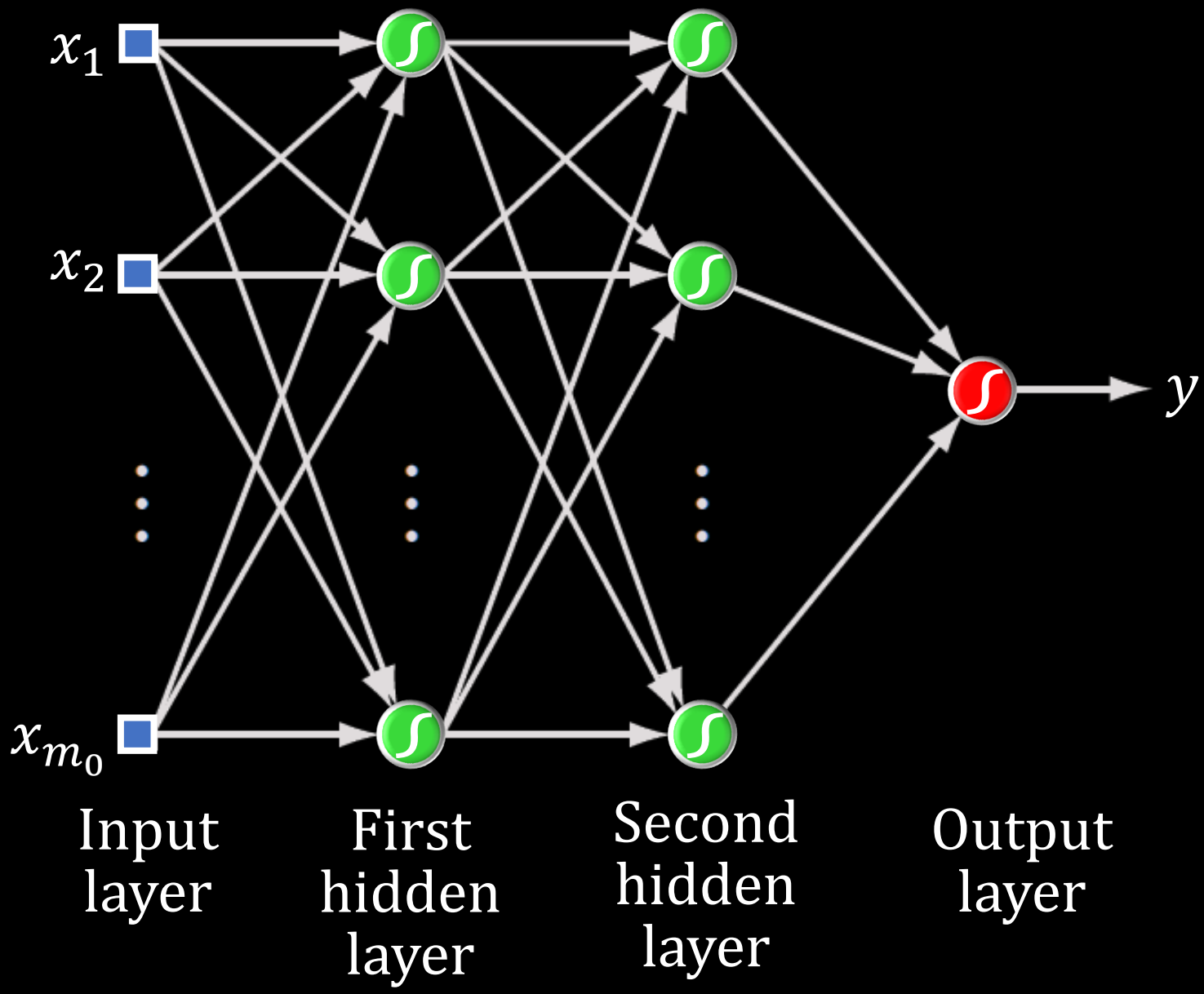


Fig. 4.18 Multilayer perceptron with two hidden layers and one output neuron.

4.10 Back-propagation and Differentiation

- The multilayer perceptron of Fig. 4.18 is parameterized by an *architecture* $\mathcal{A}$ (discrete parameter) and weight vector $\mathbf{w}$ (made up of continuous elements).
- Let $\mathcal{A}_j^{(l)}$ denote that part of the architecture extending from the input layer ($l = 0$) to node j in layer $l = 1, 2, 3$. Accordingly, we may write

$$F(\mathbf{w}, \mathbf{x}) = \varphi\left(\mathcal{A}_1^{(3)}\right) \tag{4.80}$$

- Thus, adapting (4.1), (4.2), (4.11) and (4.23), we obtain the following results:

$$\frac{\partial F(\mathbf{w},\mathbf{x})}{\partial w_{1k}^{(3)}} = \varphi'\left(\mathcal{A}_1^{(3)}\right)\varphi\left(\mathcal{A}_k^{(2)}\right) \tag{4.81}$$

$$\frac{\partial F(\mathbf{w},\mathbf{x})}{\partial w_{kj}^{(2)}} = \varphi'\left(\mathcal{A}_1^{(3)}\right)\varphi'\left(\mathcal{A}_k^{(2)}\right)\varphi\left(\mathcal{A}_j^{(1)}\right)w_{1k}^{(3)} \tag{4.82}$$

$$\frac{\partial F(\mathbf{w},\mathbf{x})}{\partial w_{ji}^{(1)}} = \varphi'\left(\mathcal{A}_1^{(3)}\right)\varphi'\left(\mathcal{A}_j^{(1)}\right)\mathbf{x}_i \sum_k \left[w_{1k}^{(3)}\varphi'\left(\mathcal{A}_k^{(2)}\right)w_{kj}^{(2)}\right] \tag{4.83}$$

4.10 Back-propagation and Differentiation

- Equations (4.81) - (4.83) shows sensitivity of the network function $F(\mathbf{w},\mathbf{x})$ to variations in the weight vector $\mathbf{w}$:
- The sensitivity of $F(\mathbf{w},\mathbf{x})$ to the variations in the elements of the weight vector $\boldsymbol{\omega}$ is formally defined by:

$$S_w^F = \frac{\partial F / F}{\partial \omega / \omega}, \qquad \omega \in \mathbf{w}$$

- This is the reason why we referred to the lower part of the signal-flow graph in Fig. 4.7 as a "sensitivity graph."

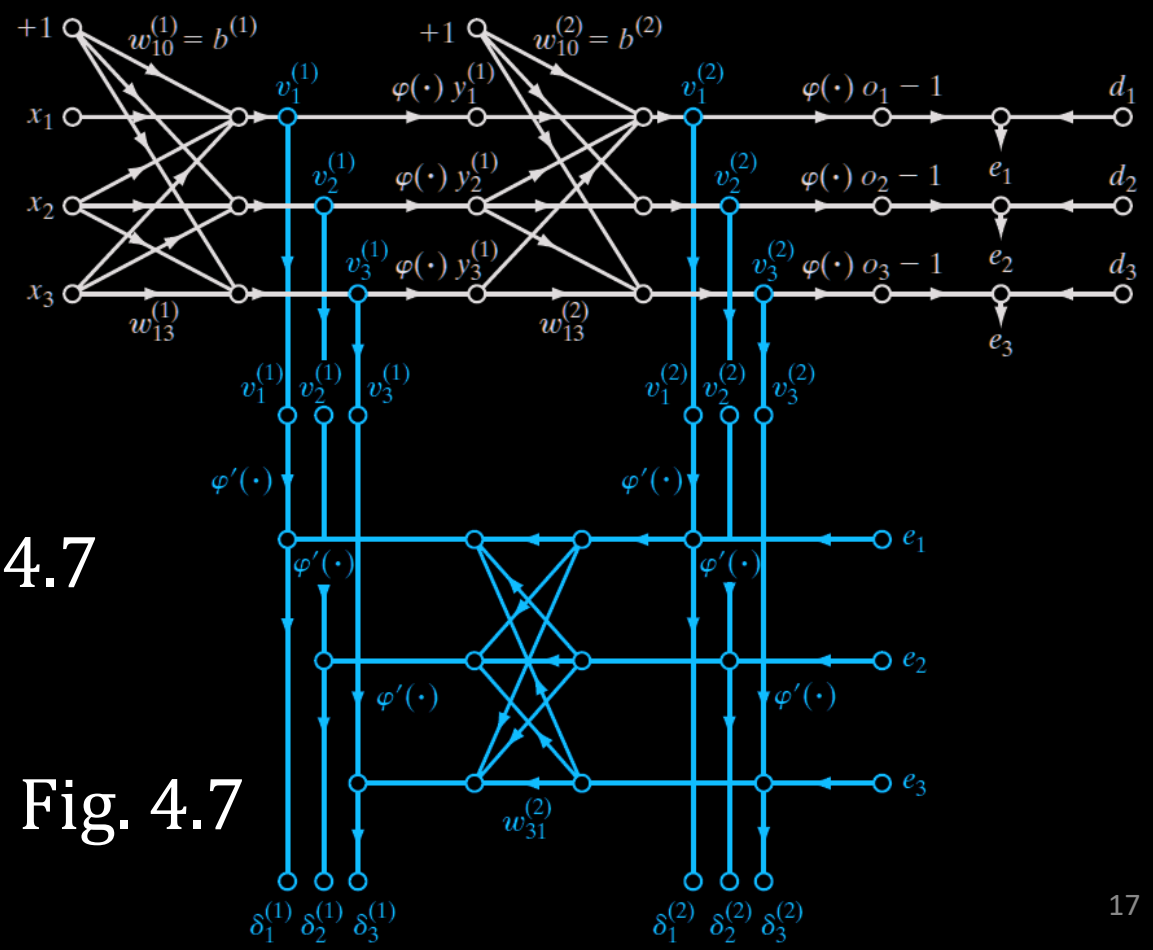


Fig. 4.7

Jacobian Matrix

- Let W denote the total number of free parameters (i.e., synaptic weights and biases) of a multilayer perceptron and N denote the total number of examples used to train the network.
- Using back-propagation we may compute a set of W partial derivatives of the approximating function $F[\mathbf{w}, \mathbf{x}(n)]$ with respect to the elements of the weight vector $\mathbf{w}$ for a specific example $\mathbf{x}(n)$ in the training set.
- Repeating these computations for $n = 1, 2, \ldots, N$, we end up with an N-by-W matrix of partial derivatives, called the Jacobian $\mathbf{J}$ of the multilayer perceptron evaluated at $\mathbf{x}(n)$.
- Each row of the Jacobian corresponds to a particular example in training set.
- Many neural network training problems are intrinsically ill-conditioned, leading to a Jacobian $\mathbf{J}$ that is almost rank deficient.

Jacobian Matrix

- The Jacobian is just the matrix of partial derivatives.
- You can compute it row-by-row. For example, the first equation is:

$$f_1(T, D, C) = \lambda - \mu T - \beta TC$$

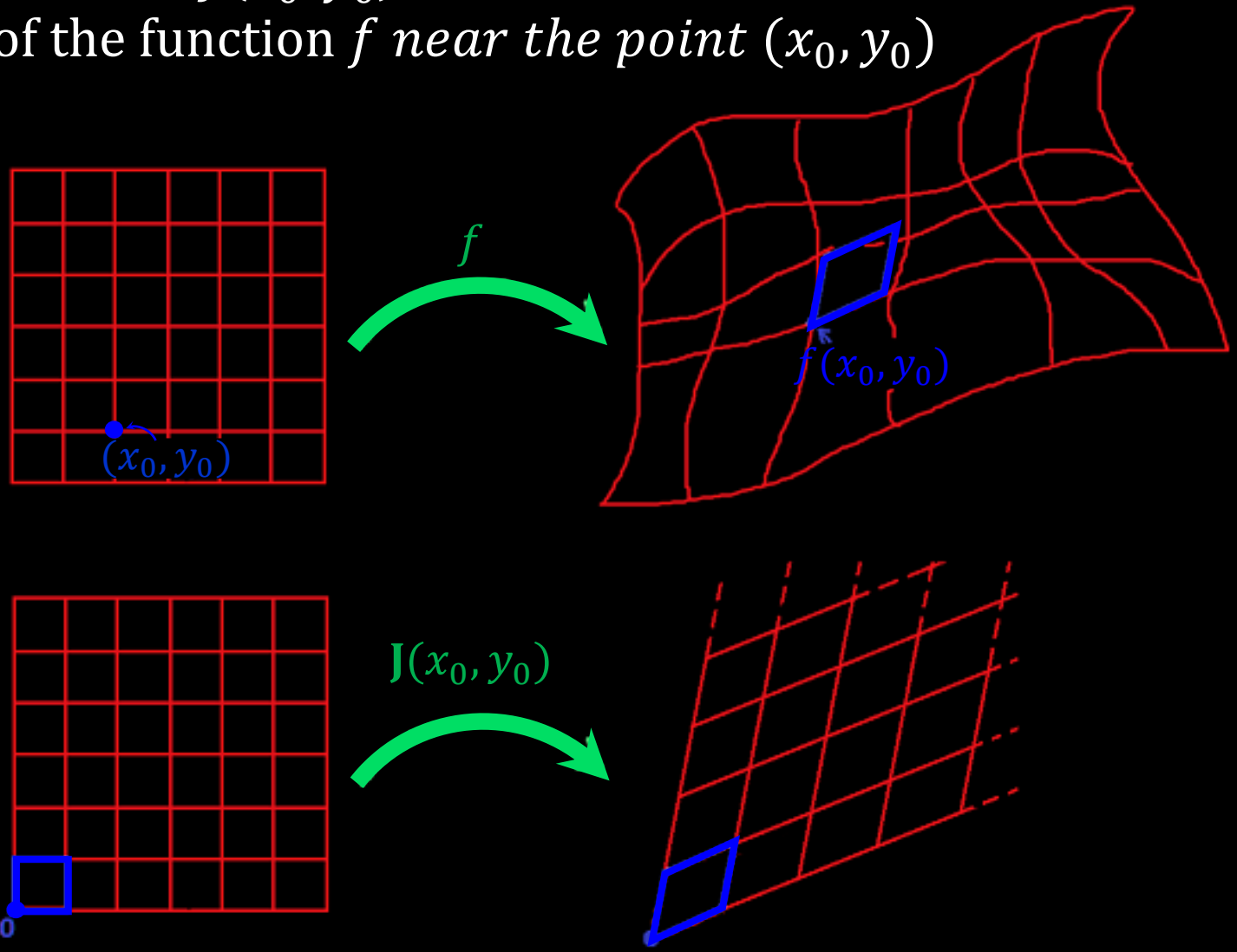
Which has partial derivatives of

$$\frac{\partial f_1}{\partial T} = -\mu - \beta C, \quad \frac{\partial f_1}{\partial D} = 0, \quad \frac{\partial f_1}{\partial C} = -\beta T$$

This gives us the first row of the Jacobian matrix:

$$J(T, D, C) = \begin{bmatrix} -\mu - \beta C & 0 & -\beta T \\ \text{second row of Jacobian} \\ \text{third row of Jacobian} \end{bmatrix}$$

Jacobian $J(x_0, y_0)$ is a local linearization of the function f *near the point* (x_0, y_0)



Nick Alger (https://math.stackexchange.com/users/3060/nick-alger), What do I do with these equations to create a Jacobian matrix?, URL (version: 2014-09-30): https://math.stackexchange.com/q/951950

Rank of a Matrix

- The rank of a matrix is the maximum number of its linearly independent column vectors (or row vectors). From this definition it is obvious that the rank of a matrix cannot exceed the number of its rows (or columns).
- The matrix has rank 2: First two columns are linearly independent, so rank is at least 2, but since the third is a linear combination of the first two (the second subtracted from the first), three columns are linearly dependent so rank must be < 3.

$$\begin{bmatrix} 1 & 0 & 1 \\ -2 & -3 & 1 \\ 3 & 3 & 0 \end{bmatrix}$$

Rank of a Matrix

- This matrix has rank 1: there are nonzero columns, so the rank is positive, but any pair of columns is linearly dependent.
- The transpose of **A** has rank 1.
  - Since the column vectors of **A** are the row vectors of the transpose of **A**,
  - the statement that the column rank of a matrix equals its row rank
  - is equivalent to the statement that the rank of a matrix is equal to the rank of its transpose, i.e., $rank(\mathbf{A}) = rank(\mathbf{A}^T)$.

$$A = \begin{bmatrix} 1 & 1 & 0 & 2 \\ -1 & -1 & 0 & -2 \end{bmatrix}$$

$$A^{\mathrm{T}} = \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ 0 & 0 \\ 2 & -2 \end{bmatrix}$$

4.11 Hessian Matrix

- The Hessian matrix of the cost function $\mathcal{E}_{av}(\mathbf{w})$, denoted by $\mathbf{H}$, is defined as the second derivative $\mathcal{E}_{av}(\mathbf{w})$ with respect to the weight vector $\mathbf{w}$, as

$$H = \frac{\partial \mathcal{E}_{av}(\mathbf{w})}{\partial \mathbf{w}^2} \tag{4.84}$$

- It is important to mention the following about the Hessian matrix:

1. The eigenvalues of the Hessian matrix have a profound influence on the Dynamics of back -propagation learning.
2. The inverse of the Hessian matrix provides a basis for deleting insignificant synaptic weights from a multilayer perceptrons.
3. The Hessian matrix is basic to the formulation of second-order optimization methods as an alternative to back-propagation learning.

4.11 Hessian Matrix

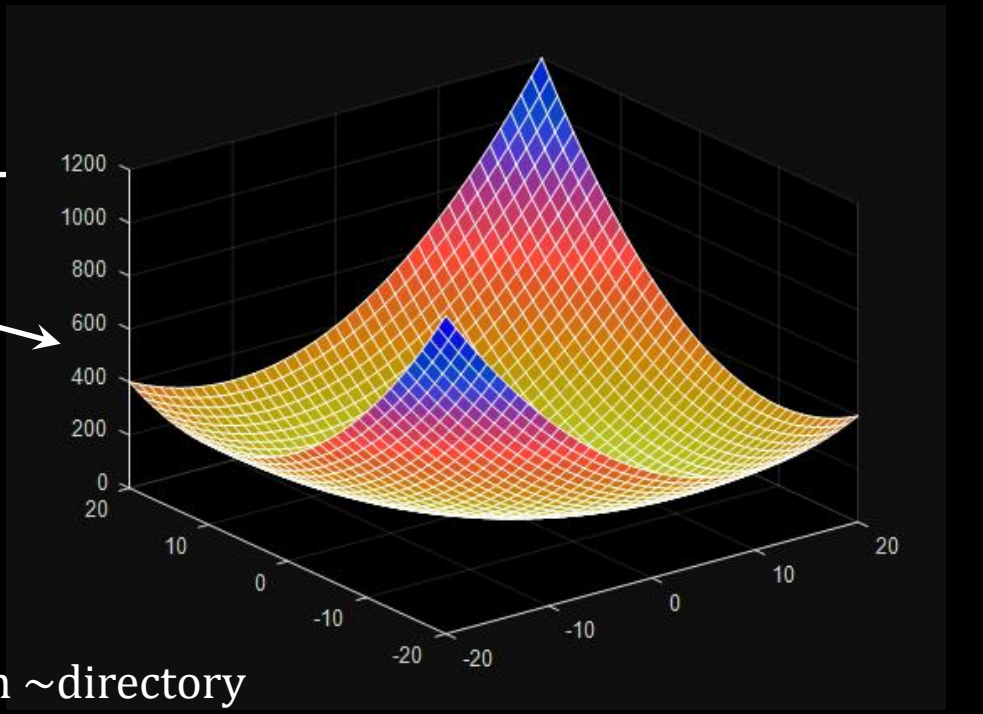
- Learning time of the LMS algorithm is sensitive to variations in the condition number $\lambda_{max}/\lambda_{min}$, where λ_{max} and λ_{min} are the largest and smallest nonzero eigenvalue of the Hessian.
- A similar result holds for the back-propagation algorithm, which is a generalization of the LMS algorithm.

4.11 Hessian Matrix

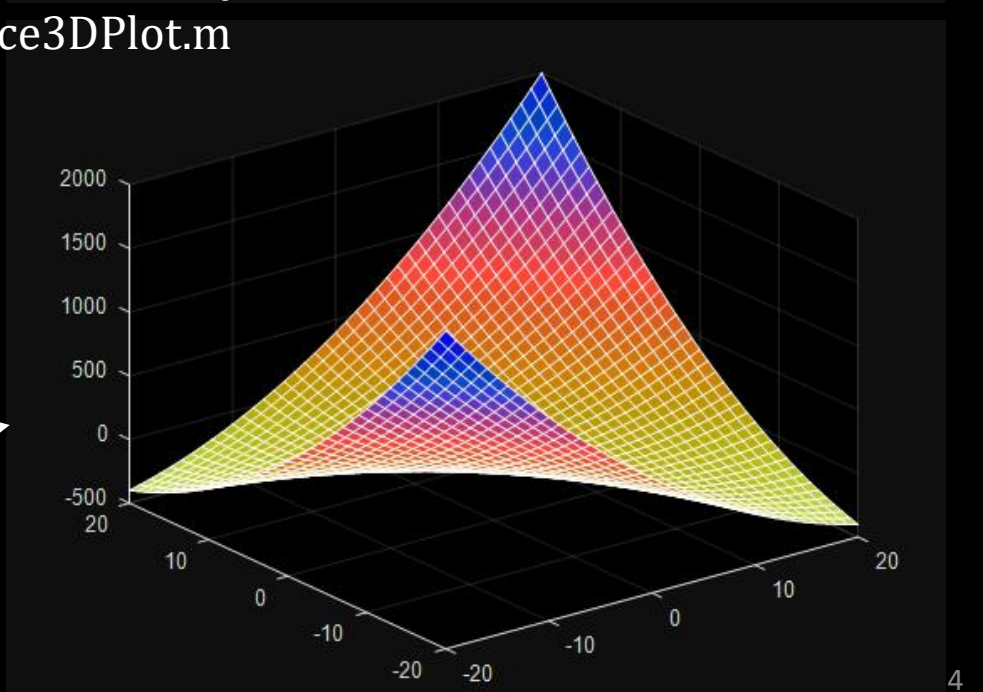
- Define the function $f = x_1^2 + x_1 x_2 + x_2^2$
- The derivative of f $\nabla f = \begin{Bmatrix} 2x_1 + x_2 \\ x_1 + 2x_2 \end{Bmatrix}$
- Stationary point $x_1 = x_2 = 0$
- Hessian matrix

$$H = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1 \partial x_2} \\ \dfrac{\partial^2 f}{\partial x_1 \partial x_2} & \dfrac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

- Eigenvalues: $\lambda_{1,2} = 1.3 \rightarrow$ minimum
- Change to $f = x_1^2 + 3x_1 x_2 + x_2^2$
- Eigenvalues: $\lambda_{1,2} = -1.5 \rightarrow$ saddle point



Use in ~directory
Surface3DPlot.m



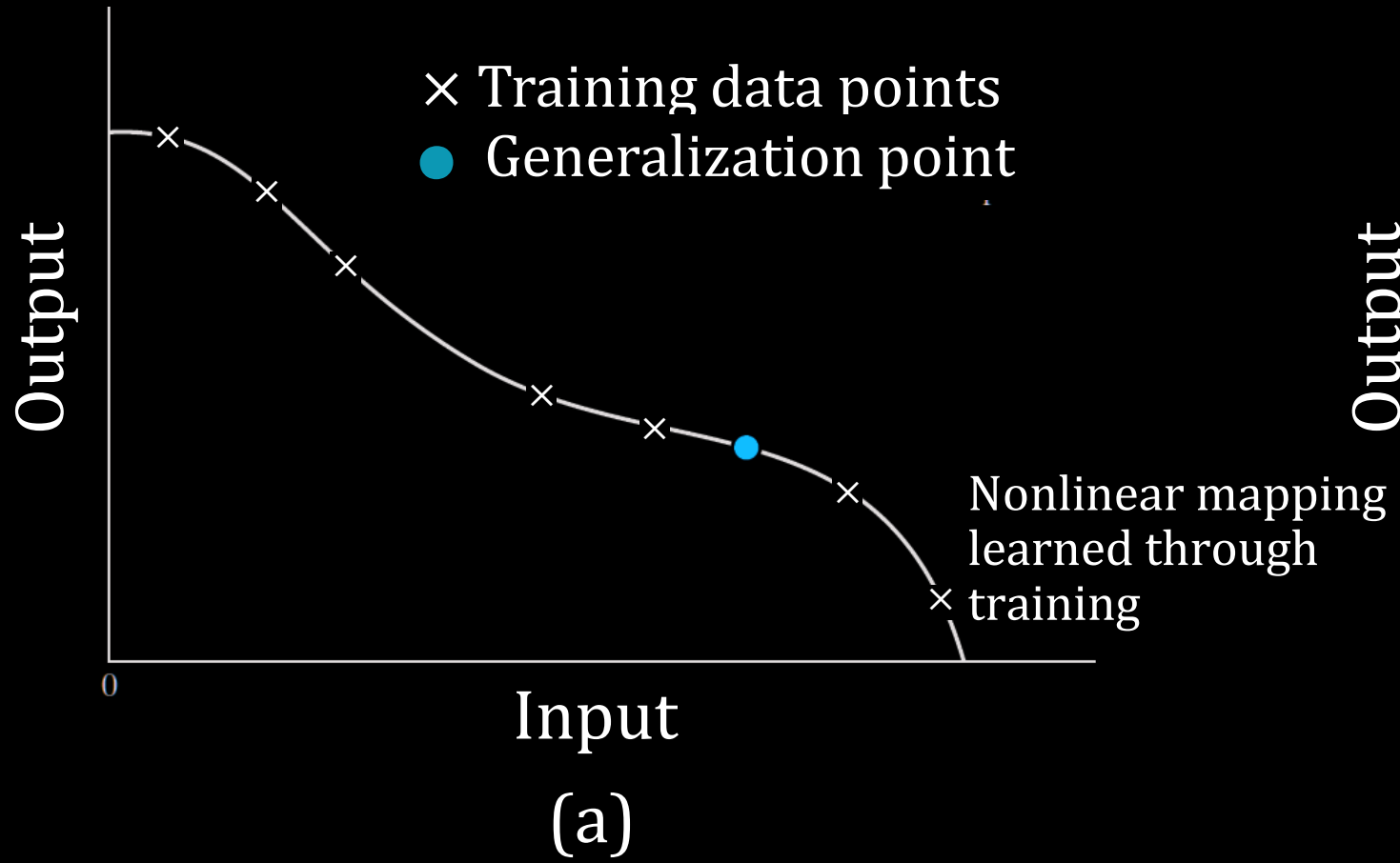
4.12 Generalization

- In back-propagation learning starts with a training sample and then compute the synaptic weights of a multilayer perceptrons by loading (encoding) as many of the training examples as possible into the network
  - So, the neural network will generalize.
- A network is said to generalize well when the input-output mapping computed by the network is correct (or nearly so) for test data never used in creating or training the network;
- Here it is assumed that the test data are drawn from the same population used to generate the training data.

4.12 Generalization

- Fig. 4.19a illustrates how generalization may occur in a hypothetical network.
- The nonlinear input-output mapping represented by the curve depicted in this figure is computed by the network as a result of learning the points labeled as "training data."
- The point marked on the curve as "generalization" is thus seen as the result of interpolation performed by the network.
- An example of how poor generalization due to memorization in a neural network may occur is illustrated in Fig. 4.19b.
- When, however, a neural network learns too many input-output examples, the network may end up memorizing the training data.
- Such a phenomenon is referred to as *overfitting* or *overtraining*.

4.12 Generalization



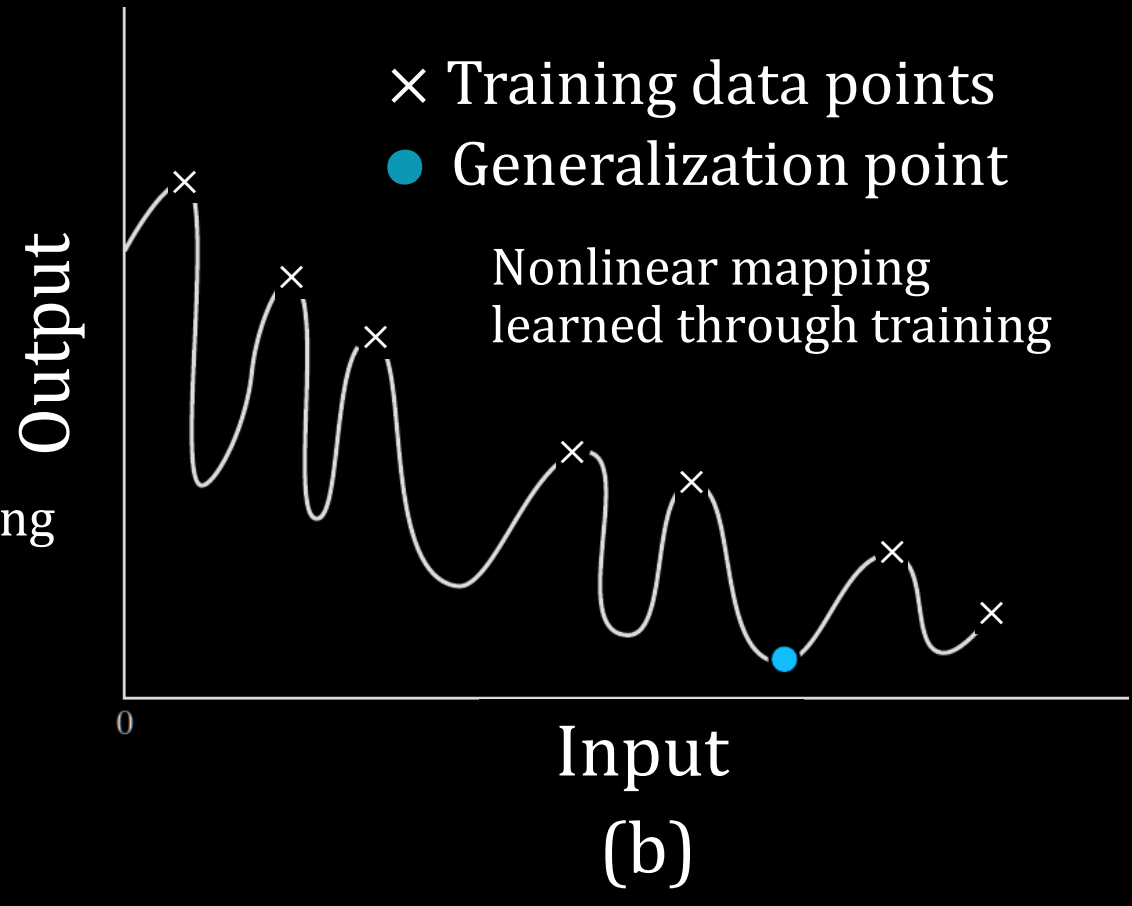


Fig. 4.19 (a) Properly fitted data (good generalization) (b) Overfitted data (poor generalization).

Sufficient Training Set Size for a Valid Generalization

- Generalization is influenced by three factors:
  1) The issue here is to determining the size of the training set needed and how representative it is of the environment of interest for achieving good generalization.
  2) The choice of best architecture of the neural network for achieving good generalization
  3) The physical complexity of the problem at hand.

Sufficient Training Set Size for a Valid Generalization

- For good generalization size of the training set, N, should satisfy the condition

$$N = O\left(\frac{W}{\epsilon}\right) \tag{4.85}$$

  - where W is the total number of free parameters (i.e., synaptic weights and biases) in the network,
  - ϵ denotes the fraction of classification errors permitted on test data (as in pattern classification) and
  - $O(.)$ denotes the order of quantity enclosed within.
- For an error of 10 percent #training examples N should be 10 times # free parameters in the network.

4.13 Approximations of Functions

- A multilayer perceptron trained with the back-propagation algorithm may be viewed as a practical vehicle for performing a nonlinear input-output mapping.
- The input-output relationship of a network with
  - m_0 input (source) nodes and $M = m_L$ neurons in the output layer,
  - defines a mapping from an m_0 dimensional Euclidean input space to an M-dimensional Euclidean output space,
  - which is continuously differentiable when the activation function is likewise.
- So, the following question arises:

What is the minimum number of hidden layers in a multilayer perceptron with an input-output mapping that provides an approximate realization of any continuous mapping?

Universal Approximation Theorem

- The universal approximation theorem for a nonlinear input-output mapping, is stated as:
- *Let $\varphi(.)$ be a nonconstant, bounded, and monotone-increasing continuous function. Let I_{m_0} denote the m_0-dimensional unit hypercube $[0,1]^{m_0}$.*
- *The space of continuous functions on I_{m_0} is denoted by $C(I_{m_0})$. Then, given any function $f \ni C(I_{m_0})$ and $\epsilon > 0$, there exists an integer M and sets of real constants α_i, b_i and w_{ij}, where $i = 1, \dots, m_1$ and $j = 1, \dots, m_0$ such that we may define*

$$F(x_1, \dots, x_{m_0}) = \sum_{i=1}^{m_1} \alpha_i \varphi\left(\sum_{i=1}^{m_1} w_{ij} x_j + b_i\right) \qquad (486)$$

as an approximate realization of the function $f(\cdot)$; that is,

$$\left|F(x_1, \dots, x_{m_0}) - f(x_1, \dots, x_{m_0})\right| < \epsilon$$

for all $x_1, \dots, x_{m_0}$. that lie in the input space.

Bounds on Approximation Errors

- The average of the norm of the frequency vector weighted by the Fourier magnitude distribution is used as a measure for the extent to which the function f oscillates.
- Let $\tilde{f}(\omega)$ denote the multidimensional Fourier transform of the function $f(\mathbf{x}), \mathbf{x} \in \mathbb{R}^{m_0}$, the m_0-by-1 vector $\boldsymbol{\omega}$ is the frequency vector.
- The function $f(x)$ is defined in terms of its Fourier transform $\tilde{f}(\boldsymbol{\omega})$ by the inverse formula:

$$f(x) = \int_{\mathbb{R}^{m_0}} \tilde{f}(\boldsymbol{\omega}) \exp(j\boldsymbol{\omega}^T \mathbf{x})\, d\boldsymbol{\omega} \qquad (4.87)$$

- where $j = \sqrt{-1}$.

Bounds on Approximation Errors

- The *first absolute moment* of the Fourier magnitude distribution of the function f is defined as:

$$C_f = \int_{\mathbb{R}^{m_0}} |\tilde{f}(\boldsymbol{\omega})| \times \|\boldsymbol{\omega}\|^{1/2} d\boldsymbol{\omega} \qquad (4.88)$$

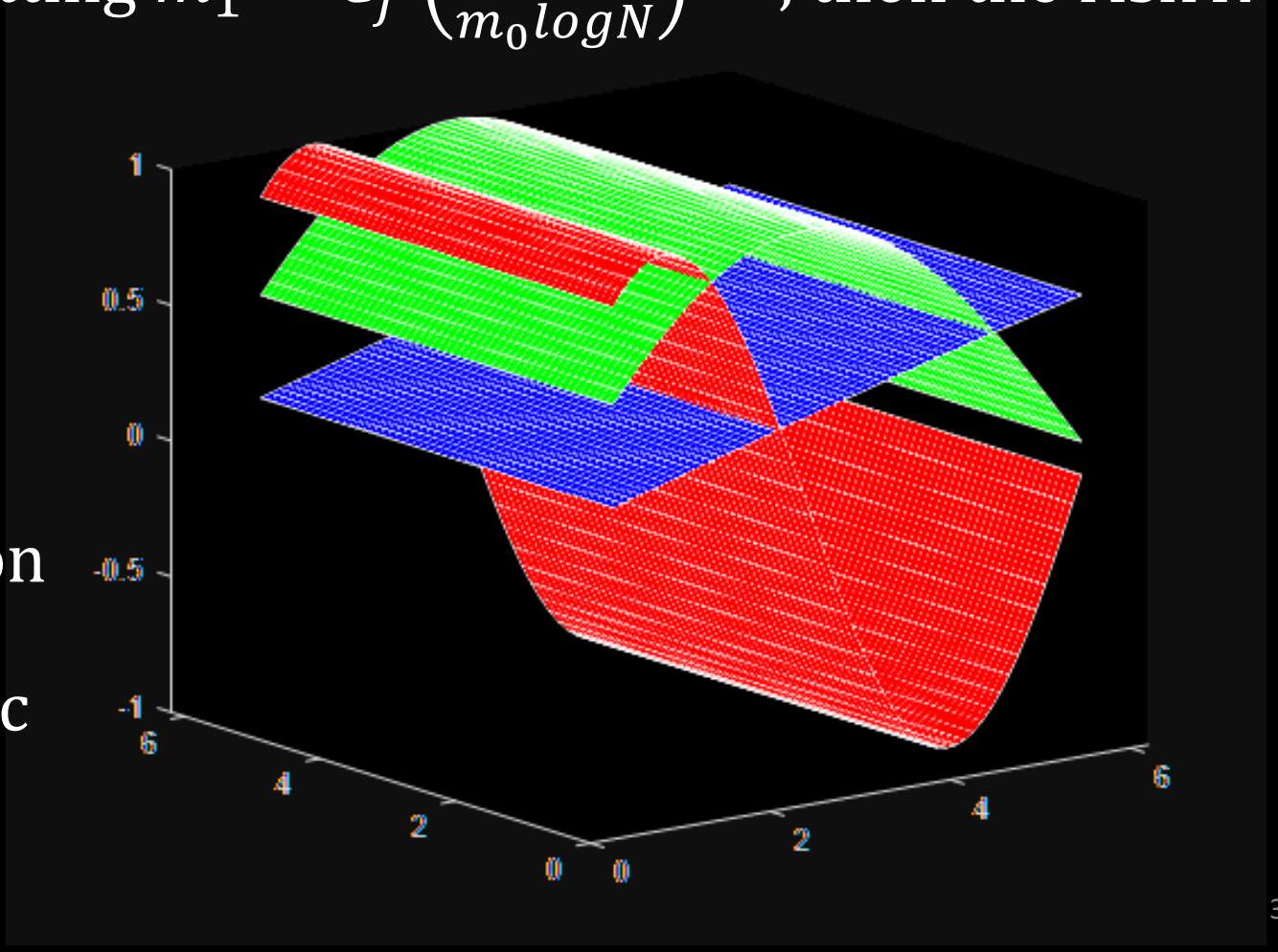
Where $\|\boldsymbol{\omega}\|$ is the Euclidean norm of $\boldsymbol{\omega}$ and $\left|\tilde{f}(\boldsymbol{\omega})\right|$ is the absolute value of $\tilde{f}(\boldsymbol{\omega})$.

- The first absolute moment C_f quantifies the smoothness of the function f.

Curse of Dimensionality

- Another interesting result that emerges from the-bounds described in (4.90) is that, when the size of the hidden layer is optimized (i.e., the risk R is minimized with respect to N) by setting $m_1 \cong C_f \left(\frac{N}{m_0 log N}\right)^{1/2}$, then the risk R is bounded by $O(C_f \sqrt{\left(m_0 \left(\frac{log N}{N}\right)\right)}$.

- A surprising aspect of this result is that in terms of the first-order behavior of the risk R, the rate of convergence expressed as a function of the training sample size N is of order $(1/N)^{1/2}$ (times a logarithmic factor).



Curse of Dimensionality

- let $\mathbf{x}$ denote an m_0-dimensional input vector, $\{(\mathbf{x}_i, d_i)\}, i = 1, 2, \ldots, N$, denote the training sample and the sampling density is proportional to N^{1/m_0}.
- Let a function $f(\mathbf{x})$ represent a surface lying in the m_0-dimensional input space, which passes near the data points $\{(\mathbf{x}_i, d_i)\}_{i=1}^{N}$.
- Now, if the function $f(\mathbf{x})$ is arbitrarily complex and completely unknown, we need dense sample (data) points to learn it well.
- Unfortunately, dense samples are hard to find in "high dimensions," hence the *curse of dimensionality*.

Practical Considerations

- The universal approximation theorem assumes that the continuous function to be approximated is given and that a hidden layer of unlimited size is available for the approximation.
- In multilayer perceptrons using a single hidden layer, the neurons tend to interact with each other globally. This interaction makes it difficult to improve the approximation.
- On the other hand, with two hidden layers the approximation (curve-fitting) process becomes more manageable (Funahashi, 1989; Chester, 1990) since…

1. *Local features are extracted in the first hidden layer*
2. *Global features are extracted in the first hidden layer*

4.14 Cross-Validation

- The essence of *back-propagation learning* is to encode an input-output mapping (represented by a set of labeled examples) into the synaptic weights and thresholds of a multilayer perceptron.
- The hope is that the network becomes well trained so that it learns enough about the past to generalize to the future.
- The initial step is to randomly partition the available data set into a training set and a test set. The training set is further partitioned into two disjoint subsets:
  - *Estimation subset*, used to select the model.
  - *Validation subset*, used to test or validate the model.
- The use of cross-validation is appealing to determine the multilayer perceptron with the best number of hidden neurons, and when it is best to stop training.

4.14 Cross-Validation: Model Selection

- Consider then a nested structure of Boolean function classes denoted by

$$\mathcal{F}_1 \subset \mathcal{F}_2 \subset \ \ldots \subset \mathcal{F}_n \qquad \mathcal{F}_k = \{F_k\} = \{F(\mathbf{x}, \mathbf{w}); \mathbf{w} \in \mathcal{W}_k\}, k = 1, 2, \ldots, n \quad (4.91)$$

- In words, the kth function class $\mathcal{F}_k$ encompasses a family of multilayer perceptrons with similar architecture and weight vectors $\mathbf{w}$ drawn from a multidimensional weight space $\mathcal{W}_k$.
- The model selection problem is essentially choosing the multilayer perceptron with the best value of W, the number of free parameters (i.e., synaptic weights and biases).
- If the scalar desired response for an input vector $\mathbf{x}$ is $d = \{0, 1\}$, the generalization error will be defined as
- $\epsilon_g(F) = P(F(\mathbf{x}) \neq d)$ for $x \in \mathcal{H}$ (the input space) for the training set of labelled examples $\mathcal{T} = \{(\mathbf{x}_i, d_i)\}_{i=1}^{N}$.

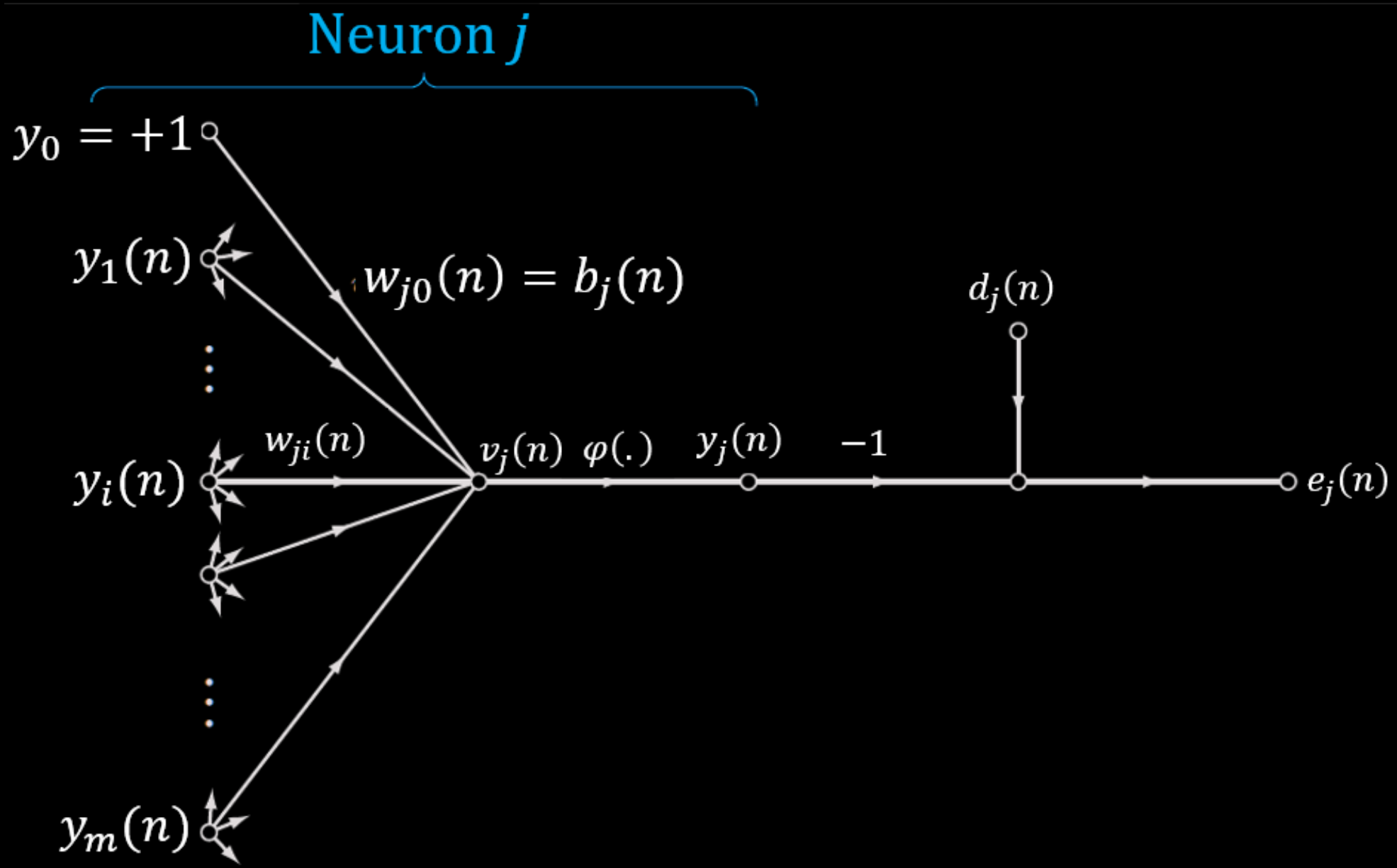
4.14 Cross-Validation: Model Selection

- The objective of the hypothesis $F(\mathbf{x},\mathbf{w})$ is to minimize the generalization error $\epsilon_g(F)$ for an input particular test set.
- Let a parameter $0 < r \leq 1$ determine the split of the training data set $\mathcal{T}$ between the estimation subset ($(1\text{-}r)N$ examples) and validation subsets (rN) examples).
- The use of cross-validation results in the choice

$$\mathcal{F}_{cv} = \min_{k=1,2,\ldots,v} \{e_t''(\mathcal{F}_k)\} \qquad (4.92)$$

- where v corresponds to $W_v \leq W_{max}\big((1-r)N\big)$ and $e_t''(\mathcal{F}_k)$ is the classification error produced by hypothesis $\mathcal{F}_k$ when it is tested on the validation subset $\mathcal{T}''$consisting of rN examples.
- Where $\mathcal{W}_v$ is the multidimensional weight space corresponding to signal v and local field $v(n)$ produced at the input of the activation function associated with neuron.

4.14 Cross-Validation: Model Selection



4.14 Cross-Validation: Model Selection

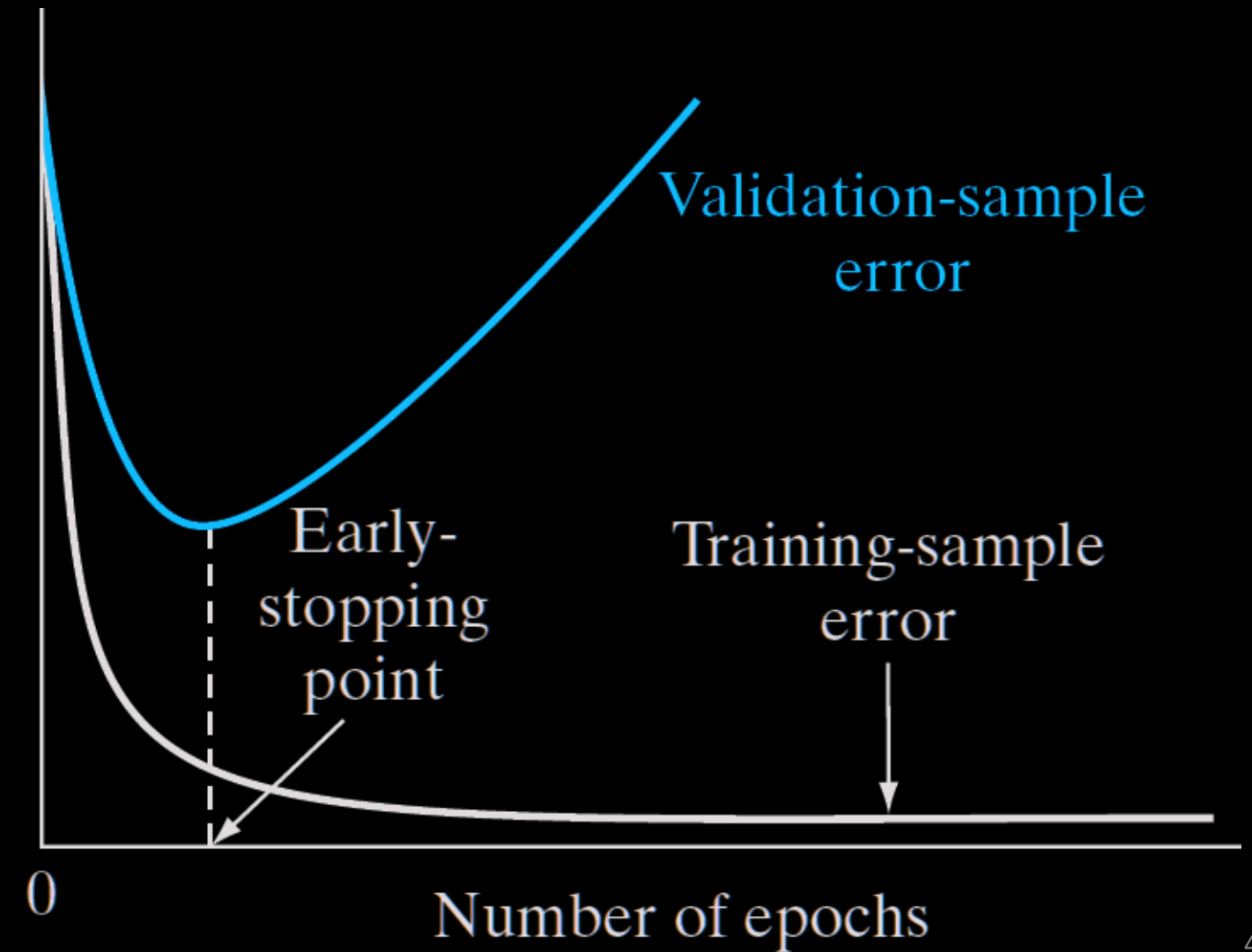
- To specify the parameter r that determines the split of the training set $\mathcal{T}$ between the estimation subset $\mathcal{T}'$ and validation subset $\mathcal{T}''$, we have
  - When the complexity of the target function is small compared to the sample size N, the performance of cross-validation is relatively insensitive to the choice of r.
  - As the complexity increases, the choice of optimum r has a more pronounced effect on cross-validation performance.
  - A single fixed value of r works nearly optimally for a wide range of target function complexity.
    - Ex. A suitable value for r =0.2 where 80% of training set $\mathcal{T}$ is assigned to estimation, 20% assigned to the validation subset.

4.14 Cross-Validation: Early Stopping Method of Training

- With good generalization, it is very difficult to figure out when it is best to stop training if we were to look at the learning curve for training all by itself.
- It is possible for the network to end up overfitting the training data if the training session is not stopped at the right point.
- The periodic estimation-followed-by-validation process proceeds as follows:
  - After a period of estimation (training), the synaptic weights and bias levels of the multilayer perceptron are all fixed and the network is operated in its forward mode. The validation error is thus measured for each example in the validation subset.
  - When the validation phase is completed, the estimation (training) is resumed for another period, and the process is repeated.

4.14 Cross-Validation: Early Stopping Method of Training

Fig. 4.20 Illustration of the early-stopping rule based on cross-validation



The conceptualized forms of two learning curves shown in Figure 4.20 does not do as well on the validation subset as it does on the estimation subset.

4.14 Cross-Validation: Early Stopping Method of Training

- Two modes of behavior are identified depending on the size of training set:
  - ***Non-Asymptotic Mode***, for which $N < W$, where N is the size of the training set and W is the number of free parameters in the network.

    For this mode of behavior, when the complete set of examples is used for training, the training session is not stopped.

    Hence, overfitting may occur when $N < 30W$, and there is practical merit in the use of cross-validation to stop training. The optimum value of parameter r_{opt} is defined by

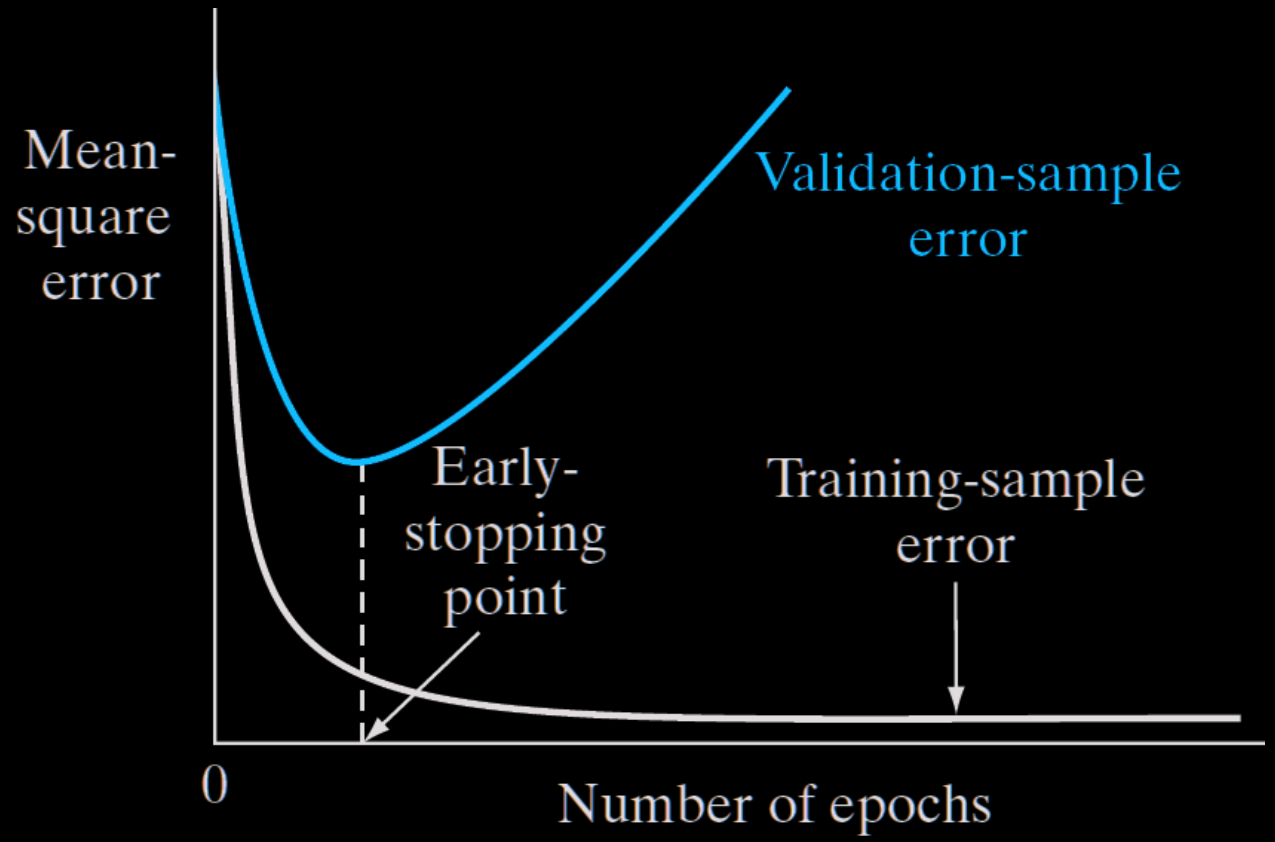
$$r_{opt} = 1 - \frac{\sqrt{2W-1}-1}{2(W-1)} \rightarrow r_{opt} = 1 - \frac{1}{\sqrt{2W}}, \text{ for large } W$$

    For $W = 100$, $r_{opt} = 0.07 \Rightarrow$ 93% of training data allocated to estimation subset, 7% allocated to validation subset.

4.14 Cross-Validation: Early Stopping Method of Training

- ***Asymptotic Mode***, for which $N > 30W$.

  For this mode of behavior, exhaustive learning is satisfactory when the size of the training sample is large compared to the number of network parameters.



4.14 Cross-Validation: Variants of Cross-Validation

- The approach to cross-validation described is referred to as ***hold out method***.
- There are other variants of cross-validation that find their own uses in practice, particularly **when there is a scarcity of labeled examples**.
  - we may use multifold cross-validation by dividing the available set of N examples into K subsets, $K > 1$; this assumes that K is divisible into N.
  - The model is trained on all the subsets except for one, and the validation error is measured by testing it on the subset left out.
  - This procedure is repeated for a total of K trials, each time using a different subset for validation, as illustrated in Fig. 4.21 for K = 4.

4.14 Cross-Validation:
Variants of Cross-Validation

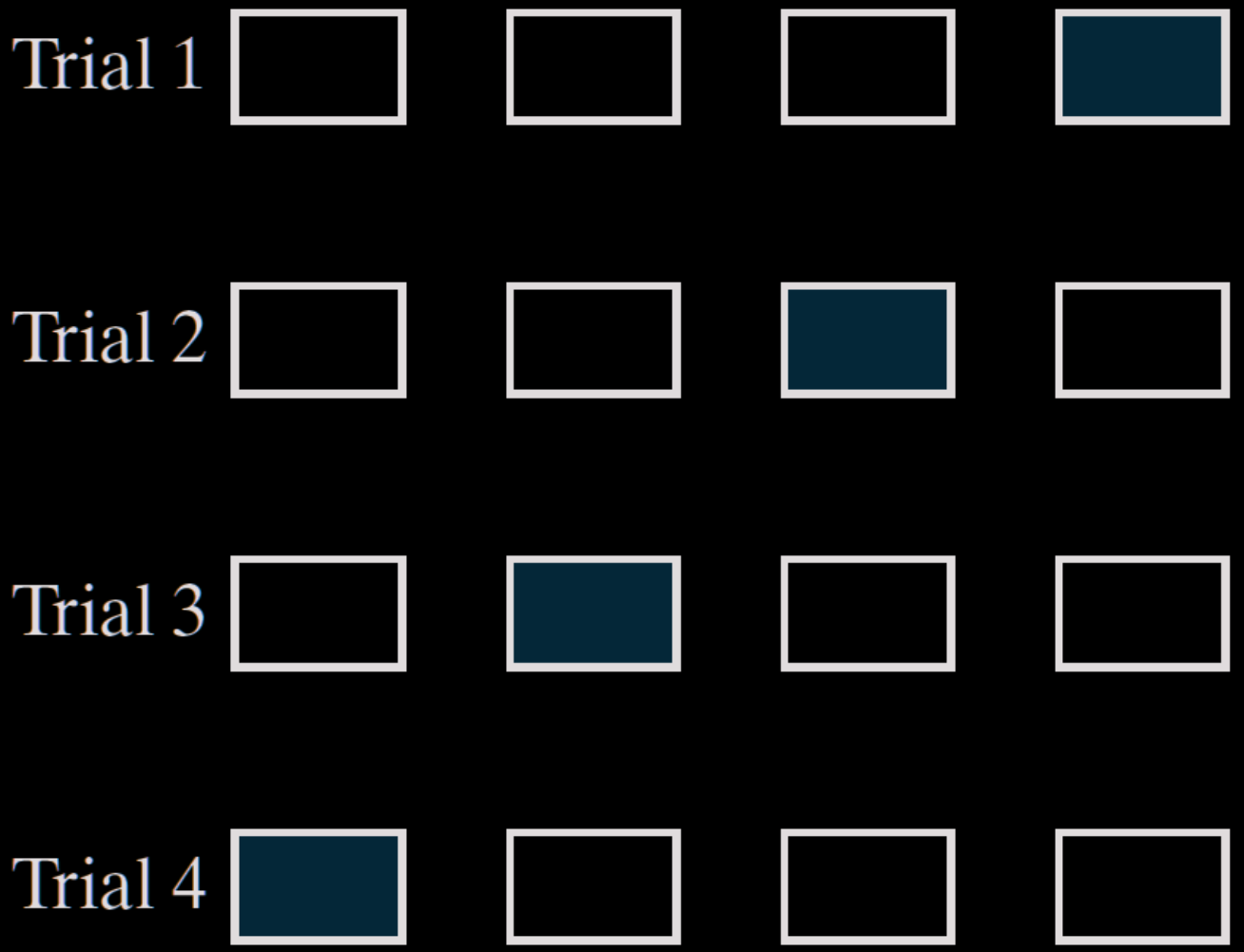


Fig. 4.21 Illustration of the *hold-out method* of cross-validation. For a given trial, the shaded subset of data is used to validate the model trained on the remaining data.

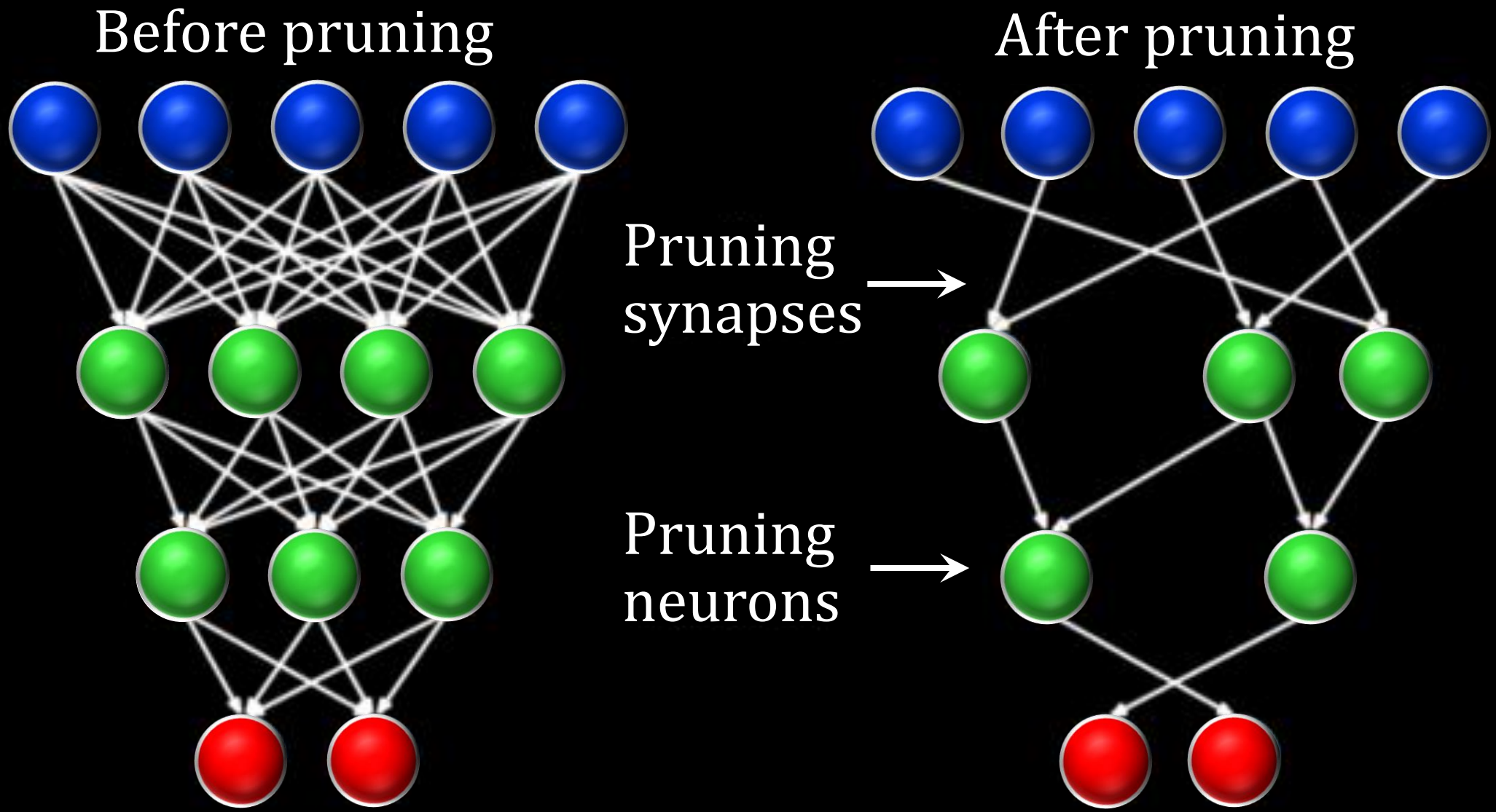
4.14 Cross-Validation: Variants of Cross-Validation

- However, *multifold cross-validation* may require excessive amount of computation since the model has to be trained K times, where $1 < K \leq N$.
- When the available number of labeled examples, N, is severely limited, we may use the extreme form of multifold cross-validation known as the leave-one-out method.
- In this case, N-1 examples are used to train the model and 1 example is used to test and validate the model.
- Experiment is repeated N times.

4.15 Network Pruning Techniques

- To solve real-world problems with neural networks usually requires the use of highly structured networks of a rather large size.
- Our objective is to minimize size of the network while maintaining good performance.
- We may achieve this design objective in one of two ways:
  - *Network growing*, in which case we start with a small multilayer perceptron, small for accomplishing the task at hand, and then add a new neuron or a new layer of hidden neurons only when we are unable to meet the design specification.
  - *Network pruning*, in which case we start with a large multilayer perceptron with an adequate performance for the problem at hand, and then prune it by eliminating certain synaptic weights in a selective and orderly fashion.

4.15 Network Pruning Techniques



Source: https://towardsdatascience.com/pruning-deep-neural-network-56cae1ec5505

4.15 Network Pruning Techniques
1. Complexity-Regularization

- In the context of *back-propagation learning* or any other supervised learning procedure, the trade-off by minimizing the total risk expressed as:

$$R(\mathbf{w}) = \mathcal{E}_s(\mathbf{W}) + \lambda \mathcal{E}_c(\mathbf{w}) \qquad (4.94)$$

- Which are the risk and error as a.
- In back-propagation learning, the standard performance measure $\mathcal{E}_s(\mathbf{W})$ is defined as a *mean-square error,* whose evaluation extends over the output neurons of the network and which is carried out for all the training examples on an epoch-by-epoch basis.
- The second term, $\mathcal{E}_c(\mathbf{w})$ is the complexity penalty of the network model scaled by the *regularization parameter* λ.

4.15 Network Pruning Techniques
1. Complexity-Regularization

- In a general setting, one choice of complexity-penalty term $\mathcal{E}_c(\mathbf{w})$ is the kth order smoothing integral

$$\mathcal{E}_c(\mathbf{w}, k) = \frac{1}{2}\int \left\| \frac{\partial^k}{\partial \mathbf{x}^k} F(\mathbf{x}, \mathbf{w}) \right\|^2 \mu(\mathbf{x}) d\mathbf{x} \qquad (4.95)$$

- where $F(\mathbf{x},\mathbf{w})$ is the input-output mapping performed by the model and $\mu(\mathbf{x})$ is some weighting function that determines the region of the input space over which the function $F(\mathbf{x},\mathbf{w})$ is required to be smooth.
- The motivation is to make the kth derivative of $F(\mathbf{x},\mathbf{w})$ with respect to the input vector $\mathbf{x}$ small.
- The larger we choose k, the smoother (i.e., less complex) the function $F(\mathbf{x},\mathbf{w})$ will become.

4.15 Network Pruning Techniques
1. Complexity-Regularization

- There are 3 different *complexity regularizations* for multilayer perceptrons
  1. Weight Decay
  2. Weight Elimination
  3. Approximate smoother
- In the ***weight-decay*** procedure, the complexity penalty term is defined as the squared norm of the weight vector **w** in the network, where all the weights in the multilayer perceptron are treated equally as

$$\mathcal{E}_c(\mathbf{w}) = \|\mathbf{w}\|^2 = \sum_{i \in \mathcal{C}_{total}} w_i^2 \qquad (4.96)$$

Where $\mathcal{C}_{total}$ refers to the synaptic weights in the network.

4.15 Network Pruning Techniques
1. Complexity-Regularization

- In the ***Weight Elimination*** complexity-regularization procedure, the complexity penalty is defined by

$$\mathcal{E}_c(\mathbf{w}) = \sum_{i \in \mathcal{C}_{total}} \frac{(w_i/w_0)^2}{1+(w_i/w_0)^2} \qquad (4.97)$$

where w_0 is a preassigned parameter, and w_i refers to the weight of some synapse *i* in the network. The set$\mathcal{C}_{total}$ refers to all the synaptic connections in the network.

- An individual penalty term varies with w_i/w_0 in a symmetric fashion, as shown in Fig. 4.22.
- Note that the *weight-elimination* procedure in (4.97) reduces to the *weight-decay* procedure in (4.96) for large w_0.

4.15 Network Pruning Techniques
1. Complexity-Regularization

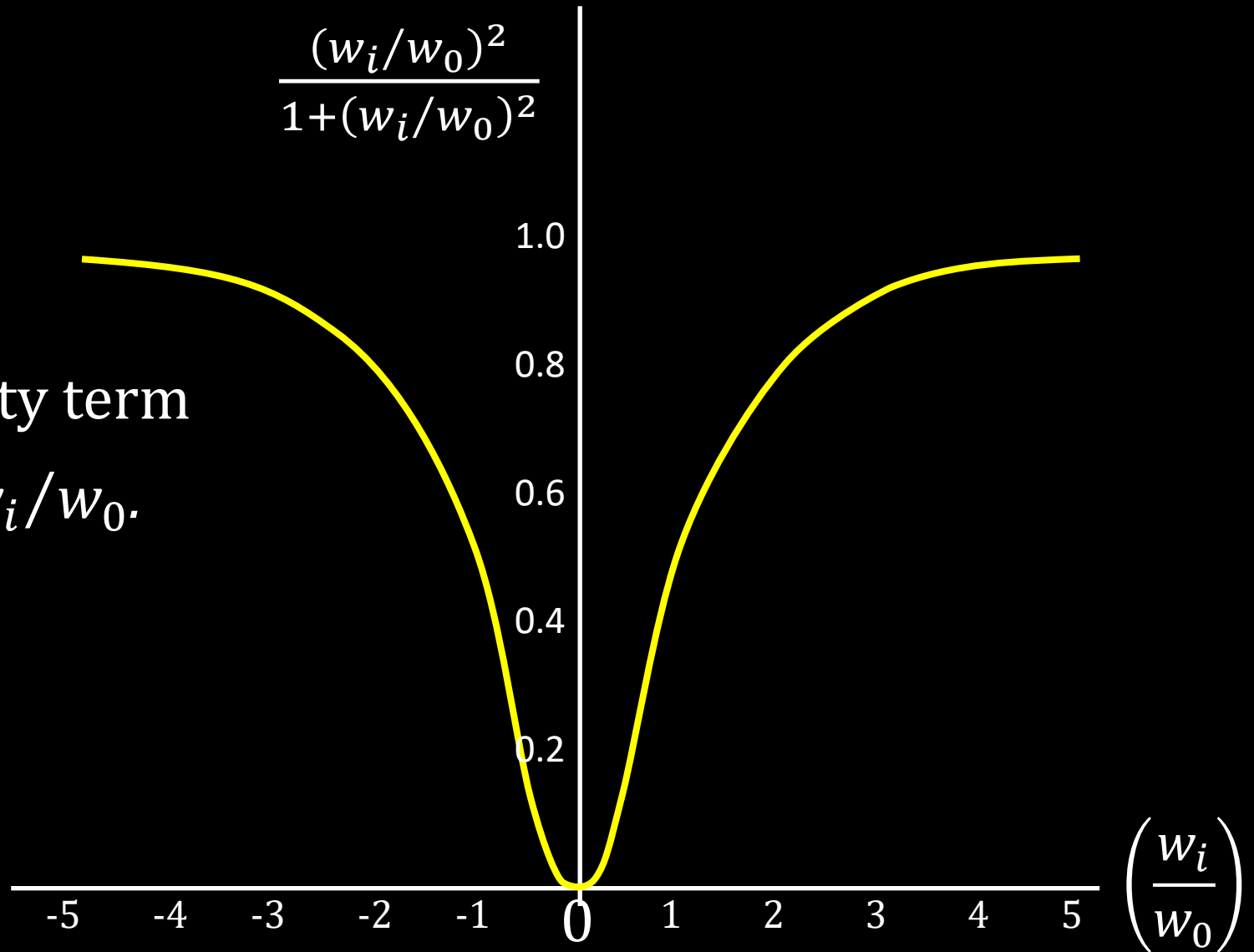


Fig. 4.22 The complexity penalty term $\frac{(w_i/w_0)^2}{1+(w_i/w_0)^2}$ plotted versus w_i/w_0.

4.15 Network Pruning Techniques
1. Complexity-Regularization

- ***Approximate Smoother***. In Moody and Rognvaldsson (1997), the following *complexity penalty* term is proposed for a multilayer perceptron with a single hidden layer and a single neuron in the output layer:

$$\mathcal{E}_c(\mathbf{w}) = \sum_{j=1}^{M} w_{0j}^2 \left\| w_j \right\|^p \tag{4.98}$$

where the w_{0j} are the weights in the output layer, and $\mathbf{w}_j$ is the weight vector for the jth neuron in the hidden layer; the power p is defined by

$$p = \begin{cases} 2k - 1 & \text{for a global smoother} \\ 2k & \text{for a local smoother} \end{cases} \tag{4.99}$$

where k is the order of differentiation of $F(\mathbf{x},\mathbf{w})$ with respect to $\mathbf{x}$.

4.15 Network Pruning Techniques
2. Hessian-Based Network Pruning

- The basic idea of this second approach to network pruning is to use information on second-order derivatives of the error surface in order to make a trade-off between *network complexity* and *training-error performance*.
- To construct such a model and find the local approximation of the cost function $\mathcal{E}_{av}$ using a Taylor series about the operating point, described as:

$$\mathcal{E}_{av}(\mathbf{w}+\boldsymbol{\Delta}\mathbf{w}) = \mathcal{E}_{av}(\mathbf{w}) + \mathbf{g}^{T}(\mathbf{w})\Delta\mathbf{w} + \frac{\mathbf{1}}{\mathbf{2}}\Delta\mathbf{w}^{T}\mathbf{H}\Delta\mathbf{w} + O(\|\Delta\mathbf{w}\|)^{3} \qquad (4.100)$$

where $\boldsymbol{\Delta}\mathbf{w}$ is a perturbation applied to the operating point $\mathbf{w}$, and $\mathbf{g}(\mathbf{w})$ is the gradient vector evaluated at $\mathbf{w}$ and $\mathbf{H}(\mathbf{w})$ is the Hessian.

4.15 Network Pruning Techniques
2. Hessian-Based Network Pruning

- The fulfill the requirement of pruning which is to identify a set of parameters whose deletion from the multilayer perceptron will cause the least increase in the value of the cost function $\mathcal{E}_{av}$, we make the following approximations:

1. *Extremal Approximation*. The parameters are deleted from the network only after the training process has converged (i.e., the network is fully trained).

   The implication of this assumption is that the parameters have a set of values corresponding to a local minimum or global minimum of the error surface.

   In such a case, the gradient vector $\mathbf{g}$ may be set equal to zero and the term $\mathbf{g}^T \Delta \mathbf{w}$ on the righthand side of (4.100) may therefore be ignored.

4.15 Network Pruning Techniques
2. Hessian-Based Network Pruning

2. Quadratic Approximation. We assume that the error surface around a local minimum or global minimum is nearly "quadratic." Hence the higher-order terms in (4.100) may be neglected. Then (4.100) becomes

$$\Delta\mathcal{E}_{av} = \mathcal{E}(\mathbf{w} + \Delta\mathbf{w}) + \mathcal{E}(\mathbf{w}) \cong \frac{1}{2}\Delta\mathbf{w}^T\mathbf{H}\Delta\mathbf{w} \qquad (4.101)$$

The *Optimal Brain Damage* (OBD) procedure simplifies the computations by assuming that the Hessian matrix $\mathbf{H}$ is a diagonal.

The goal of *Optimal Brain Surgery* (OBS) is to set one of the synaptic weights to zero to minimize the incremental increase in $\mathcal{E}_{av}$ given in (4.101)

4.15 Network Pruning Techniques
2. Hessian-Based Network Pruning

The elimination of this weight is equivalent to the condition

$$\Delta w_i + w_i = 0 \qquad \text{or} \qquad \mathbf{1}_i^T \Delta \mathbf{w} + w_i = 0 \tag{4.102}$$

Where $\mathbf{1}_i$ is the unit vector of all zeroes except for the *i*th element is 1.

There are two levels of minimization going on here.

1. minimization of the synaptic weight vectors that remain after the *i*th weight vector is set equal to zero.
2. The second minimization is over which particular vector is pruned.

4.15 Network Pruning Techniques
2. Hessian-Based Network Pruning

- To solve this constrained optimization problem, we construct the Lagrangian

$$S = \frac{1}{2}\Delta\mathbf{w}^T\mathbf{H}\Delta\mathbf{w} - \lambda\left(\mathbf{1}_i^T\Delta\mathbf{w} + w_i\right) \quad (103)$$

  where λ is the *Lagrange multiplier*.

- Taking the first derivative of S in (4.103) w.r.t. $\Delta\mathbf{w}$ and applying constraint in (4.102), we find the optimum change in weight vector $\mathbf{w}$ as

$$\Delta\mathbf{w} = \frac{w_i}{[\mathbf{H}^{-1}]_{i,i}}\boldsymbol{H}^{-\mathbf{1}}\mathbf{1}_i \quad (4.104)$$

- and the corresponding optimum value of the Lagrangian S for element w_i is

$$S = \frac{w_i^2}{2[\mathbf{H}^{-1}]_{i,i}} \quad (4.105)$$

- where $[\mathbf{H}^{-1}]_{i,i}$ is the i,i-th element of this inverse Hessian matrix.

4.15 Network Pruning Techniques
2. Hessian-Based Network Pruning

- The Lagrangian S_i optimized w.r.t. $\Delta\mathbf{w}$, subject to the constraint that the ith synaptic weight w_i be eliminated, is called the *saliency* of w_i, with an increase in the mean-square error
- Since saliency S_i is proportional to w_i^2, small weights have a small effect on the mean-square error.

4.15 Network Pruning Techniques
Computing the Inverse Hessian Matrix.

- When the number of free parameters, W, in the network is large, the problem of computing the inverse Hessian matrix $\mathbf{H}^{-1}$ may be intractable and formulation of the Optimal Brain Surgery (OBS) procedure fails.
- Assuming that the multilayer perceptron which has a single-output neuron is fully trained to a local minimum on the error surface. For a given training set we may express the cost function as

$$\mathcal{E}_{av}(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}(d(n) - o(n))^2$$

Where $F(\mathbf{w}, \mathbf{x})$ is the input-output mapping function, $o(n) = F(\mathbf{w}, \mathbf{x})$ is the actual output of the network on the presentation of the nth example, $d(n)$ is the corresponding desired response and N is the total number of examples in the training set and .

4.15 Network Pruning Techniques
Computing the Inverse Hessian Matrix.

- The first derivative $\mathcal{E}_{av}$ with respect to $\mathbf{w}$ is therefore

$$\frac{\partial \mathcal{E}_{av}}{\partial \mathbf{w}} = -\frac{1}{N}\sum_{n=1}^{N}\frac{\partial F(\mathbf{w},\mathbf{x}(n))}{\partial \mathbf{w}}\big(d(n)-o(n)\big) \qquad (4.106)$$

- Then the second derivative

$$\mathbf{H}(n) = \frac{\partial^2 \mathcal{E}_{av}}{\partial \mathbf{w}^2} = \frac{1}{N}\sum_{n=1}^{N}\left\{\left(\frac{\partial F(\mathbf{w},\mathbf{x}(n))}{\partial \mathbf{w}}\right)\left(\frac{\partial F(\mathbf{w},\mathbf{x}(n))}{\partial \mathbf{w}}\right)^{T} - \left(\frac{\partial^2 F(\mathbf{w},\mathbf{x}(n))}{\partial \mathbf{w}^2}\right)\big(d(n)-o(n)\big)\right\} \qquad (4.107)$$

- Under the assumption that the network is fully trained (that is, the cost function $\mathcal{E}_{av}$ has been adjusted to a local minimum on the error surface), it is reasonable to say that $o(n)$ is close to $d(n)$. Hence, we may ignore the second term and approximate (4.107) as

$$\mathbf{H}(n) \cong \frac{1}{N}\sum_{n=1}^{N}\left\{\left(\frac{\partial F(\mathbf{w},\mathbf{x}(n))}{\partial \mathbf{w}}\right)\left(\frac{\partial F(\mathbf{w},\mathbf{x}(n))}{\partial \mathbf{w}}\right)^{T}\right\} \qquad (4.108)$$

4.15 Network Pruning Techniques
Computing the Inverse Hessian Matrix.

- Defining

$$\xi(n) = \frac{1}{\sqrt{N}} \frac{\partial F(\mathbf{w},\mathbf{x}(n))}{\partial \mathbf{w}} \quad (4.109)$$

Then we rewrite (4.108) as

$$\mathbf{H}(n) = \sum_{k=1}^{N} \xi(k)\xi^T(k) = \mathbf{H}(n-1) + \xi(n)\xi^T(n), \quad n = 1, 2, \dots, N \quad (4.110)$$

- Application of the matrix inversion lemma (see next page) yields the desired formula for recursive computation of the inverse Hessian:

$$\mathbf{H}^{-1}(n) = \mathbf{H}^{-1}(n-1) - \frac{\mathbf{H}^{-1}(n-1)\xi(n)\xi^T(n)\mathbf{H}^{-1}(n-1)}{1+\xi^T(n)\mathbf{H}^{-1}(n-1)\xi(n)} \quad (4.111)$$

- Note that the denominator of (4.111) is scalar. It is therefore straightforward to calculate its reciprocal. A summary of the brain surgeon algorithm is presented in Table 4.6.

4.15 Network Pruning Techniques
Matrix Inversion Lemma

- Let $\mathbf{A}$ and $\mathbf{B}$ denote two positive definite matrices related by

$$\mathbf{A} = \mathbf{B}^{-1} + \mathbf{C}\mathbf{D}\mathbf{C}^{\mathrm{T}}$$

where $\mathbf{C}$ and $\mathbf{D}$ are two other matrices. According to the matrix inversion lemma, the inverse of matrix $\mathbf{A}$ is defined by

$$\mathbf{A}^{-1} = \mathbf{B} - \mathbf{B}\mathbf{C}\left(\mathbf{D} + \mathbf{C}^{\mathrm{T}}\mathbf{B}\mathbf{C}\right)^{-1}\mathbf{C}^{\mathrm{T}}\mathbf{B}$$

- For the problem described in Eq. (4.110) we have

$$\mathbf{A} = \mathbf{H}(n)$$
$$\mathbf{B}^{-1} = \mathbf{H}(n-1)$$
$$\mathbf{C} = \xi(n)$$
$$\mathbf{D} = 1$$

TABLE 4.6 Summary of the Optimal Brain Surgeon Algorithm

1. Train the given multilayer perceptron to minimum mean-square error.
2. Use the procedure described in Section 4.8 to compute the vector

$$\boldsymbol{\xi}(n) = \frac{1}{\sqrt{N}} \frac{\partial F(\mathbf{w}, \mathbf{x}(n))}{\partial \mathbf{w}}$$

where $F(\mathbf{w}, \mathbf{x}(n))$ is the input–output mapping realized by the multilayer perceptron with an overall weight vector $\mathbf{w}$, and $\mathbf{x}(n)$ is the input vector.

3. Use the recursion in Eq. (4.108) to compute the inverse Hessian $\mathbf{H}^{-1}$.
4. Find the i that corresponds to the smallest saliency

$$S_i = \frac{w_i^2}{2[\mathbf{H}^{-1}]_{i,i}}$$

where $[\mathbf{H}^{-1}]_{i,i}$ is the (i,i)th element of $\mathbf{H}^{-1}$. If the saliency S_i is much smaller than the mean-square error $\mathscr{E}_{\text{av}}$, then delete the synaptic weight w_i and proceed to step 5. Otherwise, go to step 6.

5. Update all the synaptic weights in the network by applying the adjustment

$$\Delta \mathbf{w} = -\frac{w_i}{[\mathbf{H}^{-1}]_{i,i}} \mathbf{H}^{-1} \mathbf{1}_i$$

Go to step 2.

6. Stop the computation when no more weights can be deleted from the network without a large increase in the mean-square error. (It may be desirable to retrain the network at this point).

End of Section 4.9-4.15