EENG582: Artificial Neural Networks

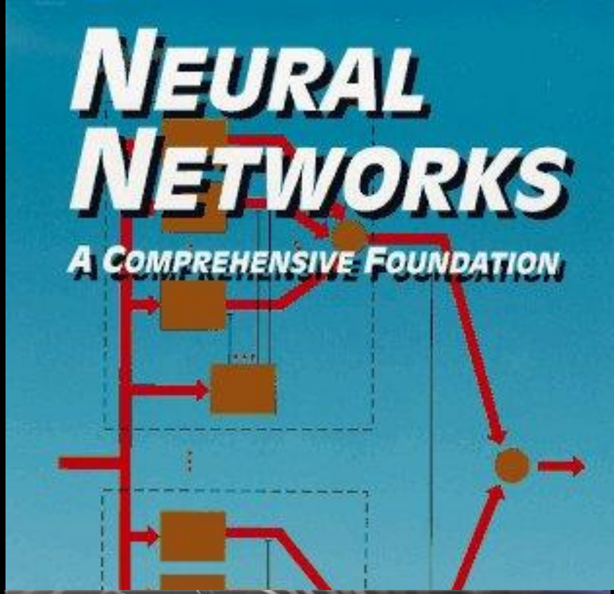
Neural Networks
A Comprehensive Foundation
by S. Haykin

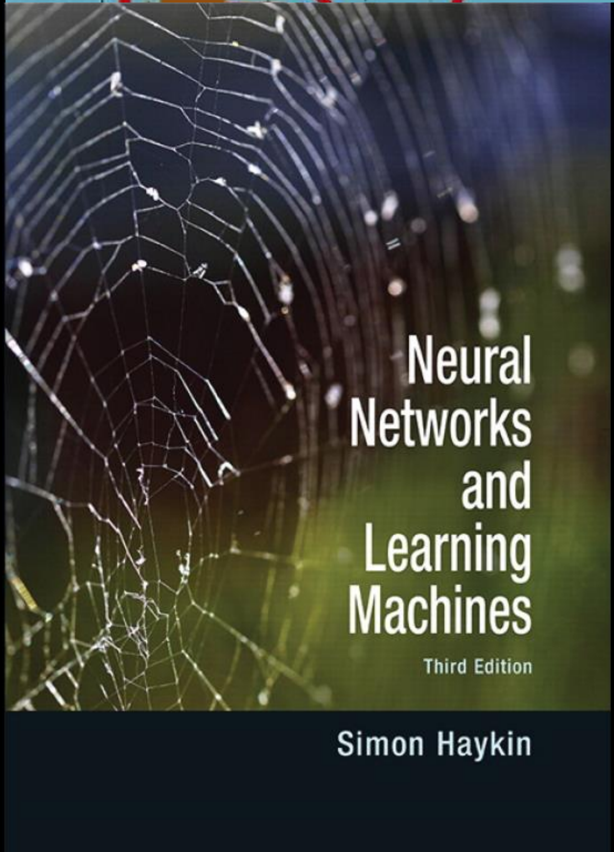
Sections 4.9 – 4.x

4 Multilayer Perceptrons

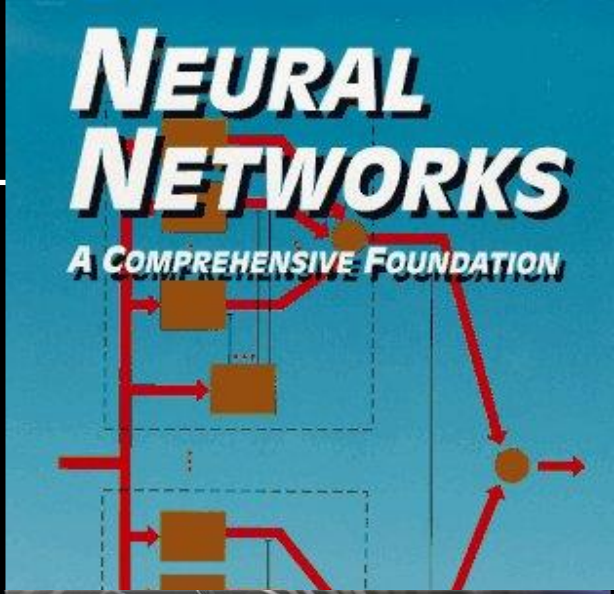
Prof. Dr. Hasan AMCA
Electrical and Electronic Engineering Department
(ee.emu.edu.tr)

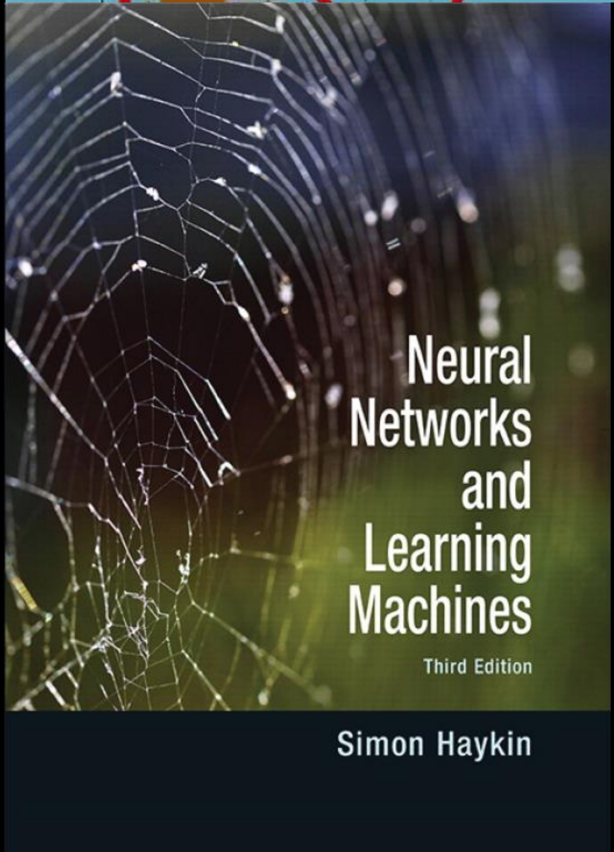
Eastern Mediterranean University
(emu.edu.tr)





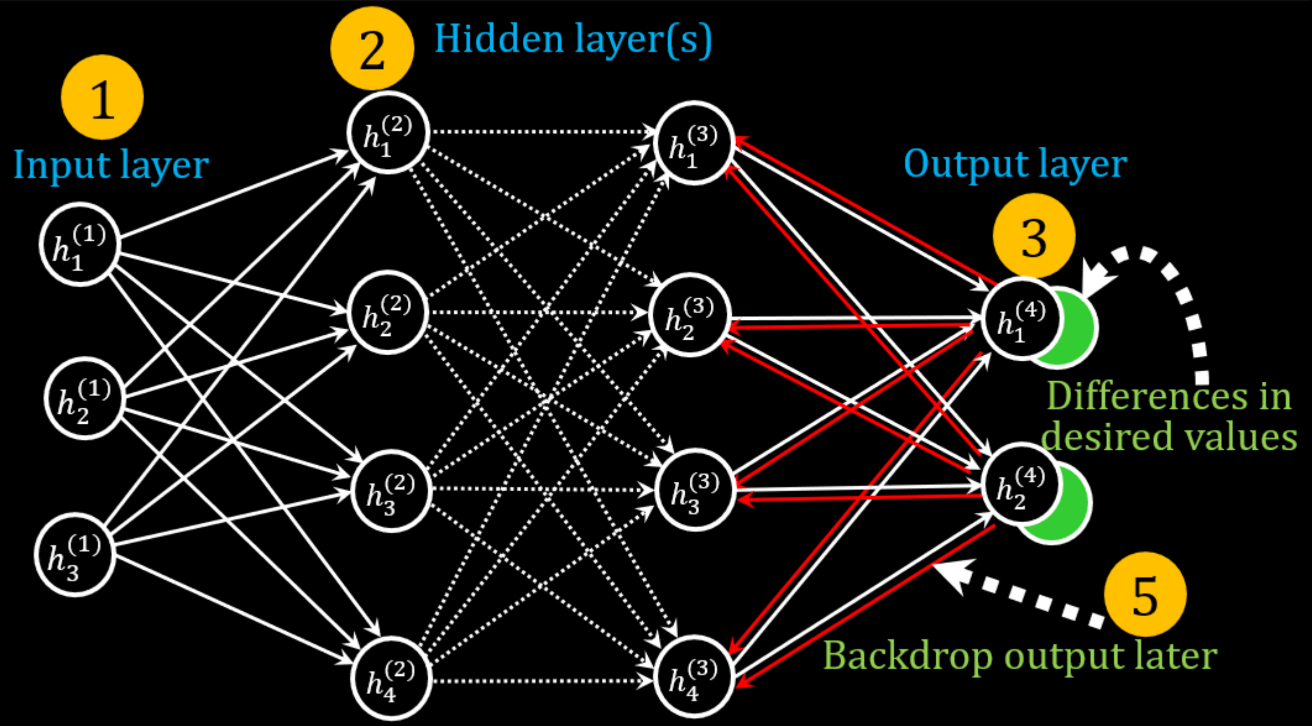
4 Multi Layer Perceptrons



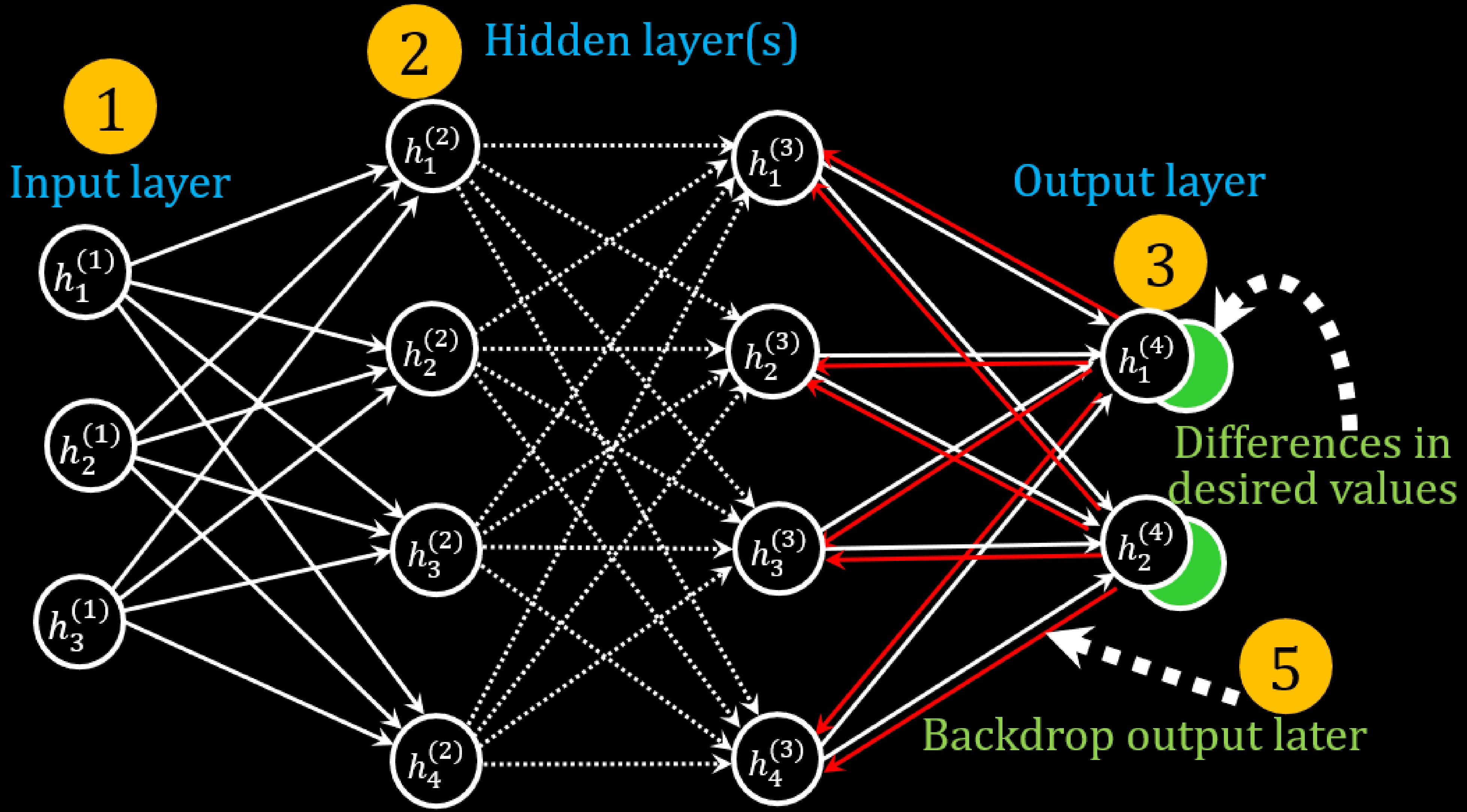


4.16 Virtues and Limitations of Back-Propagation Learning

- The back-propagation algorithm is the most popular algorithm for the supervised training of multilayer perceptrons.
- It is a gradient (derivative) technique and not an optimization technique.
- Back-propagation has two distinct properties:
  1. It is simple to compute locally.
  2. It performs stochastic gradient descent in weight space (for pattern-by-pattern updating of synaptic weights).



4.16 Virtues and Limitations of Back-Propagation Learning



4.16 Virtues and Limitations of Back-Propagation Learning

Connectionism

- The back-propagation algorithm is an example of a *connectionist paradigm* that relies on local computations to discover the information-processing capabilities of neural networks.
- The use of local computations in the design of artificial neural networks is usually advocated for three principal reasons:
    1. Artificial neural networks that perform local computations are often held up as metaphors for biological neural networks.
    2. The use of local computations permits a graceful degradation in performance due to hardware errors, and therefore provides the basis for a fault-tolerant network design.
    3. Local computations favor the use of parallel architectures as an efficient method for the implementation of artificial neural networks.

4.16 Virtues and Limitations of Back-Propagation Learning
Feature Detection

- The hidden neurons of a multilayer perceptron trained with the back-propagation algorithm play a critical role as feature detectors.
- The multilayer perceptron can be employed as a *replicator* or *identity map*.
- Figure 4.23 illustrates how this can be accomplished for the case of a multilayer perceptron using a single hidden layer.
- The network layout satisfies the following structural requirements, as illustrated in Fig. 4.23a:
  - The input and output layers have the same size, m.
  - The size of the hidden layer, M, is smaller than m.
  - The network is fully connected.

4.16 Virtues and Limitations of Back-Propagation Learning

Feature Detection

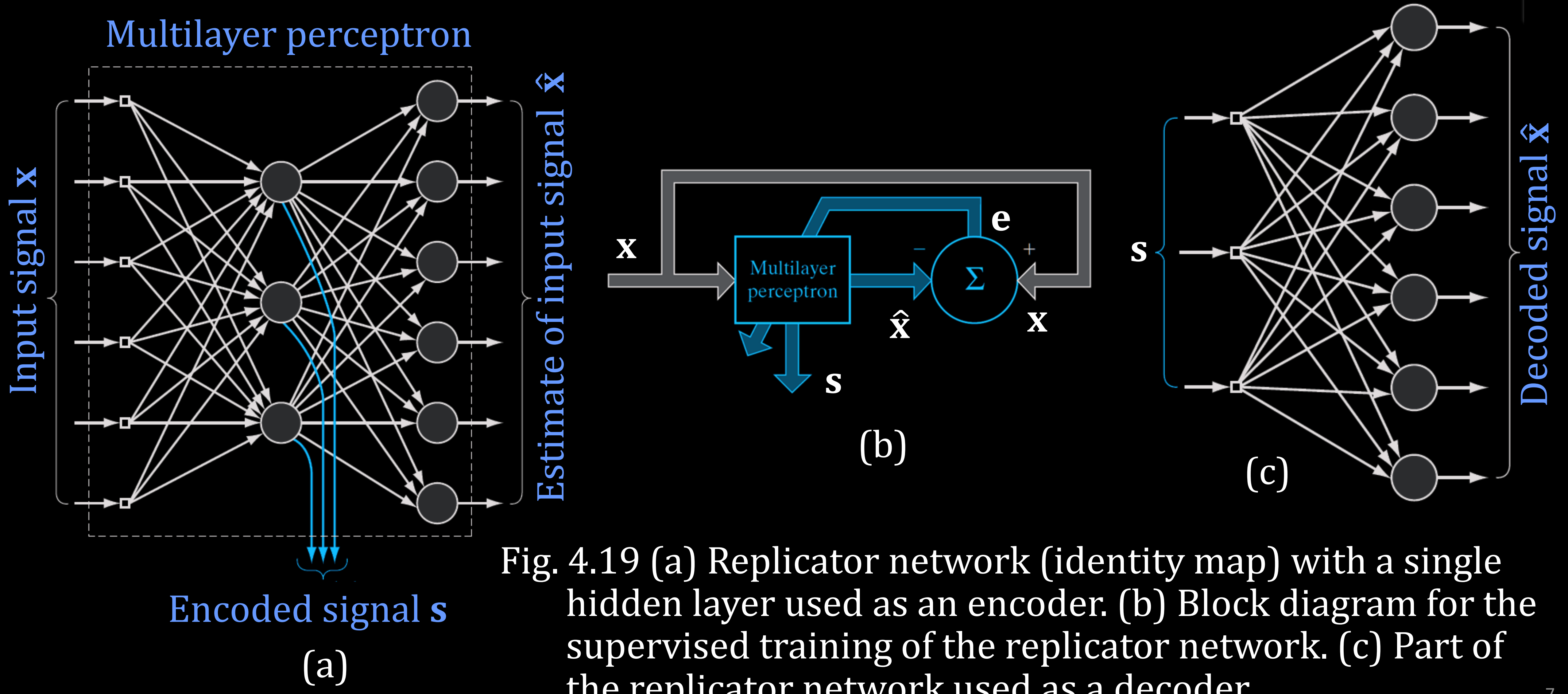


Fig. 4.19 (a) Replicator network (identity map) with a single hidden layer used as an encoder. (b) Block diagram for the supervised training of the replicator network. (c) Part of the replicator network used as a decoder.

4.16 Virtues and Limitations of Back-Propagation Learning
Feature Detection

- The network is trained using the back-propagation algorithm with the estimation error vector $(\mathbf{x} - \hat{\mathbf{x}})$ treated as the error signal, as in Fig. 4.23b.
- Training is performed in an unsupervised manner (i.e., without a teacher).
- The network is constrained to perform identity mapping through its hidden layer.
- As depicted in Fig. 4.23a, the fully trained multilayer perceptron performs the role of an "encoder" to reconstruct an estimate $\hat{\mathbf{x}}$ of the original input pattern $\mathbf{x}$ (i.e., to perform decoding).
- The encoded signal is applied to the hidden layer of the replica tor network, as illustrated in Fig. 4.23c.
- In effect, this latter network performs the role of a "decoder".

4.16 Virtues and Limitations of Back-Propagation Learning

Function Approximation

- A multilayer perceptron trained with the back-propagation algorithm as a nested sigmoidal scheme can be written in the following compact form

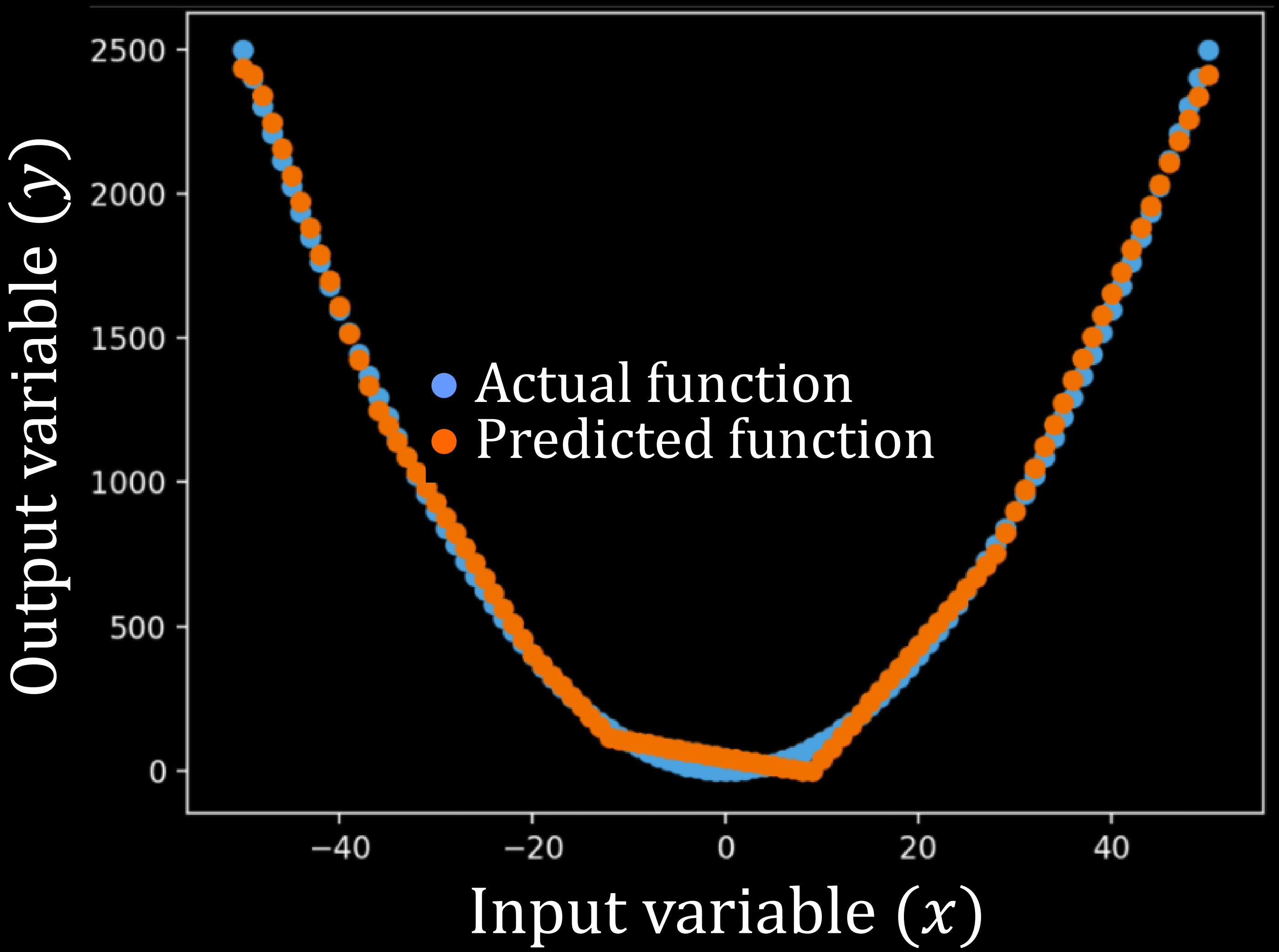
$$F(\mathbf{x},\mathbf{w}) = \varphi\left(\sum_k w_{0k}\varphi\left(\sum_j w_{kj}\varphi\left(\ldots\varphi\left(\sum_i w_{li}x_i\right)\right)\right)\right) \qquad (4.113)$$

- where $\varphi(.)$ is a common sigmoid activation function, wok is the synaptic weight from neuron k in the last hidden layer to the single output neuron o, and so on

- and x_i is the ith element of the input vector $\mathbf{x}$, w denotes the set of synaptic weights.

4.16 Virtues and Limitations of Back-Propagation Learning
Function Approximation

- Function approximation is a technique for estimating an unknown underlying function using available observations (data) from the domain.
- Artificial neural networks learn to approximate a function.
- In supervised learning, a dataset is comprised of inputs and outputs, and the supervised learning algorithm learns how to best map examples of inputs to examples of outputs.



Source: Function Approximation by Python https://machinelearningmastery.com/neural-networks-are-function-approximators/

4.16 Virtues and Limitations of Back-Propagation Learning
Computational Efficiency

- Computational complexity of an algorithm is measured in terms of number of multiplications, additions and storage involved in its implementation.
- the back-propagation algorithm is computationally efficient in using it to train a multilayer perceptron containing total of W synaptic weights (including biases),
  - its computational complexity is linear in W.
  - In the forward pass, the only computations involving the synaptic weights are those that relate to the induced local fields of the various neurons in the network.
  - From (4.44) these computations are all linear in synaptic weights of the network.
- The computational complexity of the back-propagation algorithm is linear in W, that is, it is $O(W)$.

4.16 Virtues and Limitations of Back-Propagation Learning
Sensitivity Analysis

- The sensitivity of an input-output mapping function F with respect to a parameter of the function, denoted by w, is defined by

$$S_w^F = \frac{\partial F/F}{\partial w/w} \tag{4.114}$$

- It can be shown that the partial derivatives of the function $F(\mathbf{w})$ with respect to all the elements of the weight vector $\mathbf{w}$ can be computed efficiently.
- The complexity involved in computing each of these partial derivatives is linear in the total number of weights contained in the network, W.

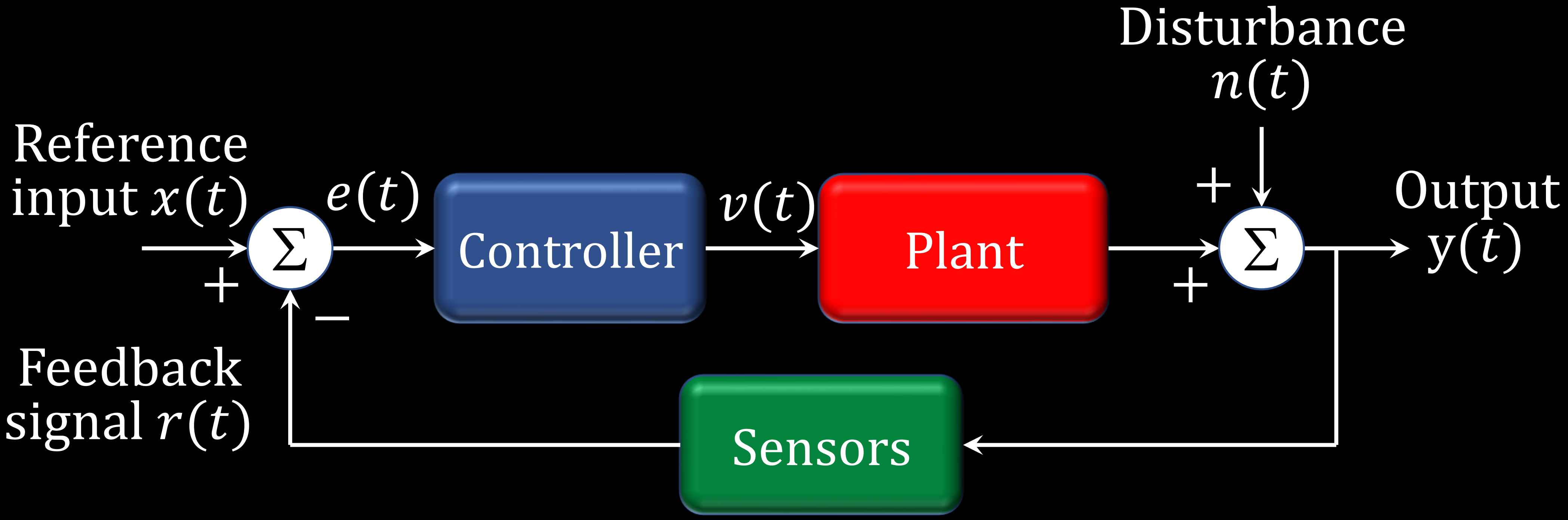
4.16 Virtues and Limitations of Back-Propagation Learning
Robustness

- The LMS algorithm is robust in the sense that small disturbances can only give rise to small estimation errors.
- Since observation model is linear, the LMS algorithm is an H^{∞}-optimal filter.
- The LMS algorithm minimizes the maximum energy gain from the disturbances to the estimation errors.
- If the underlying observation model is nonlinear, the back-propagation algorithm is a locally H^{∞}-optimal filter.
- The term "local" used here means that the initial value of the weight vector used in the back-propagation algorithm is sufficiently close to the optimum value w* of the weight vector to ensure that the algorithm does not get trapped in a poor local minimum.

4.16 Virtues and Limitations of Back-Propagation Learning

Robustness

- The LMS algorithm is robust in the sense that small disturbances can only give rise to small estimation errors.



4.16 Virtues and Limitations of Back-Propagation Learning
Convergence

- The back-propagation algorithm uses an "*instantaneous estimate*" for the gradient of the error surface in weight space.
- The algorithm is therefore stochastic in nature; that is, it has a tendency to zigzag its way about the true direction to a minimum on the error surface.
- Consequently, it tends to converge slowly due to the fact that

1. Error surface is fairly flat along a weight dimension and derivative of the error surface with respect to that weight is small in magnitude and adjustment applied to the weight is small and many iterations of the algorithm may be required to produce a significant reduction in error.

   In this second situation, the adjustment applied to weight is large, which may cause the algorithm to overshoot the minimum of the error surface.

4.16 Virtues and Limitations of Back-Propagation Learning

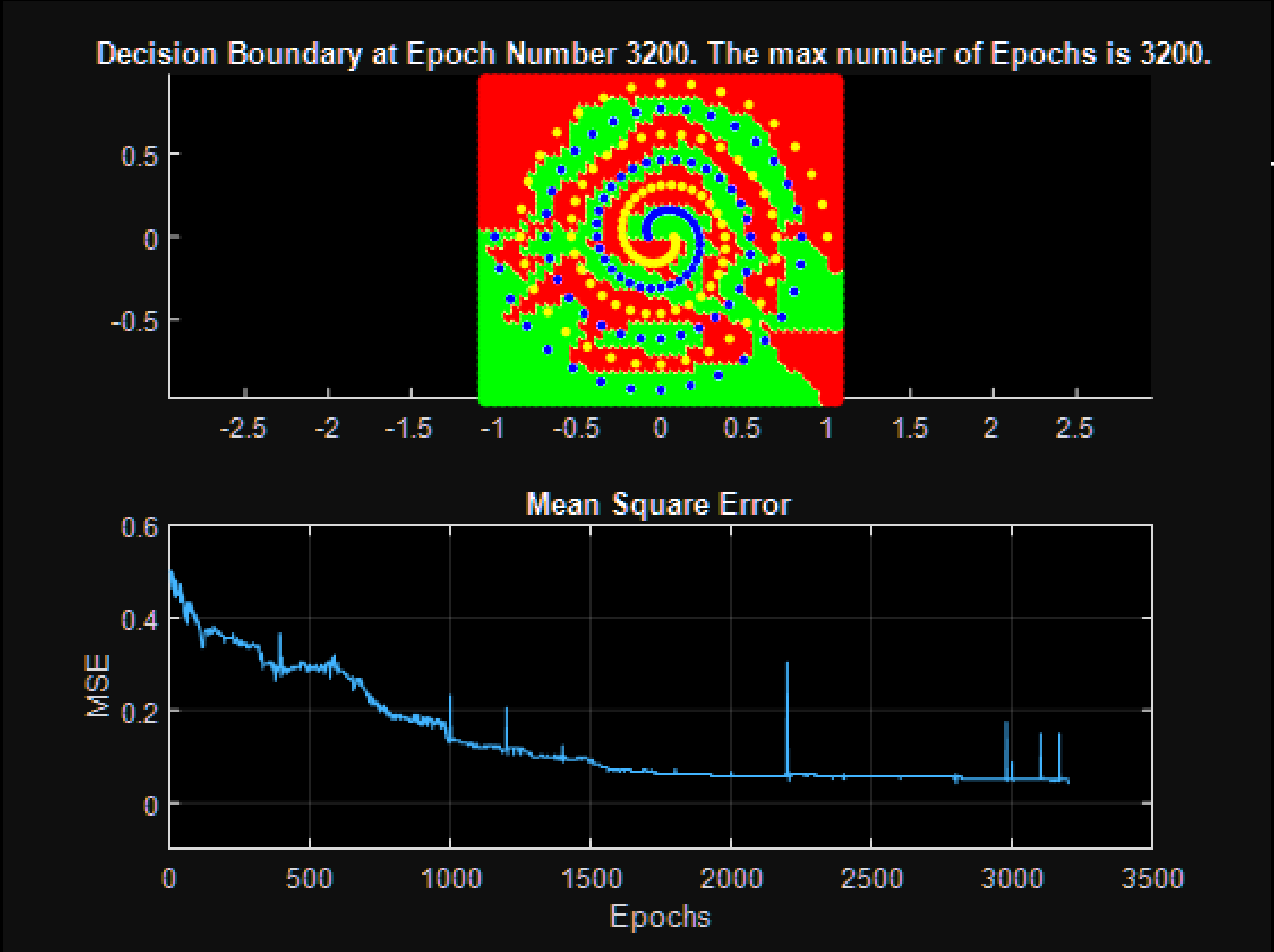
Convergence

2. The direction of the negative gradient vector (i.e., the negative derivative of the cost function with respect to the vector of weights) may point away from the minimum of the error surface: hence, the adjustments applied to the weights may induce the algorithm to move in the wrong direction.

- MATLAB Example: A MLP NN program with 2 hidden layers, manually adjustable leraning rate 0.15 and momentum alpha 0.05 Multilayer Perceptron (MLP) Neural Network Function using MATLAB: An implementation for Multilayer Perceptron Feed Forward Fully Connected Neural Network with a sigmoid activation function. The training is done using the Backpropagation algorithm with options for Resilient Gradient Descent, Momentum Backpropagation, and Learning Rate Decrease. The training stops when the Mean Square Error (MSE) reaches zero or a predefined maximum number of epochs is reached. Four example data for training and testing are included with the project. They are generated by SharkTime Sharky Neural Network (http://sharktime.com/us_SharkyNeuralNetwork.html). Copyright (C) 9-2015 Hesham M. Eraqi. All rights reserved. hesham.eraqi@gmail.com. (p.t.o.)

4.16 Virtues and Limitations of Back-Propagation Learning

Convergence

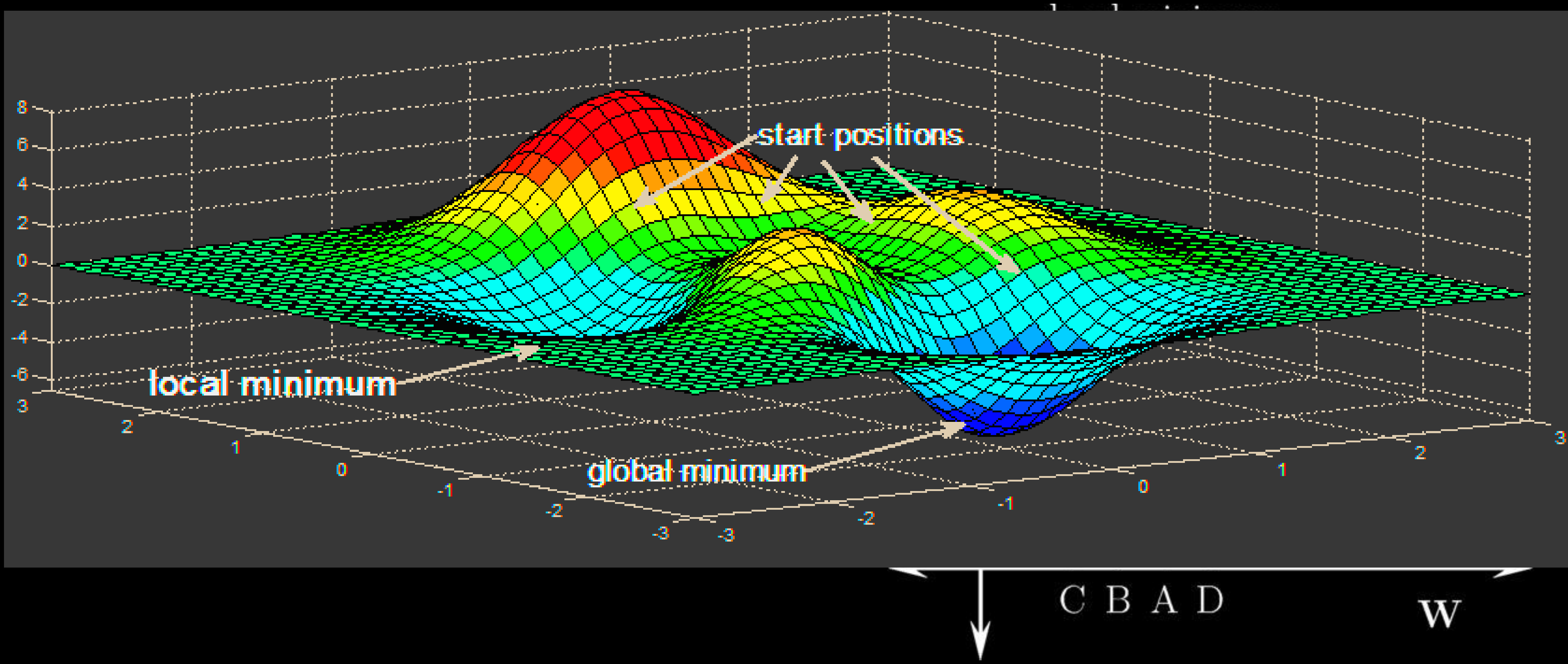


Source: Hesham Eraqi (2021). MLP Neural Network with Backpropagation (https://www.mathworks.com/matlabcentral/fileexchange/54076-mlp-neural-network-with-backpropagation), MATLAB Central File Exchange. Retrieved March 8, 2021.

4.16 Virtues and Limitations of Back-Propagation Learning
Local Minima

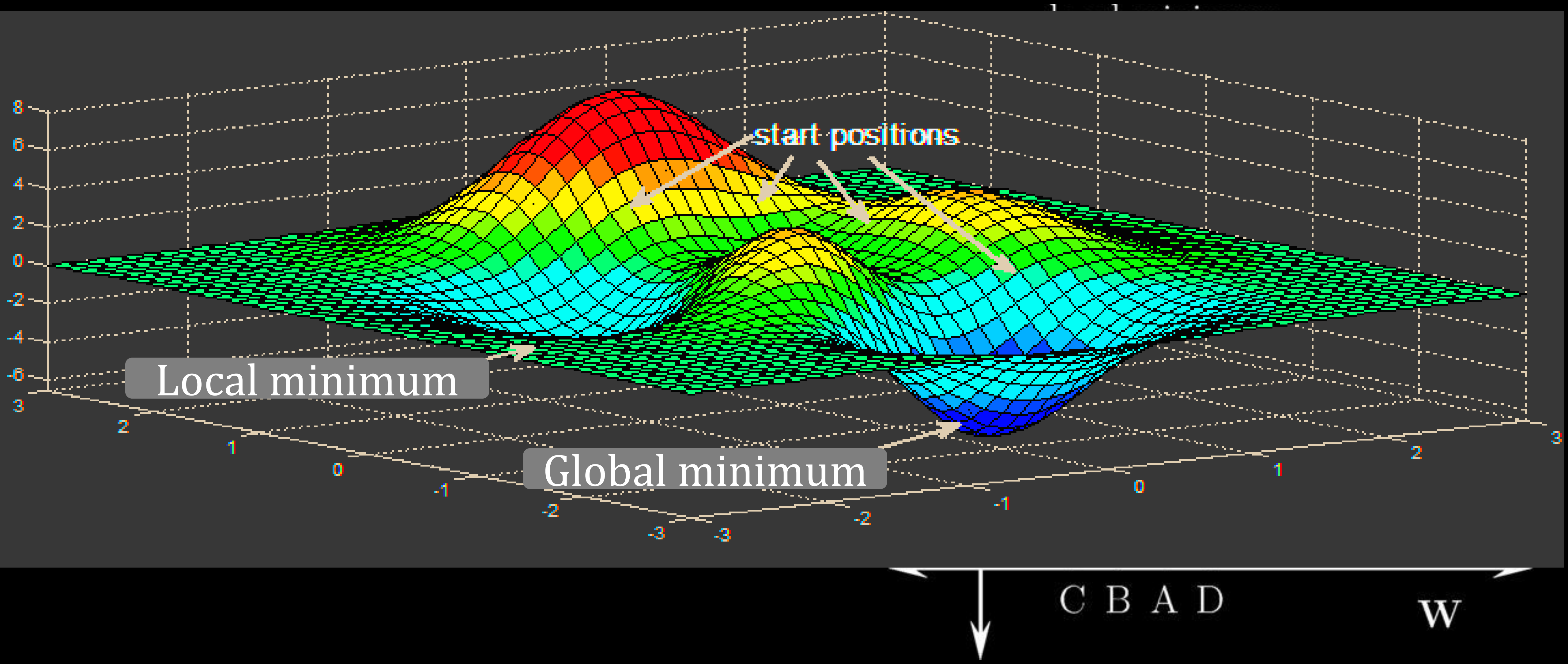
- Another peculiarity of the error surface that affects the performance of the backpropagation algorithm is the presence of local minima (i.e., isolated valleys) in addition to global minima;
- In general, it is difficult to determine the numbers of local and global minima.
- Since back-propagation learning is basically a hill-climbing technique, it runs the risk of being trapped in a local minimum where every small change in synaptic weights increases the cost function.
- But somewhere else in the weight space, there exists another set of synaptic weights for which the cost function is smaller than the local minimum in which the network is stuck.
- It is clearly undesirable to have the learning process terminate at a local minimum, especially if it is located far above a global minimum.

4.16 Virtues and Limitations of Back-Propagation Learning
Local Minima



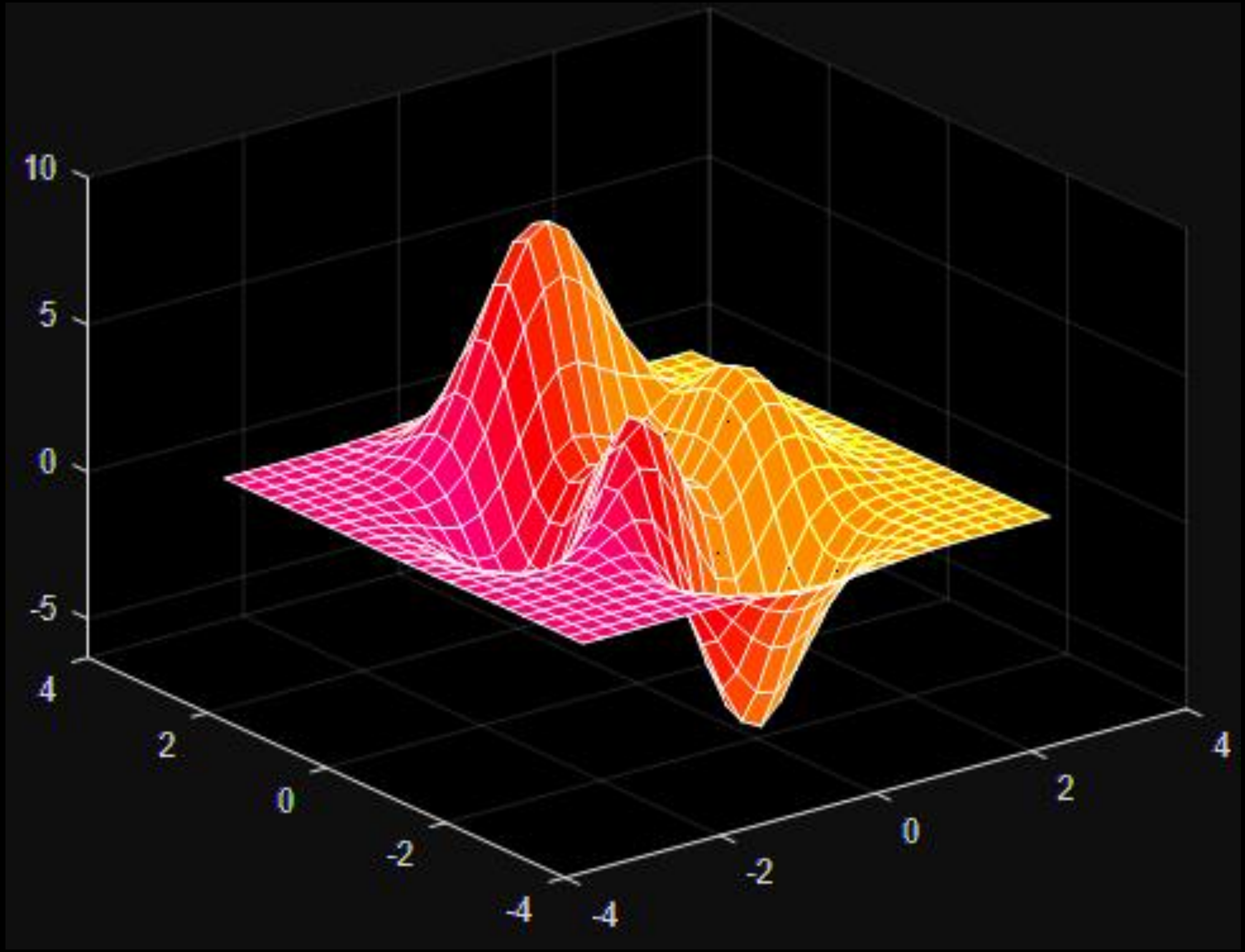
https://www.researchgate.net/publication/6792622_On_Adaptive_Learning_Rate_That_Guarantees_Convergence_in_Feedforward_Networks/figures?lo=1

4.16 Virtues and Limitations of Back-Propagation Learning
Local Minima



4.16 Virtues and Limitations of Back-Propagation Learning

Local Minima



https://www.researchgate.net/publication/6792622_On_Adaptive_Learning_Rate_That_Guarantees_Convergence_in_Feedforward_Networks/figures?lo=1

4.16 Virtues and Limitations of Back-Propagation Learning

Scaling

- Scaling addresses the issue of how well the network behaves as the computational task increases in size and complexity
  - (e.g., as measured by the time required for training or the best generalization performance attainable) .
- Among the many possible ways of measuring the size or complexity of a computational task, the predicate order provides the most useful and important measure.
- To explain what predicate means, let $\psi(X)$ denote a function that can have only two values as 0 and 1. For example, we may write

$$\psi_{\mathrm{CIRCLE}}(X) = \begin{cases} 1 & \text{if the figure } X \text{ is circle} \\ 0 & \text{if the figure } X \text{ is not a circle} \end{cases}$$

4.16 Virtues and Limitations of Back-Propagation Learning

Scaling

- Using the idea of a predicate, the use of a multilayer perceptron trained with the back-propagation algorithm to learn to compute the parity function defined by

$$\psi_{\text{PARITY}}(X) = \begin{cases} 1 & \text{if } |X| \text{ is an odd number} \\ 0 & \text{otherwise} \end{cases}$$

- Since a multilayer perceptron should not be fully connected, how should the synaptic connections of the network be allocated?
- This question is of no major concern in the case of small-scale applications, but it is certainly crucial to the successful application of backpropagation learning for solving large-scale, real-world problems.
- The network architecture and the constraints imposed on synaptic weights of the network should be designed so as to incorporate prior information about the task into the makeup of the network.

4.16 Supervised Learning Viewed as an Optimization Problem …

- In this section we view the supervised training of a multilayer perceptron as a problem in *numerical optimization*.
- We point out that error surface of a multilayer perceptron with supervised learning is a nonlinear function of weight vector **w** (synaptic weights).
- Let $\mathcal{E}_{av}$ denote the cost function, averaged over the training sample.
- Using the Taylor series, we may expand about the current operating point on the error surface as in (4.97), reproduced here in the form:

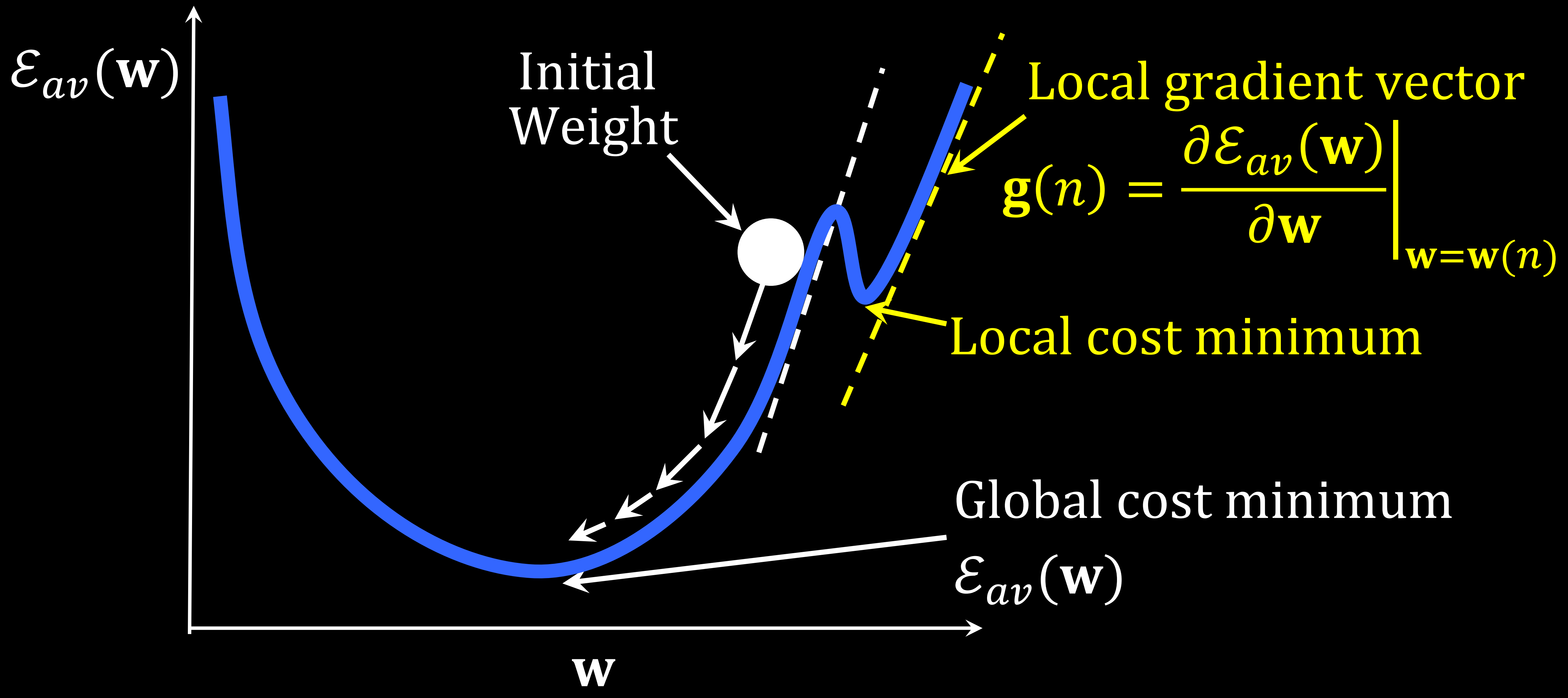
$$\mathcal{E}_{av}\big(\mathbf{w}(n) + \Delta\mathbf{w}(n)\big) = \mathcal{E}_{av}\big(\mathbf{w}(n)\big) + g^T(n)\Delta\mathbf{w}(n) + \frac{1}{2}\Delta\mathbf{w}^T(n)\mathbf{H}(n)\Delta\mathbf{w}(n) + $$

$$+(\text{3rd} + \text{higher ordered terms}) \quad (4.111)$$

Where the local gradient vector $\mathbf{g}(n)$ is defined by

$$\mathbf{g}(n) = \left.\frac{\partial \mathcal{E}_{av}(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}(n)} \quad (4.112)$$

4.16 Supervised Learning Viewed as an Optimization Problem …



4.16 Supervised Learning Viewed as an Optimization Problem …

- The local Hessian matrix $\mathbf{H}(n)$ represent the curvature of error performance surface employing batch mode learning

$$\mathbf{H}(n) = \left.\frac{\partial^2 \mathcal{E}_{av}(\mathbf{w})}{\partial \mathbf{w}^2}\right|_{\mathbf{w}=\mathbf{w}(n)} \qquad (4.112)$$

- In the steepest-descent method, exemplified by back-propagation algorithm, adjustment $\Delta\mathbf{w}(n)$ applied to the synaptic weight vector $\mathbf{w}(n)$ is defined by

$$\Delta\mathbf{w}(n) = -\eta\mathbf{g}(n) \qquad (4.114)$$

- Where η is the learning rate parameter.
- In effect, the *steepest-descent* method operates on the basis of a *liner approximation* of the *cost function* in the local neighborhood of the operating point $\mathbf{w}(n)$.

4.16 Supervised Learning Viewed as an Optimization Problem …

- In order to produce a significant improvement in the convergence performance of a multilayer perceptron (compared with back-propagation learning), we have to use higher-order information in the training process.
- So, we find from (4.111) that the optimum value of the adjustment $\Delta\mathbf{w}(n)$ applied to the synaptic weight vector $\mathbf{w}(n)$ is given by

$$\Delta\mathbf{w}^*(n) = \mathbf{H}^{-1}(n)\mathbf{g}(n) \qquad (4.115)$$

- where $\mathbf{H}^{-1}(n)$ is the inverse Hessian. Equation (4.115) is the essence of *Newton's method*.
- If the cost function is quadratic (i.e., the third and higher-order terms in (4.109) are zero), *Newton's method* converges to the optimum solution in one iteration.

4.16 Supervised Learning Viewed as an Optimization Problem …

- However, practical application of Newton's method to supervised training of a multilayer perceptron is handicapped by three factors
  1. Calculation of the inverse Hessian $\mathbf{H}^{-1}(n)$ required by Newton's method can be computationally expensive.
  2. For $\mathbf{H}^{-1}(n)$ to be computable, $\mathbf{H}(n)$ has to be nonsingular. Also, $\mathbf{H}(n)$ could be rank deficient (i.e. not all the columns of $\mathbf{H}$ are linearly independent).
  3. When the cost function $\mathcal{E}_{av}(\mathbf{w})$ is nonquadratic, there is no guarantee for convergence of Newton's method, which makes it unsuitable for the training of a multilayer perceptron.

4.16 Supervised Learning Viewed as an Optimization Problem
Conjugate-Gradient Method

- Conjugate-gradient method belongs to a class of *second-order optimization methods* known as *conjugate-direction methods*.
- It is based on the minimization of the quadratic function

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + \mathrm{c} \tag{4.116}$$

- where $\mathbf{x}$ is a W-by-1 parameter vector; $\mathbf{A}$ is a W-by-W symmetric, positive-definite matrix; $\mathbf{b}$ is a W-by-1 vector; and c is a scalar.

4.16 Supervised Learning Viewed as an Optimization Problem
Conjugate-Gradient Method

- Minimizing the quadratic function $f(\mathbf{x})$ is achieved by assigning to $\mathbf{x}$ the unique value

$$\mathbf{x}^* = \mathbf{A}^{-1}\mathbf{b} \tag{4.117}$$

- Thus, minimizing $f(\mathbf{x})$ and solving the linear system of equations $\mathbf{A}\mathbf{x}^* = \mathbf{b}$ are equivalent problems.

- Given matrix $\mathbf{A}$, the set of nonzero vectors $\mathbf{s}(0)$, $\mathbf{s}(1)$, $\mathbf{s}(W$ - 1) is $\mathbf{A}$-conjugate (i.e., noninterfering with each other in the context of matrix $\mathbf{A}$) if the following condition is satisfied:

$$\mathbf{s}^T(n)\mathbf{A}\mathbf{s}(j) = 0 \qquad \text{for all } n \text{ and j such that } n \neq j \tag{4.118}$$

Conjugate-Gradient Method
Example 1. Interpretation of A-Conjugate Vectors

- For interpretation of **A**-conjugate vectors, consider the situation described in Fig. 4.20a, associated with a two-dimensional elliptic locus problem shown in this figure corresponding to a plot of (4.116) for $\mathbf{x} = [x_0, x_1]^T$.
- Figure 4.20a also includes a pair of direction vectors that are conjugate with respect to the matrix **A**.
- Suppose that we define a new parameter vector **v** related to **x** by the transformation $\mathbf{v} = \mathbf{A}^{1/2}\mathbf{x}$.
- The elliptic locus of Fig. 4.20a is transformed into circular locus, as shown in Fig. 4.20b.
- Correspondingly, the pair of **A**-conjugate direction vectors in Fig. 4.20a is transformed into a pair of orthogonal direction vectors in Fig. 4.20b.
- An important property of A-conjugate vectors is that they are linearly independent. We prove this property by contradiction.

Conjugate-Gradient Method

Example 1. Interpretation of A-Conjugate Vectors

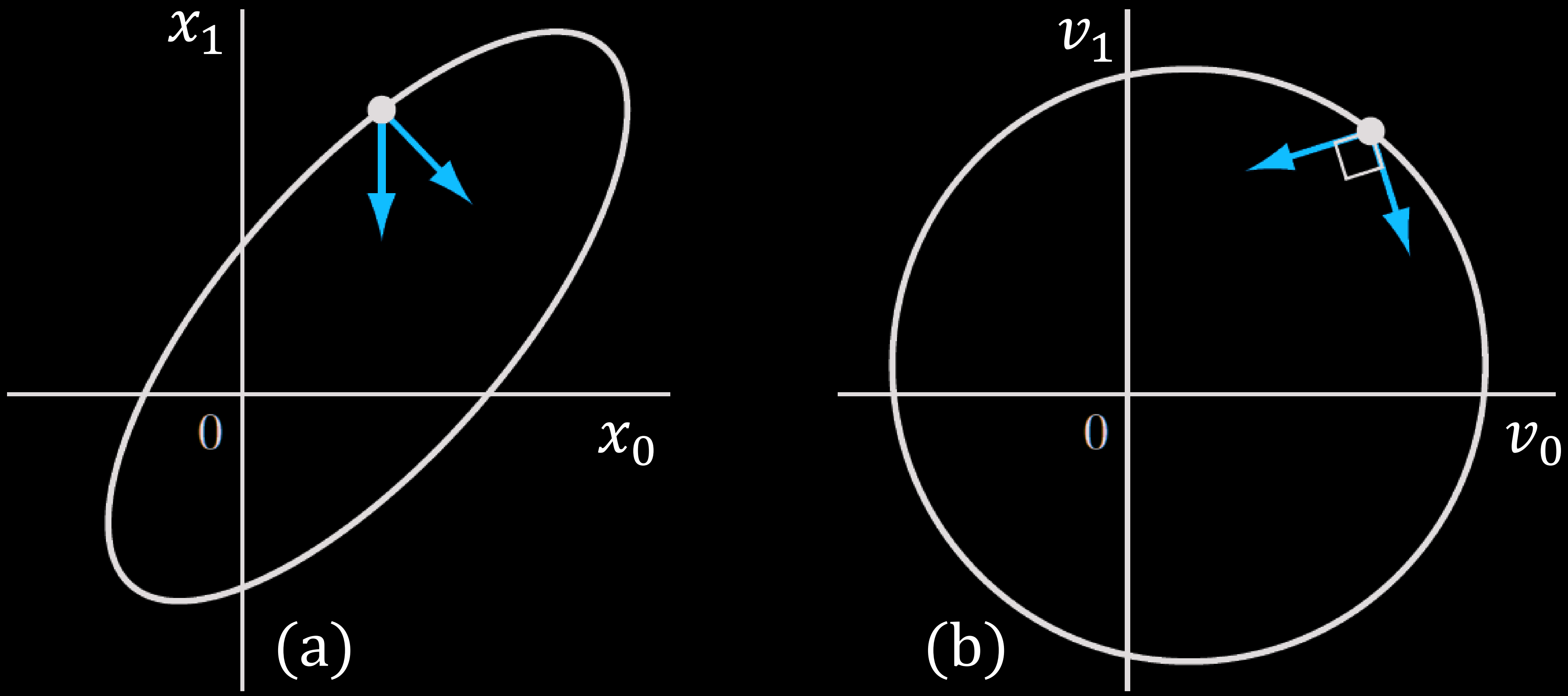


Fig. 4.20 Interpretation of A-conjugate vectors. (a) Elliptic locus in two-dimensional weight space. (b) Transformation of the elliptic locus into a circular locus.

Conjugate-Gradient Method
Solution 1. Interpretation of A-Conjugate Vectors

- Let one of these vectors, say $\mathbf{s}(0)$, be expressed as a linear combination of the remaining W - 1 vectors as follows:

$$\mathbf{s}(0) = \sum_{j=1}^{W-1} \alpha_j \mathbf{s}(j)$$

- Multiplying by $\mathbf{A}$ and then taking the inner product of $\mathbf{As}(0)$ with $\mathbf{s}(0)$ yields

$$\mathbf{s}^T(0)\mathbf{As}(0) = \sum_{j=1}^{W-1} \alpha_j \mathbf{s}^T(0)\mathbf{As}(j) = 0$$

- However, it is impossible for quadratic form $\mathbf{s}^T(0)\mathbf{As}(0)$ to be zero, for two reasons: Matrix $\mathbf{A}$ is positive definite by assumption, and the vector $\mathbf{s}(0)$ is nonzero by definition. It follows that the $\mathbf{A}$-conjugate vectors $\mathbf{s}(0)$, $\mathbf{s}(1)$, $\mathbf{s}(W$ - 1) cannot be linearly dependent; that is, they must be linearly independent.

Conjugate-Gradient Method
Solution 1. Interpretation of A-Conjugate Vectors

- For a given set of **A**-conjugate vectors $\mathbf{s}(0)$, $\mathbf{s}(1)$, $\mathbf{s}(W - 1)$, the corresponding conjugate-direction method for unconstrained minimization of the quadratic error function $f(\mathbf{x})$ is defined by

$$\mathbf{x}(n+1) = \mathbf{x}(n) + \eta(n)\mathbf{s}(n), \qquad n = 0, 1, \dots, W-1 \qquad (4.119)$$

- where $\mathbf{x}(0)$ is an arbitrary starting vector and $\eta(n)$ is a scalar defined by

$$f\big(\mathbf{x}(n)\big) + \eta(n)\mathbf{s}(n) = \min_{\eta} f\big(\mathbf{x}(n) + \eta\mathbf{s}(n)\big) \qquad (4.120)$$

- The procedure of choosing η so as to minimize the function $f\big(\mathbf{x}(n) + \eta\mathbf{s}(n)\big)$ for some fixed n is referred to as a *line search*, which represents a one-dimensional minimization problem.

Conjugate-Gradient Method

- The principal property of the conjugate-direction method is described in the following statement:

At successive iterations, the conjugate-direction method minimizes the quadratic function $f(\mathbf{x})$ over a progressively expanding linear vector space that eventually includes the global minimum of $f(\mathbf{x})$.

- In particular, for each iteration n, the iterate $\mathbf{x}(n+1)$ minimizes the function $f(\mathbf{x})$ over a linear vector space that passes through some arbitrary point $\mathbf{x}(0)$ and is spanned by the $\mathbf{A}$-conjugate vectors $\mathbf{s}(0)$, $\mathbf{s}(1)$, $\mathbf{s}(n)$, as shown by

$$\mathbf{x}(n+1) = \min_{x \in \mathcal{D}_n} f(\mathbf{x}) \tag{4.123}$$

Where the space $\mathcal{D}_n$ is defined by

$$\mathcal{D}_n = \left\{ \mathbf{x}(n) | \mathbf{x}(n) = \mathbf{x}(0) + \sum_{j=1}^{W-1} \eta(j)\mathbf{s}(j) \right\} \tag{4.124}$$

4.16 Supervised Learning Viewed as an Optimization Problem

Conjugate-Gradient Method

- Except for n $n = 0$, the set of direction vectors $\{\mathbf{s}(n)\}$ is not specified beforehand, but rather it is determined in a sequential manner at successive steps of the method.
- First, we define the *residual* as the steepest-descent direction:

$$\mathbf{r}(n) = \mathbf{b} - \mathbf{A}\mathbf{x}(n) \tag{4.125}$$

- Then, to proceed, use a linear combination of $\mathbf{r}(n)$ and $\mathbf{s}(n - 1)$, as shown by

$$\mathbf{s}(n) = \mathbf{r}(n) + \beta(n)\mathbf{s}\,(n-1),\ \text{n=1, 2, ... , } W\text{-1.} \tag{4.126}$$

- Where $\beta(n)$ is the scaling factor. Solving and rearranging, we get

$$\beta(n) = -\frac{\mathbf{s}^T(n-1)\mathbf{A}\mathbf{r}(n)}{\mathbf{s}^T(n-1)\mathbf{A}\mathbf{s}(n-1)} \tag{4.127}$$

- Hence, $\mathbf{s}(0)$, $\mathbf{s}(1)$, ... , $\mathbf{s}(W\text{-}1)$ so generated are $\mathbf{A}$-conjugate.

Conjugate-Gradient Method

- For computational reasons, it would be desirable to evaluate $\beta(n)$ without explicit knowledge of $\mathbf{A}$. This evaluation can be achieved by using one of two formulas:

1. The Polak–Ribiere formula, for which $\beta(n)$ is defined by

$$\beta(n) = \frac{\mathbf{r}^T(n)\big(\mathbf{r}(n)-\mathbf{r}(n-1)\big)}{\mathbf{r}^T(n-1)\mathbf{r}(n-1)} \qquad (4.128)$$

2. the Fletcher–Reeves formula, for which $\beta(n)$ is defined by

$$\beta(n) = \frac{\mathbf{r}^T(n)\mathbf{r}(n)}{\mathbf{r}^T(n-1)\mathbf{r}(n-1)} \qquad (4.129)$$

Conjugate-Gradient Method

- To use the conjugate-gradient method to attack the unconstrained minimization of the cost function $\mathcal{E}_{av}(\mathbf{w})$ associated with the unsupervised training of multilayer perceptron,
  - Approximate the cost function by a quadratic function by eliminating the third and higher-order terms in Eq. (4.111)
  - which means operating close to a local minimum on the error surface.
  - Hence, we can make the associations indicated in Table 4.2.

Table 4.2 Correspondence Between $f(x)$ and $\mathcal{E}_{av}(\mathbf{w})$

Quadratic function $f(\mathbf{x})$	Cost function $\mathcal{E}_{av}(\mathbf{w})$
Parameter vector $\mathbf{x}(n)$ Gradient vector $\partial f(\mathbf{x})/\partial \mathbf{x}$ Matrix $\mathbf{A}$	Synaptic weight vector $\mathbf{w}(n)$ Gradient vector $\partial \mathcal{E}_{av}(\mathbf{w})/\partial \mathbf{w}$ Hessian matrix $\mathbf{H}$

4.16 Supervised Learning Viewed as an Optimization Problem

Conjugate-Gradient Method

- Consider next the issue of computing the parameter $\eta(n)$, which determines the *learning rate* of the conjugate-gradient algorithm.
- As with $\beta(n)$, the preferred method for computing $\eta(n)$ is one that avoids having to use the Hessian $\mathbf{H}(n)$.
- We recall that the *line minimization* based on (4.120) leads to the same formula for $\eta(n)$ as that derived from the update equation (4.119).
- We therefore need a line search, the purpose of which is to minimize the function $\mathcal{E}_{av}(\mathbf{w} + \eta\mathbf{s})$ with respect to η.
    - That is, given fixed values of the vectors **w** and **s**, the problem is to vary η such that this function is minimized.
    - As η varies, the argument w $\mathbf{w} + \eta\mathbf{s}$ traces a line in the W-dimensional vector space of **w** - hence the name "line search."

Conjugate-Gradient Method

- A *line-search algorithm* is an iterative procedure that generates a sequence of estimates $\{\eta(n)\}$ for each iteration of the conjugate gradient algorithm.
- The line search is terminated when a satisfactory solution is found.
- The computation of a line search must be performed along each search direction.
- There are two phases to any line-search algorithm
  - *the bracketing phase*, which searches for a *bracket* (that is, a nontrivial interval that is known to contain a minimum), and
  - *the sectioning phase*, in which the bracket is *sectioned* (i.e., divided), thereby generating a sequence of brackets whose length is progressively reduced.

4.16 Supervised Learning Viewed as an Optimization Problem

Conjugate-Gradient Method

- To describe a *curve-fitting procedure* that takes care of these two phases in a
- straightforward manner, consider the cost function of the multilayer perceptron $\mathcal{E}_{av}(\eta)$, expressed as a function of η.
- It is assumed that $\mathcal{E}_{av}(\eta)$ is strictly *unimodal* (i.e., it has a single minimum in the neighborhood of the current point $\mathbf{w}(n)$) and is twice continuously differentiable.
- The search procedure is initiated by searching along the line until three points η_1, η_2 and η_3 are found and the following condition is satisfied, as illustrated in Fig. 4.21:

$$\mathcal{E}_{av}(\eta_1) \geq \mathcal{E}_{av}(\eta_3) \geq \mathcal{E}_{av}(\eta_2) \quad \text{for } \eta_1 < \eta_2 < \eta_3 \tag{4.131}$$

- (4.131) ensures that the bracket $[\eta_1, \eta_3]$ contains a minimum of the function.

Conjugate-Gradient Method

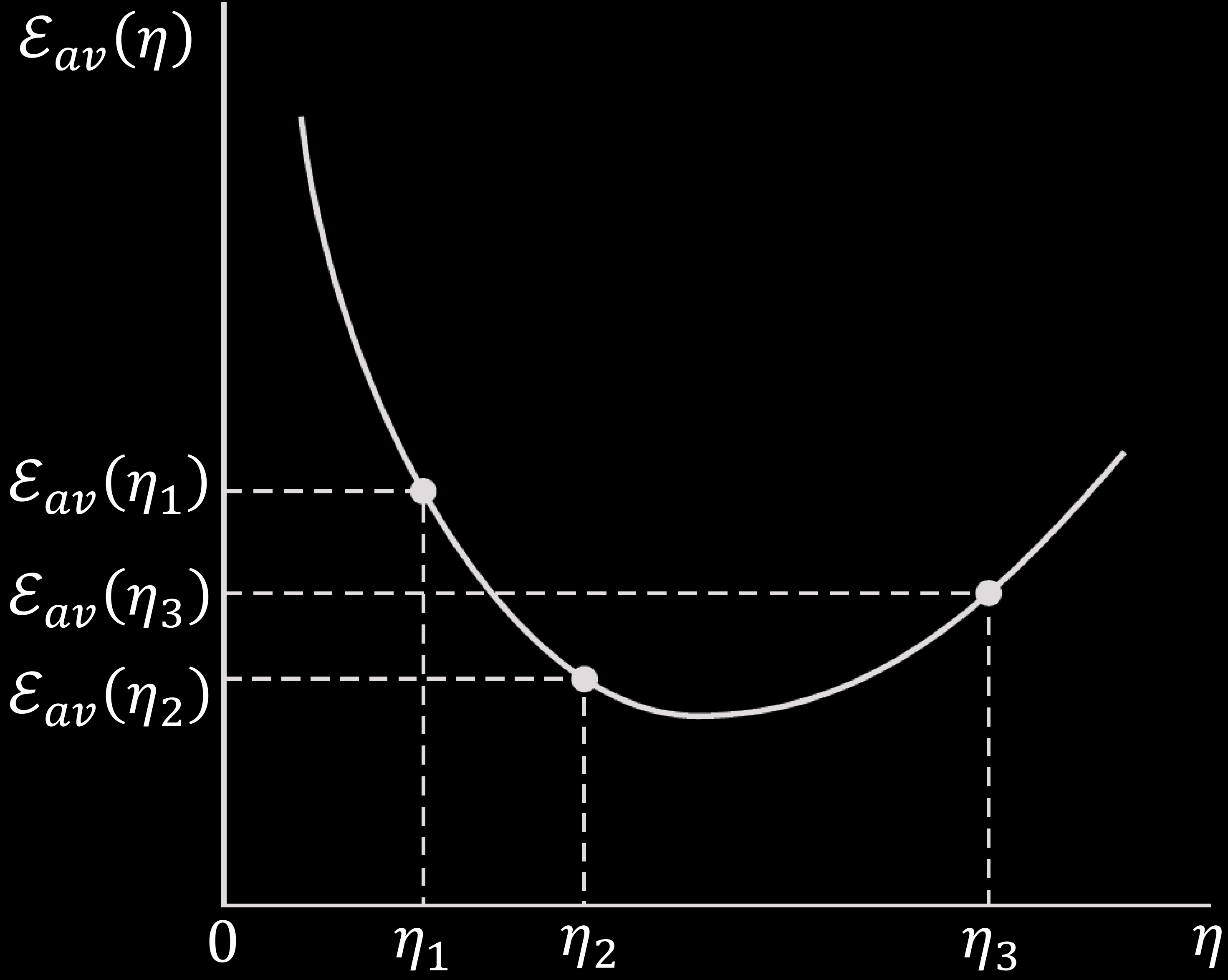


Fig. 4.21 Illustration of the line search.

4.16 Supervised Learning Viewed as an Optimization Problem

Conjugate-Gradient Method

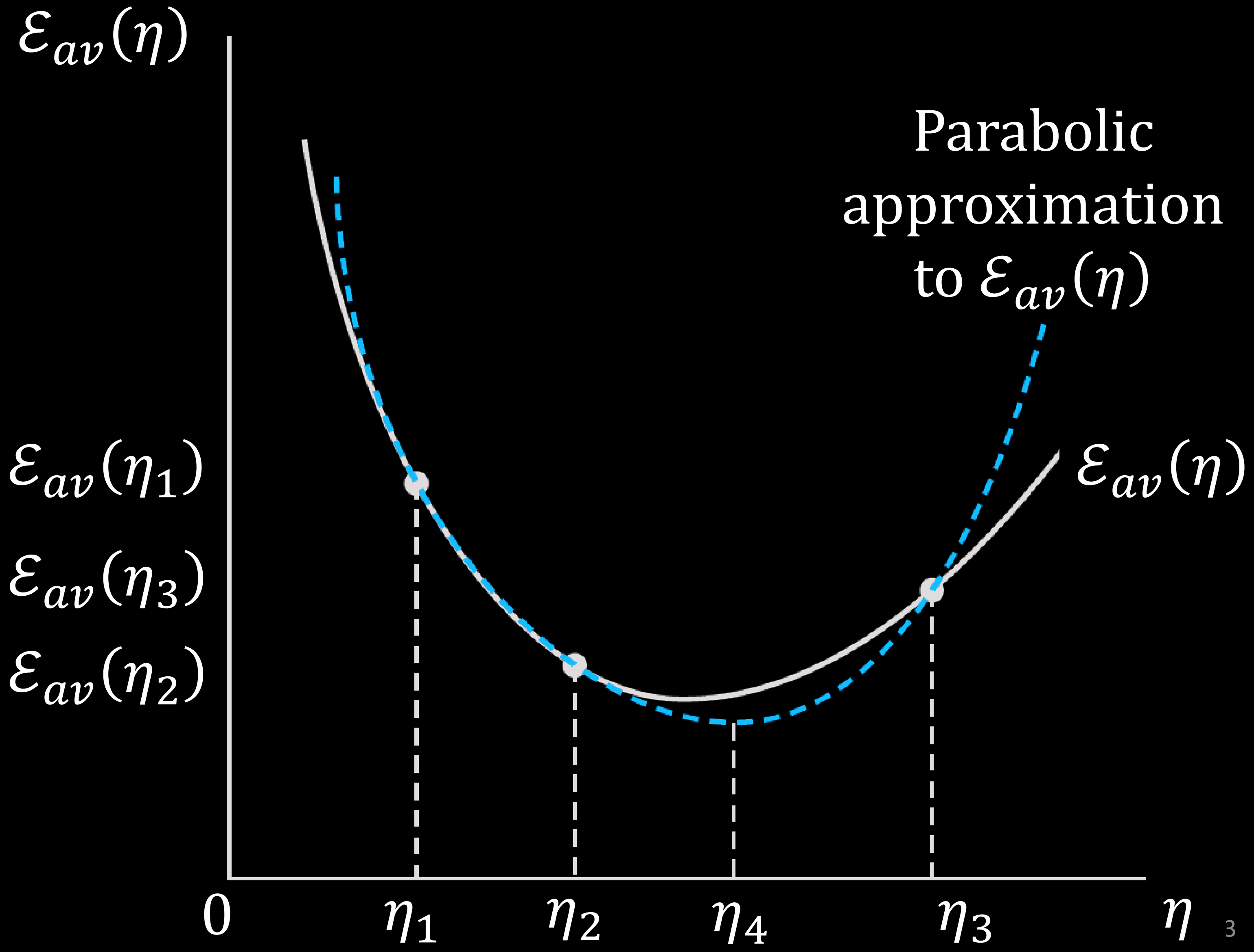


Fig. 4.22 Inverse parabolic interpolation.

4.16 Supervised Learning Viewed as an Optimization Problem
Conjugate-Gradient Method

- In the example illustrated in Fig. 4.22, point η_3 is replaced in favor of η_4, making $[\eta_1, \eta_4]$ the new bracket.
- The process is repeated several times until a point close enough to the minimum of $\mathcal{E}_{av}(\eta)$ is located by constructing a new parabola through the points η_1, η_2 and η_4.
- A summary of the algorithm is presented in Table 4.3.

TABLE 4.3 Summary of the Nonlinear Conjugate-Gradient Algorithm for the Supervised Training of a Multilayer Perceptron

Initialization

Unless prior knowledge on the weight vector w is available, choose the initial value $\mathbf{w}(0)$ by using a procedure similar to that described for the back-propagation algorithm.

Computation

1. For $\mathbf{w}(0)$, use back propagation to compute the gradient vector $\mathbf{g}(0)$.
2. Set $\mathbf{s}(0) = \mathbf{r}(0) = -\mathbf{g}(0)$.
3. At time-step n, use a line search to find $\eta(n)$ that minimizes $\mathcal{E}_{av}(\eta)$ sufficiently, representing the cost function $\mathcal{E}_{av}(\eta)$ expressed as a function of η for fixed values of $\mathbf{w}$ and $\mathbf{s}$.
4. Test to determine whether the Euclidean norm of the residual $\mathbf{r}(n)$ has fallen below a specified value, that is, a small fraction of the initial value $\|\mathbf{r}(0)\|$.
5. Update the weight vector: $\mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n)\mathbf{s}(n)$
6. For $\mathbf{w}(n+1)$, use back propagation to compute the updated gradient vector $\mathbf{g}(n+1)$.
7. Set $\mathbf{r}(n+1) = -\mathbf{g}(n+1)$.
8. Use the Polak–Ribière method to calculate: $\beta(n+1) = \max\left\{\frac{\mathbf{r}^T(n+1)\big(\mathbf{r}(n+1)-\mathbf{r}(n)\big)}{\mathbf{r}^T(n)\mathbf{r}(n)}\right\}$
9. Update the direction vector: $\mathbf{s}(n+1) = \mathbf{r}(n+1) + \beta(n+1)\mathbf{s}(n)$
10. Set $n = n + 1$ and go back to step 3.

Stopping criterion. Terminate the algorithm when the condition $\|\mathbf{r}(n)\| \le \epsilon\|\mathbf{r}(0)\|$ is satisfied where ϵ is prescribed small number.

4.16 Supervised Learning Viewed as an Optimization Problem

Quasi-Newton Methods

- Resuming the discussion on quasi-Newton methods, we find that these are basically gradient methods described by the update equation

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta\mathbf{s}(n) \tag{4.132}$$

- where direction vector $\mathbf{s}(n)$ is defined in terms of the gradient vector $\mathbf{g}(n)$ by

$$\mathbf{s}(n) = -\mathbf{S}(n)\mathbf{g}(n) \tag{4.133}$$

- in order to make the direction vector $\mathbf{s}(n)$ approximate the Newton direction, $-(\partial^2\mathcal{E}_{av}/\partial\mathbf{w}^2)^{-1}(\partial\mathcal{E}_{av}/\partial\mathbf{w}^2)$
- Quasi-Newton methods use second-order information about the error surface without requiring knowledge of the Hessian by using the iterations

$$\mathbf{q}(n) = \mathbf{g}(n+1) - \mathbf{g}(n) \tag{4.134}$$

and

$$\Delta\mathbf{w}(n) = \mathbf{w}(n+1) - \mathbf{w}(n) \tag{4.135}$$

4.16 Supervised Learning Viewed as an Optimization Problem

Quasi-Newton Methods

- We may then derive curvature information by using the approximate formula

$$\mathbf{q}(n) \cong \left(\frac{\partial}{\partial \mathbf{w}}\mathbf{g}(n)\right)\Delta\mathbf{w}(n) \qquad (4.136)$$

- In particular, given W linearly independent weight increments $\Delta\mathbf{w}(0), \Delta\mathbf{w}(1), \ldots, \Delta\mathbf{w}(W-1)$ and the respective gradient increments $\mathbf{q}(0)$, $\mathbf{q}(1)$, $\mathbf{q}(W$ - 1), we may approximate the Hessian as

$$H \cong [\mathbf{q}(0), \mathbf{q}(1), \ \ldots, \mathbf{q}(w-1)][\Delta\mathbf{w}(0), \Delta\mathbf{w}(1), \ldots, \Delta\mathbf{w}(W-1)]^{-1} \qquad (4.137)$$

- We may also approximate the inverse Hessian as

$$\mathbf{H}^{-1} \cong [\Delta\mathbf{w}(0), \Delta\mathbf{w}(1), \ldots, \Delta\mathbf{w}(W-1)][\mathbf{q}(0), \mathbf{q}(1), \ \ldots, \mathbf{q}(w-1)]^{-1} \qquad (4.138)$$

- When cost function $\mathcal{E}_{av}(\mathbf{w})$ is quadratic, Eqs. (4.137) and (4.138) are exact.

Comparison of Quasi-Newton Methods with Conjugate-Gradient Methods

- We conclude this brief discussion of quasi-Newton methods by comparing them with conjugate-gradient methods in the context of nonquadratic optimization problems
  - Both quasi-Newton and conjugate-gradient methods avoid the need to use the Hessian. However, quasi-Newton methods also generate an approximation to the inverse Hessian. Accordingly, when the *line search* is accurate and we are close to a local minimum, a quasi-Newton method tends to approximate Newton's method and attain faster convergence than the conjugate-gradient method.
  - Quasi-Newton methods are not as sensitive to accuracy in the line-search stage of the optimization as the conjugate-gradient method.
  - Quasi-Newton methods require storage of the matrix $\mathbf{S}(n)$, in addition to the matrix-vector multiplication associated with computation of direction vector $\mathbf{s}(n)$. When dimension W (size of weight vector w) is large, conjugate-gradient methods are preferable to quasi-Newton methods in computational terms.

4.16 Supervised Learning Viewed as an Optimization Problem

Levenberg–Marquardt Method

- The Levenberg–Marquardt method is a compromise between the following two methods:
  - ~ Newton's method, which converges rapidly near a local or global minimum, but may also diverge;
  - ~ Gradient descent, which is assured of convergence through a proper selection of the step-size parameter but converges slowly.
- Considering optimization of a second-order function $F(\mathbf{w})$, $\mathbf{g}$ be its gradient vector and $\mathbf{H}$ be its Hessian. According to the Levenberg–Marquardt method, the optimum adjustment applied to the parameter vector $\mathbf{w}$ is defined by

$$\Delta\mathbf{w} = [\mathbf{H} + \lambda\mathbf{I}]^{-1}\mathbf{g} \tag{4.142}$$

- where $\mathbf{I}$ is identity matrix of same dimensions as $\mathbf{H}$ and λ is a regularizing (loading) parameter that forces the sum matrix $(\mathbf{H} + \lambda\mathbf{I})$ to be positive definite and well conditioned.

4.16 Supervised Learning Viewed as an Optimization Problem

Levenberg–Marquardt Method

- Considering a multilayer perceptron with a single output neuron trained by minimizing the cost function

$$\mathcal{E}_{av}(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}[d(i) - F(\mathbf{x}(i);\mathbf{w})]^2 \qquad (4.143)$$

- Where $F(\mathbf{x}(i);\mathbf{w})$ is the approximating function realized by the network.

- The gradient and Hessian of the cost function are defined by

$$\mathbf{g}(\mathbf{w}) = \frac{\partial \mathcal{E}_{av}(\mathbf{w})}{\partial w} = -\frac{1}{N}\sum_{i=1}^{N}[d(i) - F(\mathbf{x}(i);\mathbf{w})]\frac{\partial F(\mathbf{x}(i);\mathbf{w})}{\partial \mathbf{w}} \qquad (4.144)$$

- and $$\mathbf{H}(\mathbf{w}) = \frac{\partial^2 \mathcal{E}_{av}(\mathbf{w})}{\partial w^2} = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{\partial F(\mathbf{x}(i);\mathbf{w})}{\partial \mathbf{w}}\right]\left[\frac{\partial F(\mathbf{x}(i);\mathbf{w})}{\partial \mathbf{w}}\right]^T$$

$$-\frac{1}{N}\sum_{i=1}^{N}[d(i) - F(\mathbf{x}(i);\mathbf{w})]\frac{\partial^2 F(\mathbf{x}(i);\mathbf{w})}{\partial^2 \mathbf{w}} \qquad (4.145)$$

4.16 Supervised Learning Viewed as an Optimization Problem

Levenberg–Marquardt Method

- To mitigate the high computational complexity of (4.145) when dimensionality of **w** is high, we ignore the second term and approximate

$$\mathbf{H}(\mathbf{w}) =\approx \frac{1}{N}\sum_{i=1}^{N}\left[\frac{\partial F(\mathbf{x}(i);\mathbf{w})}{\partial \mathbf{w}}\right]\left[\frac{\partial F(\mathbf{x}(i);\mathbf{w})}{\partial \mathbf{w}}\right]^{T} \qquad (4.146)$$

- Which is the outer product approximation of $\partial F(\mathbf{x}(i);\mathbf{w})/\partial\mathbf{w}$ averaged over the training sample. Hence, outer product approximation of the Hessian.
-

4.16 Supervised Learning Viewed as an Optimization Problem

Levenberg–Marquardt Method

- At each iteration of the algorithm, the value assigned to λ should be just large enough to maintain the sum matrix in its positive-definite form. In specific terms, the recommended Marquardt recipe for the selection of λ is as follows

1. Compute $\mathcal{E}_{av}(\mathbf{w})$ at iteration $n-1$.
2. Choose a modest value for λ, say $= 10^{-3}$.
3. Solve Eq. (4.142) for the adjustment $\Delta\mathbf{w}$ at iteration n and evaluate $\mathcal{E}_{av}(\mathbf{w}+\Delta\mathbf{w})$.
4. If $\mathcal{E}_{av}(\mathbf{w}+\Delta\mathbf{w}) \geq \mathcal{E}_{av}(\mathbf{w})$, increase λ by a factor of 10, update the trial solution $\mathbf{w} \rightarrow \mathbf{w}+\Delta\mathbf{w}$ and go back to step 3.

4.16 Supervised Learning Viewed as an Optimization Problem
Second-Order Stochastic Gradient Descent for On-line Learning

- Now we turn our attention to second-order stochastic gradient-descent methods for on-line learning which share a common purpose:

The second-order information contained in the Hessian (curvature) of the cost function is used to improve the performance of supervised-learning algorithms.

- To expand the performance of the optimally annealed on-line learning algorithm, replace the learning-rate parameter $\eta(n)$ in (4.60) with the scaled inverse of the Hessian $\mathbf{H}$ (and the precomputed $\mathbf{H}^{-1}$), as shown by

$$\underbrace{\hat{\mathbf{w}}(n+1)}_{\substack{\text{Updated}\\ \text{estimate}}} = \underbrace{\hat{\mathbf{w}}(n)}_{\substack{\text{Old}\\ \text{estimate}}} - \underbrace{\frac{1}{n}\mathbf{H}^{-1}}_{\substack{\text{Annealed}\\ \text{inverse of}\\ \text{the Hessian}}} \underbrace{\mathbf{g}\big(\mathbf{x}(n+1), \mathbf{d}(n+1); \hat{\mathbf{w}}(n)\big)}_{\substack{\text{Gradient}\\ \text{vector } \mathbf{g}}} \qquad (4.147)$$

4.16 Supervised Learning Viewed as an Optimization Problem
Second-Order Stochastic Gradient Descent for On-line Learning

- The price paid for the accelerated convergence is summarized as follows

  i) In *stochastic gradient descent* of (4.60), computation cost per iteration of the algorithm is $O(W)$ and the corresponding computation cost per iteration of second order stochastic *gradient-descent algorithm* in (4.147) is $O(W^2)$, where W is the dimension of weight vector **w** being estimated.

  ii) For each training example (**x**, **d**) processed by the algorithm of (4.147), the algorithm requires multiplication of the W-by-l gradient vector **g** and the W-by-W inverse matrix $\mathbf{H}^{-1}$ and storage of the product.

  iii) Unfortunately, the Hessian **H** is typically a full matrix and therefore there is no sparsity in the training sample. Hence there is no possibility of exploiting sparsity.

4.16 Supervised Learning Viewed as an Optimization Problem
Second-Order Stochastic Gradient Descent for On-line Learning

- To overcome these limitations, resort to one of the following approximation procedures:

  i) *Diagonal approximation*: Retain only the diagonal elements of the Hessian, so the $\mathbf{H}^{-1}$will also be a diagonal and the product $\mathbf{H}^{-1}\mathbf{g}$ will consist of sum of terms of the form $h_{ii}^{-1} g_i$ where h_{ii} are diagonal elements of $\mathbf{H}$ and g_i is corresponding element of gradient $\mathbf{g}$ for $i = 1, 2, \ldots, W$. Since $\mathbf{g}$ is linear in the weights, computational complexity of the approximated second-order online learning algorithm is $O(W)$.

$$\begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1i} \\ h_{21} & h_{22} & & h_{2i} \\ \vdots & & \ddots & \vdots \\ h_{i1} & h_{i2} & \cdots & h_{ii} \end{bmatrix} [g_1 \; g_2 \; \cdots g_i]$$

4.16 Supervised Learning Viewed as an Optimization Problem

Second-Order Stochastic Gradient Descent for On-line Learning

ii) *Low-rank approximation*: The rank of a matrix equals the number of algebraically independent columns of the matrix. Given a Hessian **H**, the use of singular value decomposition (SVD) provides an important procedure for the low-rank approximation of **H**. Let the rank of **H** be denoted by p and a rank r approximation of **H** be denoted by $\mathbf{H}_r$, where $r < p$. The squared error between the Hessian and its approximation is defined by the *Frobenius norm*

$$e^2 = \mathrm{tr}[(\mathbf{H} - \mathbf{H}_r)^T(\mathbf{H} - \mathbf{H}_r)] \qquad (4.148)$$

Where tr[.] is the trace (i.e. Sum of diagonal components) of square matrix enclosed inside the square brackets.

Second-Order Stochastic Gradient Descent for On-line Learning

ii) Applying the SVD to the matrices $\mathbf{H}$ and $\mathbf{H}_r$, we write

$$\mathbf{H} = \mathbf{V\Sigma U}^{\mathrm{T}} \tag{4.149}$$

and

$$\mathbf{H}_r = \mathbf{V\Sigma}_r\mathbf{U}^{\mathrm{T}} \tag{4.150}$$

where the orthogonal matrices $\mathbf{U}$ and $\mathbf{V}$ define the common *right* and *left singular vectors*, respectively, and the rectangular matrix

$$\mathbf{\Sigma}_r = \mathrm{diag}[\lambda_1, \lambda_2, \dots, \lambda_r, 0 \;\dots, 0] \tag{4.151}$$

defines the *singular values* of the low-rank approximation $\mathbf{H}_r$. The new square matrix

$$\mathbf{H}_r = \mathbf{U\Sigma}_r\mathbf{V}^{\mathrm{T}} \tag{4.152}$$

provides the *least-squares, rank r approximation* to the Hessian $\mathbf{H}$. Correspondingly, the new matrix $\mathbf{H}_r$ in place of the Hessian $\mathbf{H}$ in the on-line learning algorithm of (4.147) reduces the computational complexity of between $O(W)$ and $O(W^2)$.

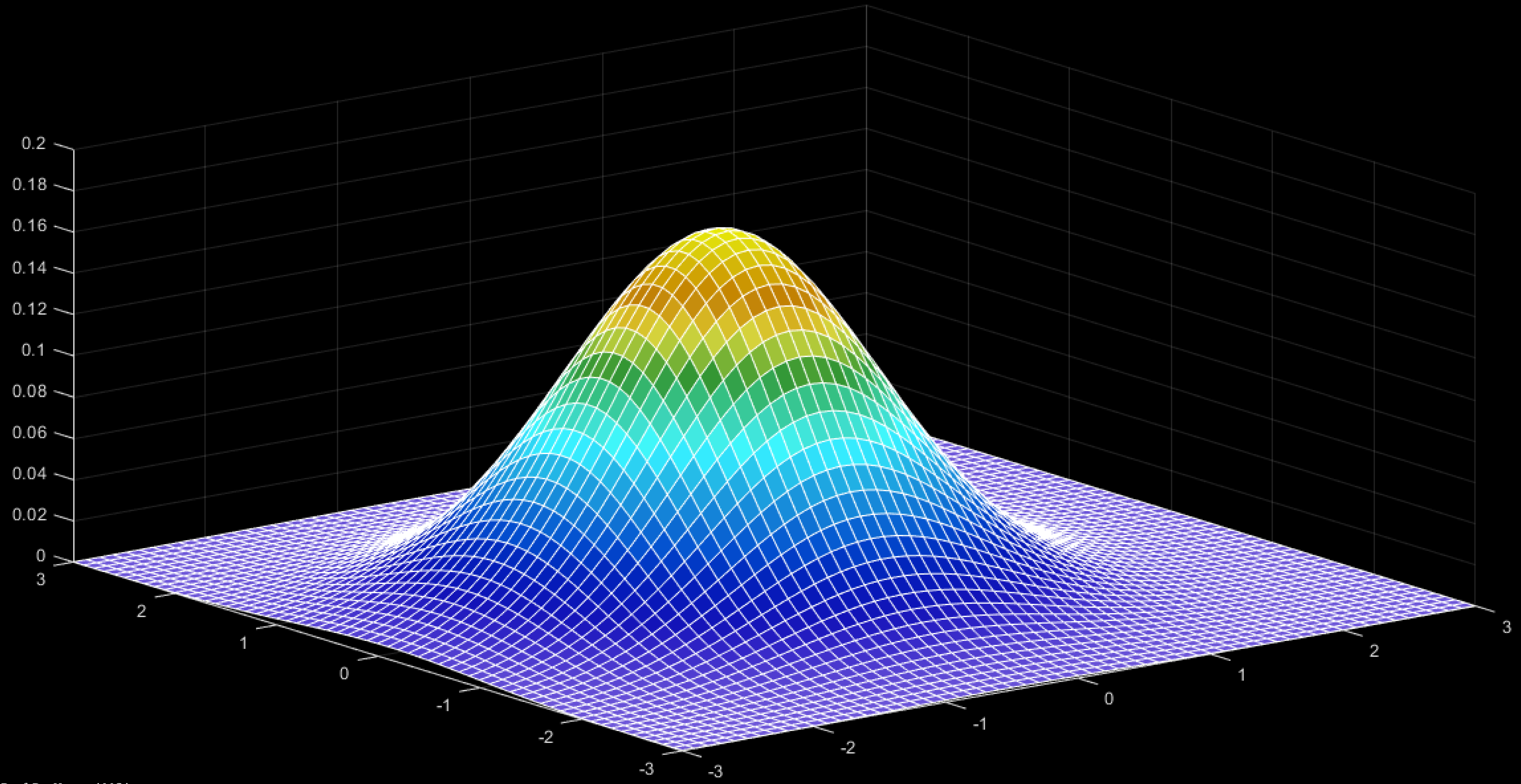
4.16 Supervised Learning Viewed as an Optimization Problem

Second-Order Stochastic Gradient Descent for On-line Learning

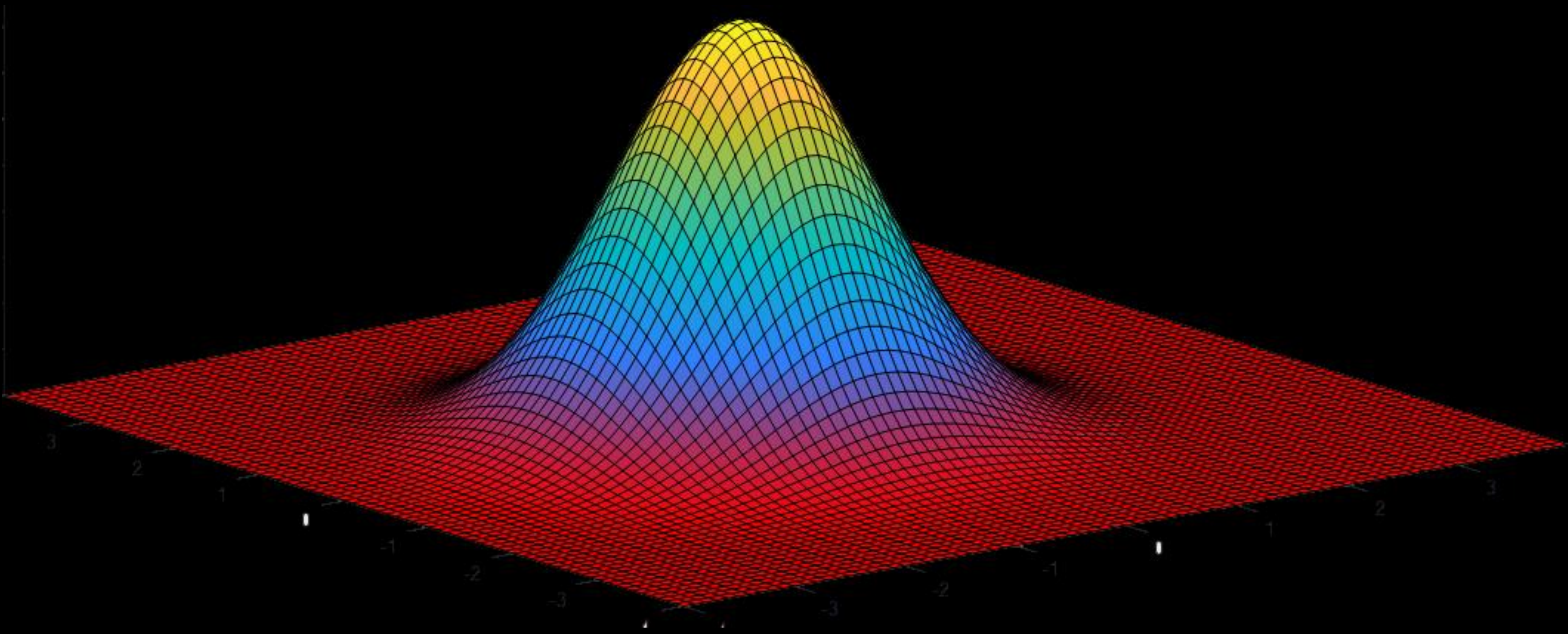
iii) *BFGS approximation*: The BFGS algorithm is considered to be the best form of a quasi-Newton method. The algorithm appears to provide a fast, scalable, stochastic quasi-Newton procedure for on-line convex optimization. The BFGS quasi-Newton method and its limited-memory variant are also extended to deal with non-smooth convex objective functions.

End of Section 4.x

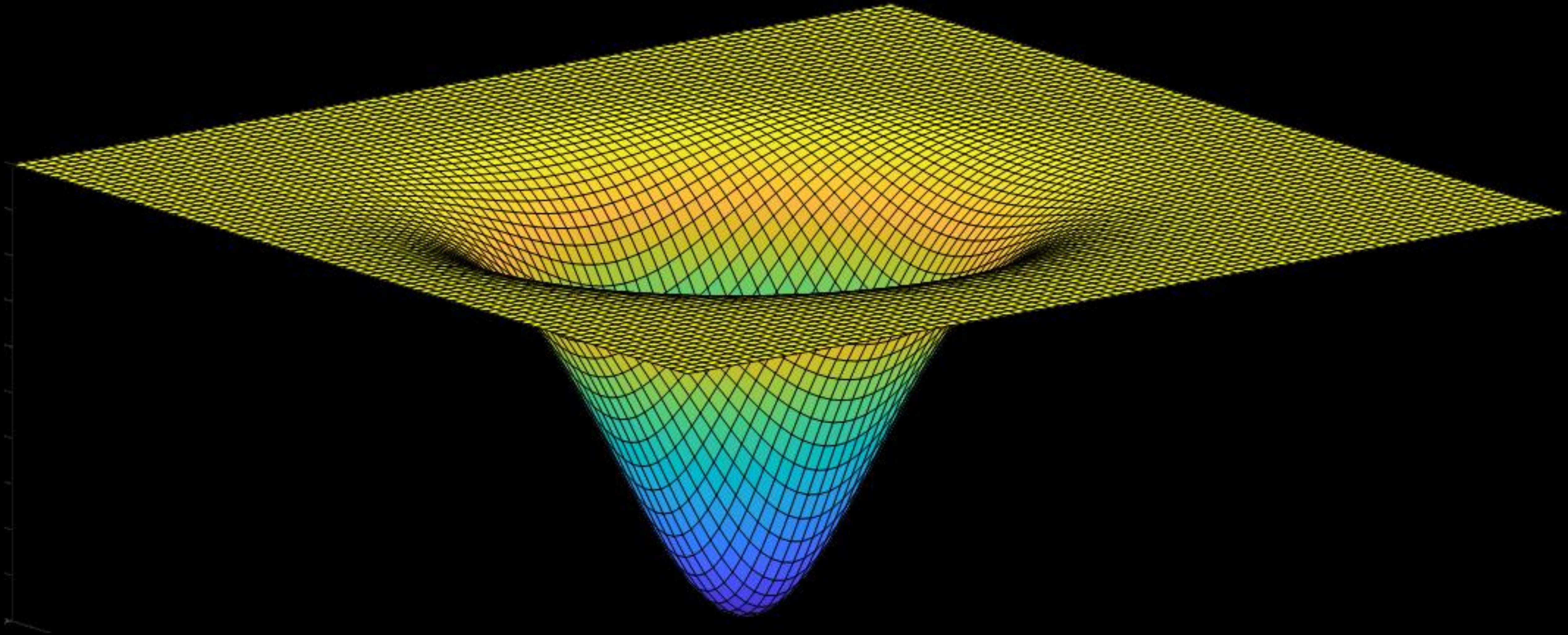
Multivariate Gaussian PDF Plot



Multivariate Gaussian PDF Plot



Multivariate Gaussian PDF Plot



Tips and Tricks:
Getting Started Using Optimization with MATLAB

https://www.mathworks.com/videos/tips-and-tricks-getting-started-using-optimization-with-matlab-81594.html?s_iid=doc_rw_GD_footer