Neural Networks

References

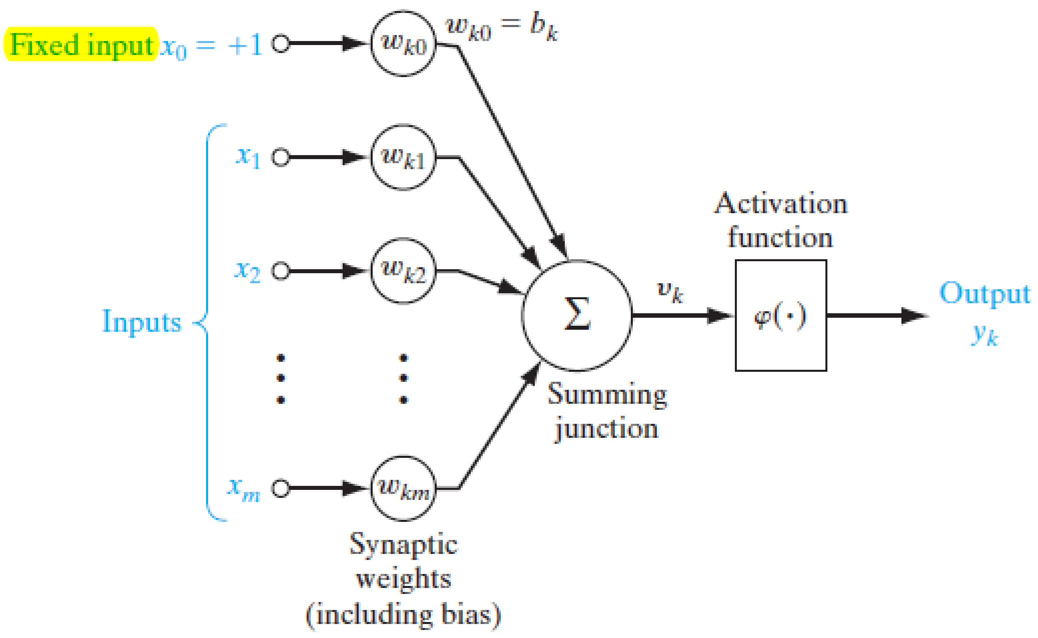
- Haykin, S., *Neural Networks and Learning Machines*, Pearson Education, Upper Saddle River, NJ, 2009.

Introduction

3. Models of a Neuron 10

Three basic elements

- Block diagram of a neuron



1. A set of *synapses* (or connecting links) is characterized by a *weight* (or strength) of its own. A signal $x_j$ at the input of synapse $j$ connected to neuron $k$ is multiplied by the synaptic weight $w_{ji}$.

2. An *adder* for summing the input signals weighted by the respective synaptic strengths of the neuron to constitute a *linear combiner*.

3. An *activation function* (also referred to as a squashing function) for limiting the amplitude of the output of a neuron.

   $$ u_k = \sum^m_{j=1} w_{kj}x_j $$

   $$ v_k = u_k + b_k = \sum^m_{j=1} w_{kj}x_j + w_{k0} = \sum^m_{j=0} w_{kj}x_j $$

   where $x_0 = 1$ and $w_{k0} = b_k$.

   $$ y_k = \varphi(v_k) = \varphi\left(\sum^m_{j=0} w_{kj}x_j\right) $$

- Components

- x_j = input signal
- u_k = linear combiner output
- b_k = bias
- v_k = induced local field, or activation potential
- $\varphi(\cdot)$ = activation function
- y_k = output signal

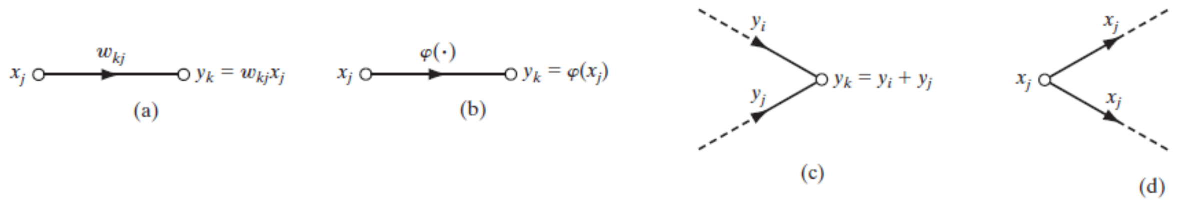
Types of activation function

- Threshold function
- Sigmoid function
- Hyperbolic tangent

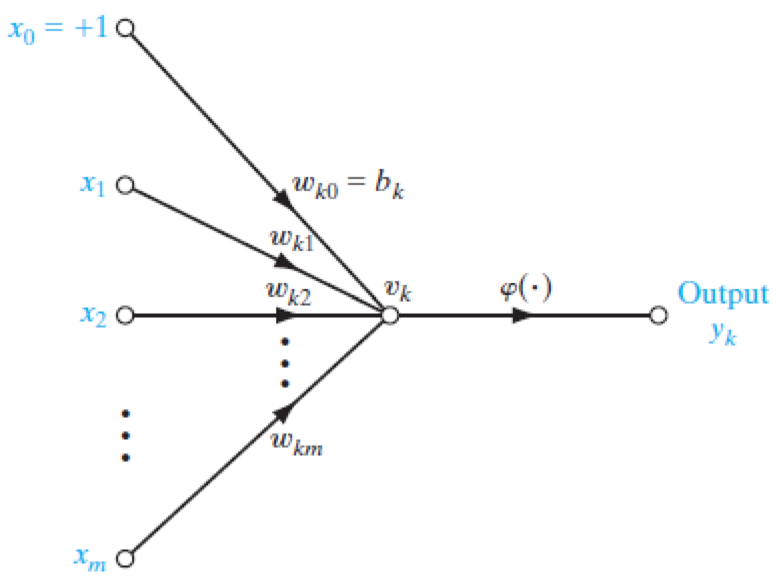
4. Neural Networks Viewed As Directed Graphs 15

Three representations of neural networks

1. Block diagram
   - a functional description of the various elements that constitute the model of an artificial neuron
2. Signal-flow graph
   - a network of *directed links (branches)* that are interconnected at certain points called *nodes*.
   - A *directed link* originates at node j and terminates on node k. It has an associated *transfer function*, or *transmittance*, that specifies the manner in which the signal y_k at node k depends on the signal x_j at node j.

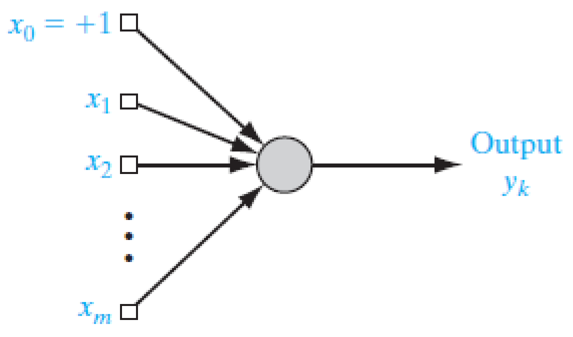


     - (a) Synaptic links
     - (b) Activation links
     - (c) Algebraic sum, or fan-in
     - (d) Synaptic divergence, or fan-out
   - Signal-flow graph of a neuron



3. Architectural graph
   A reduced form of the signal-flow graph

   - Three components
     - Source node
     - Computation node
     - communication link (this carries no weight.)
   - Architectural graph of a neuron



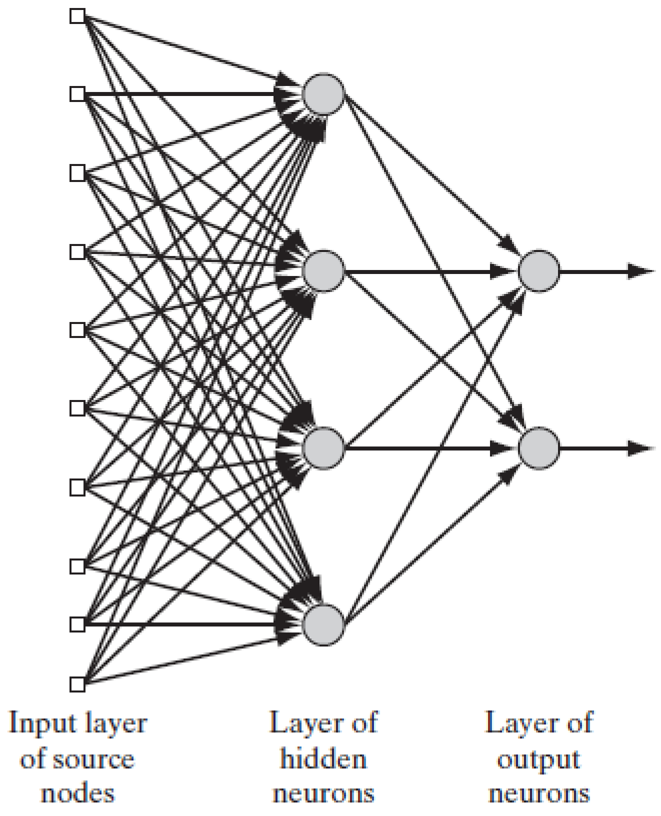
- Summary
  - Block diagram, providing a functional description of the network;
  - Signal-flow graph, providing a complete description of signal flow in the network;
  - Architectural graph, describing the network layout.

5. Feedback 18

- Feedback exists in a *dynamic system* whenever the output of an element in the system influences in part the input applied to that particular element, thereby giving rise to *one or more closed paths* for the transmission of signals around the system.
- Example
  - Recurrent networks

6. Network Architectures 21

Multilayer feedforward networks



- Layers
  - Input layer of source nodes
  - hidden layers of hidden neurons (computation nodes)
  - Output layer of neurons (computation nodes)
- Network types:
  A feedforward network with m source nodes, h_1 neurons in the first hidden layer, h_2 neurons in the second hidden layer, and q neurons in the output layer is referred to as an $m–h_1–h_2–q$ network.

Recurrent networks

- A *recurrent neural network* distinguishes itself from a feedforward neural network in that it has at least one feedback loop.

7. Knowledge Representation 24

- A major task for a neural network is to learn a model of the **world (environment)**, and to maintain the model sufficiently consistently with the real world.
- **Knowledge** of the world consists of two kinds of information:
  - **prior information**: The known world state, represented by facts about what is and what has been. known;
  - **Observations (measurements)**: Ordinarily, these observations are **inherently noisy**. In any event, the observations so obtained provide the pool of information, from which the **examples** used to train the neural network are drawn.

Examples

- **labeled**: **input signal** is paired with a corresponding **desired response**.
- **unlabeled**:
- **positive examples**: positive examples refer to input training data that contain the target of interest.
- **negative examples**: negative examples are included purposely in the training data to teach the network not to confuse the non-target with the target.

Design procedure of neural network

- **Architecture selection**
- **Learning**: A subset of examples is used to **train** the network by means of a suitable algorithm.
- **Testing**: The performance of the trained network is *tested* with data not seen before.

Terminologies

- **Training data, or training sample**: a set of input-output pairs
- **Generalization**: Successful performance on the test pattern.

Rules of knowledge representation

- Rules about how knowledge is actually represented inside an artificial network

- **Rule 1**. Similar inputs (i.e., patterns drawn) from similar classes should usually produce similar representations inside the network, and should therefore be classified as belonging to the same class.

    - Measure of similarity ($d(x_i,x_j)$): Euclidean norm, dot product, Mahalanobis distance, etc. $$ d_{ij}^2 = (x_i-\mu_i)^TC^{-1}(x_j-\mu_j) $$ where $\mu_i = E[x_i]$. For a covariance matrix, $C = E[(x_i-\mu_i)(x_i-\mu_i)^T] = E[(x_j-\mu_j)(x_j-\mu_j)^T]$ is assumed.

- **Rule 2**. Items to be categorized as separate classes should be given widely different representations in the network.

- **Rule 3**. If a particular feature is important, then there should be **a large number of neurons** involved in the representation of that item in the network.

- **Rule 4**. **Prior information** and **invariances** should be built into the design of a neural network whenever they are available, so as to simplify the network design by its not having to learn them.

How to build prior information into neural network design

- receptive fields
- weight-sharing

How to build invariance into neural network design

- invariance by structure
- invariance by training
- invariant feature space

8. Learning Processes 34

- Learning with a teacher, or supervised learning

- Learning without a teacher
  - Reinforcement learning
  - Unsupervised learning

9. Learning Tasks 38

- Pattern association
- Pattern recognition
- Function approximation
  - System identification
  - Inverse modeling
- Control

Chapter 1 Rosenblatt’s Perceptron 47

- The perceptron is the simplest form of a neural network used for the classification of patterns said to be linearly separable (i.e., patterns that lie on opposite sides of a hyperplane).
- Basically, it consists of a *single neuron* with adjustable synaptic weights and bias.
- Rosenblatt proved that if the patterns (vectors) used to train the perceptron are drawn from two linearly separable classes, then the perceptron algorithm converges and positions the decision surface in the form of a hyperplane between the two classes. The proof of convergence of the algorithm is known as the perceptron convergence theorem.

1.2 Perceptron 48

- McCulloch-Pitts model of a neuron: threshold function is used as an activation function

1.3 Perceptron convergence theorem

- Error correction learning rule

$$ w(n+1) = w(n) + \eta*(d(n)-y(n))x(n) $$

Fixed-increment convergence theorem

- For linearly separable training vectors, the perceptron converges after some bounded $N$ iterations in the sense that

  $$ w(N) = w(N+1) = w(N+2) = \cdots $$

  is a solution vector for $n = N$.

Chapter 2 Model Building through Regression 68

2.1 Introduction 68

- Regression is a special form of function approximation.
- Regression

- response: dependent random variables
- regressor: independent random variables
- expectational error, or explanational error: error in relation between the regressor and response.

2.2 Linear Regression Model: Preliminary Considerations 69

- Model
  - inputs: $x = [x_1,x_2,\cdots,x_M]^T$ is probed from unknown stochastic environment.
  - parameters: $w = [w_1,w_2,\cdots,w_M]^T$ is fixed, unknown parameters. -> stationary environment.
  - model order M
  - linear regression model

    $$ d = w^T x + \epsilon $$
- Problem statement
  - Given the **joint statistics** of the regressor X and the corresponding response D, **estimate** the unknown parameter vector w.
- Statistical parameters
  - the correlation matrix of the regressor X;
  - the variance of the desired response D;
  - the cross-correlation vector of the regressor X and the desired response D.
- Likelihood function

  $$ l(w|d,x) = p_{D|W,X}(d|w,x) $$
- Prior

  $$ \pi (w) $$
- Posterior density

  $$ \pi(w|d,x) = p_{D|W,X}(d|w,x)p_W (w) = l(w|d,x)\pi(w) $$

2.3 Maximum a Posteriori Estimation of the Parameter Vector 71

- Assumption
  - N examples are *iid*
  - The environment responsible for generating the training sample is **Gaussian distributed**.
- Maximum likelihood estimation

  $$ E (w) = \frac{1}{2} \sum_{i=1}^N (d_i - w^T x_i)^2 $$
- Maximum *a posteriori* estimation

$$ E (w) = \frac{1}{2} \sum_{i=1}^N (d_i - w^T x_i)^2 + \frac{\lambda}{2} ||w||^2 $$

2.4 Relationship Between Regularized Least-Squares Estimation and MAP Estimation 76

2.6 The Minimum-Description-Length Principle 79

2.7 Finite Sample-Size Considerations 82

2.8 The Instrumental-Variables Method 86

Chapter 3 The Least-Mean-Square Algorithm 91

- LMS is an adaptive filtering algorithm
- LMS is used for a **single linear neuron** (MISO).
- Advantages of LMS algorithm
    - complexity: linear w.r.t. free parameters (efficient)
    - Simple
    - robust against external disturbances

3.2 Filtering Structure of the LMS Algorithm 92

- Adaptive filter processes
    - Filtering process: error computation $e(i) = d(i) - y(i)$, i = temporal index
    - Adaptive procecss: weight adjustment

3.3 Unconstrained Optimization: a Review 94

- Necessary condition: zero gradient
- In the design of adaptive filters: iterative descent

Steepest descent

- Gradient: $\triangledown_w E(w)$

- Correction

  $$ \Delta w(n) = -\eta g(n) $$

  where $\eta > 0$.

- Slow

- Highly dependent of choise of $\eta$.

    - overdamped
    - underdamped (zigzag)
    - unstable

Newton method

- Quadratic approximation of the cost function about the current point $w(n)$.
- Correction

  $$ \Delta w(n) = - H^{-1}(n) g(n) $$
- Converges quickly asymptotically
- H must be p.d.
- Complex

Gauss-Newton method

- Similar to Newton method, but does not require Hessian but only Jacobian.
- Correction

  $$ \Delta w(n) = - (J^T(n)J(n))^{-1} J^T(n) e(n) $$
- J should have **full row rank**
- A regularization factor can be added to force full row rank of J: **diagonal loading**

3.4 The Wiener Filter 100

- Form a cumulative desired response vector d and input matrix X as new data is measured.
- Correction

  $$ w(n+1) = X^+(n) d(n) $$
- As $n \rightarrow \infty$, $w(n) \rightarrow w^*$. Wiener solution.

3.5 The Least-Mean-Square Algorithm 102

- Update

  $$\hat{w} (n+1) = \hat{w} + \eta x(n) e(n)$$
- produces an instantaneous estimate of the weight vector
- **stochastic gradient algorithm**
- starts from zero initial estimation

3.6 Markov Model Portraying the Deviation of the LMS Algorithm from the Wiener Filter 104

- For small learning rate η, $w(n)$ becomes sufficient close to w_o.

3.7 The Langevin Equation: Characterization of Brownian Motion 106

3.8 Kushner’s Direct-Averaging Method 107

3.9 Statistical LMS Learning Theory for Small Learning-Rate Parameter 108

3.12 Virtues and Limitations of the LMS Algorithm 113

3.13 Learning-Rate Annealing Schedules 115

Chapter 4 Multilayer Perceptrons 122

4.1 Introduction 123

The basic features of MLP

- The model of each neuron in the network includes a nonlinear activation function that is *differentiable*.
- The network contains *one or more layers that are hidden* from both the input and output nodes.
- The network exhibits *a high degree of connectivity*, the extent of which is determined by synaptic weights of the network.

Back-propagation

- The training proceeds in two phases:
  - **Forward phase**: the synaptic weights of the network are fixed and the input signal is propagated through the network, layer by layer, until it reaches the output.
  - **Backward phase**: an error signal is produced by comparing the output of the network with a desired response.The resulting **error signal** is propagated through the network, again layer by layer, but this time the propagation is performed in the backward direction. In this second phase, successive adjustments are made to the synaptic weights of the network.

4.2 Some Preliminaries 124

Signals in neural networks

- **Function signals**: An input signal (stimulus) that propagates forward (neuron by neuron).
- **Error signals**: This originates at an output neuron and propagates backward (layer by layer).

Function of the hidden neurons

- The hidden neurons act as **feature detectors**.
  - They performance a nonlinear transformation on the input data into a **feature space**.

Credit assignment problem

- In a multilayer perceptron using *error-correlation learning*, the credit-assignment problem arises because the operation **of each hidden neuron and of each output neuron** in the network is important to the network's correct overall action on a learning task of interest.

4.3 Batch Learning and On-Line Learning 126

Terms

- Error signal produced at the output neuron j

  $$ e_j (n) = d_j (n) - y_j (n) $$

- Instantaneous error energy

  $$ E_j(n) = \frac{1}{2} e_j (n) ^2 $$

- Total instantaneous error energy

  $$ E(n) = \frac{1}{2} \sum_{j\in C} e_j (n) ^2 $$

- Error energy averaged over the training sample (of N examples), or empirical risk

  $$ E_a(N) = \frac{1}{2N} \sum_{n = 1}^N \sum_{j\in C} e_j (n) ^2 $$

Batch learning

- Weight adjustment is performed after all N examples that constitute one epoch in the training sample using **empirical risk** as a cost function. (epoch by epoch)
- Advantages
  - accurate estimation of the **gradient**
  - **parallelization** of learning process
- Disadvantages
  - Storage requirement

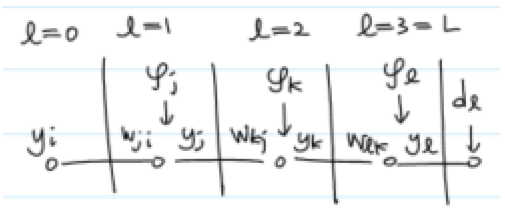
On-line learning

- Weight adjustment is performed on each example of training sample using **total instantaneous error energy** as a cost function. (example by example)
- On-line learning is referred to as a **stochastic method**.
- Training examples are randomly **shuffled** after each epoch.
- Advantages
  - It is less likely for the learning process to be trapped in a local minimum.
  - It requires much less storage.
  - It is able to take advantage of the redundancy of training data.
  - Ability to track small changes in the training data

4.4 The Back-Propagation Algorithm 129

- On-line learning is enhanced by the development of the back-propagation.
- BP efficiently computes the gradient of the cost function with respect to the weights of the network for a single input-output example.
- BP gives an analytic representation of the gradient of cost function in recursive equation.

Terms in BP



- ${\partial E}/{\partial w_{ji}}$ = sensitivity factor
- δ_j^l = local gradient of a neuron j in the layer l
- y_i^l = output of neuron i in the layer l
- l = layer index
- m_l = the number of neurons in the layer l
- $\varphi_j^l = \varphi_j(v_j(n)^l)$
- $\varphi_j^{L\prime} = {\partial \varphi_j^L}/{\partial y_j}$

BP summary

- Output layer

  $$\delta_j^L = e_j^L \varphi_j^{L\prime}$$

- Hidden layer

  $$\delta_j^l = \varphi_j^{l\prime}\sum_{k=0}^{m_{l+1}} \delta_k^{l+1} w_{kj}^{l+1}$$

  where $1 \le l \le L-1$, and $0\le j \le m_l$.

- cost sensitivity

  $$\frac{\partial E}{\partial w_{ji}^{l+1}} = -\delta_j^{l+1} y_i^l$$

  where $0 \le i \le m_l$, $0 \le j \le m_{l + 1}$, and $0 \le l \le L-1$.

Weight adjustment

- Gradient descent

  $$ \Delta w_{ji}^{l+1} = -\eta \frac{\partial E}{\partial w_{ji}^{l+1}} = \eta_{ji}^{l+1} \delta_j^{l+1} y_i^l $$
  where η is the learn-rate parameter.

- Generalized delta rule

  $$ \Delta w_{ji} (n) = \alpha \Delta w_{ji} (n-1) + \eta \delta_j (n) y_i (n) $$

  where $0 \le \alpha < 1$.

  - The gradient descent with a momentum is a simple method of increasing the rate of learning while avoiding the danger of instability.
  - The momentum accelerates descent in steady downhill directions.
  - The momentum stabilizes the oscillation.

Stopping Criteria

- The learning process is stopped when the **generalization performance** is adequate or when it is apparent that the generalization performance has peaked.

4.6 Heuristics for Making the Back-Propagation Algorithm Perform Better 144

Stochastic update vs. batch update

- Stochastic is better when data sample is large and highly redundant.

Maximizing information content

- Use an example that results in the largest training error.
- Use an example that is radically different from all those previously used.

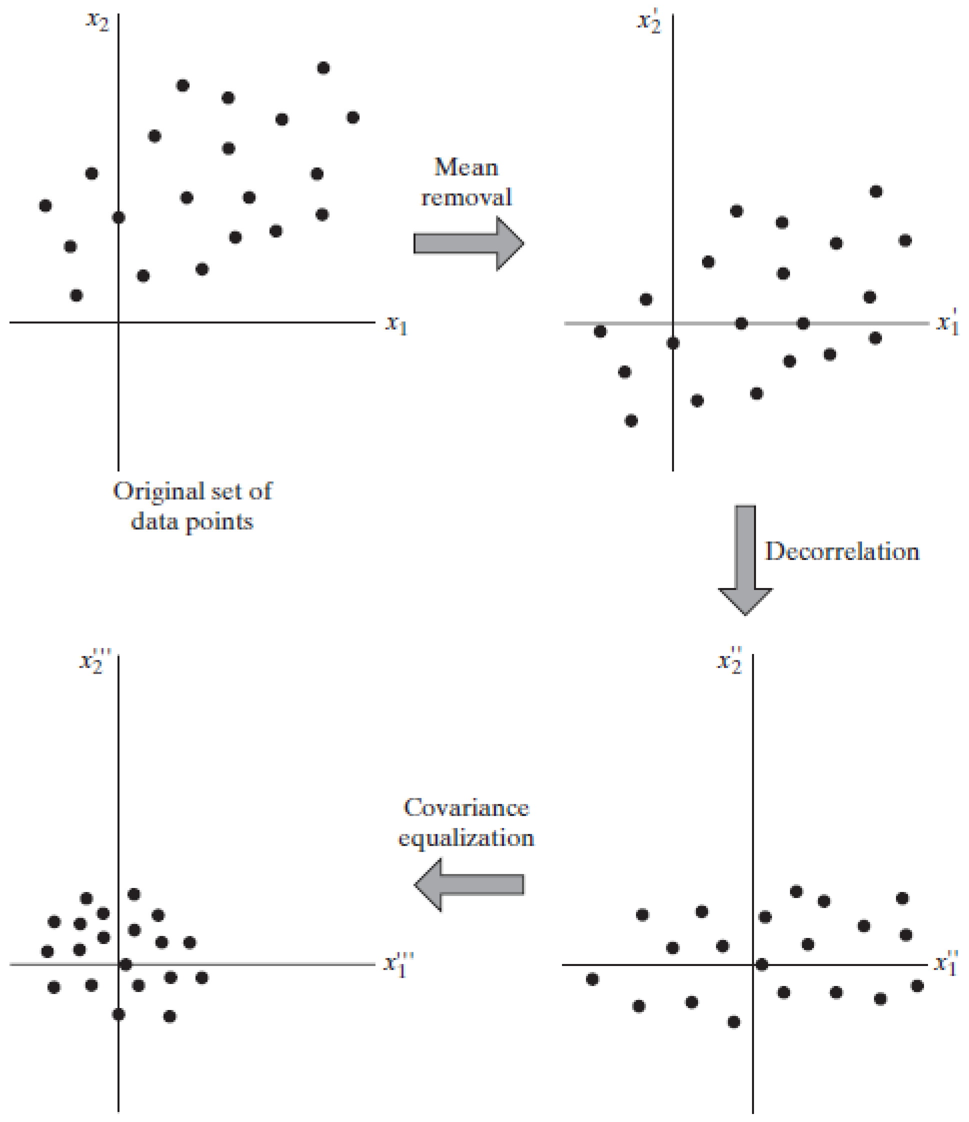
Activation function

- odd function

Target values

- It is important that the target values (desired response) be chosen within the range of the sigmoid activation function.

Normalizing the inputs



- Mean removal
- De-correlation
- Covariance equalization

Initialization

- The initial values of the **synaptic weights** and **thresholds** of the network.
- Synaptic weight initialization
    - Too large -> the neurons in the network will be driven into saturation
    - Too small -> descent may start from a saddle point.
- Strategy for hyperbolic tangent (LeCun, 1993)

  $$\sigma_w = m^{-1/2} $$

4.8 Back Propagation and Differentiation 153

4.9 The Hessian and Its Role in On-Line Learning 155

- Hessian of the cost function (empirical risk)

$$H = \frac{\partial^2 E_a (w)}{\partial w^2}$$

Roles

- Influence on the **dynamics** of gradient descent
- provides the basis for **pruning** (=deleting) insignificant synaptic weights from MLP.
- basic to the formulation of **second-order optimization method** as an alternative to GD based algorithms

4.10 Optimal Annealing and Adaptive Control of the Learning Rate 157

4.11 Generalization 164

- 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;
- The term **generalization** is borrowed from psychology.
- The learning process (i.e., training of a neural network) may be viewed as a **curvefitting** problem. The network itself may be considered simply as a **nonlinear input–output mapping**.
- When, however, a neural network learns **too many input–output examples**, the network may **end up memorizing the training data**. -> **overfitting**, or **overtraining**

Factors affecting generalization

- The size of the training sample and how representative the training sample is of the environment of interest
- The architecture of the neural network
- The physical complexity of the problem at hand

Two perspectives

- Let the network architecture fixed -> determine the size of the training sample.

- Let the size of the training sample fixed -> determine the network architecture.

- In practice,

  $$ N = O(W/\epsilon) $$

  where N = size of training sample and W = total number of free parameters.

4.12 Approximations of Functions 166

Universal approximation theorem

- Existence theorem
- $\varphi(\cdot)$ is a non-constant, bounded, monotone-increasing function.
- The theorem states that a **single hidden layer** is sufficient for a multilayer perceptron to compute a uniform ϵ-approximation to a given training set represented by the set of inputs $x_1, \cdots, x_{m_{0}}$ and a desired (target) output $f(x_1, ..., x_{m_{0}})$.
- The theorem does not say that a single hidden layer is optimum in the sense of **learning time**, **ease of implementation**, or (more importantly) **generalization**.

Bounds on approximation errors

- Functions
    - f is the target function.
    - F is approximating function with the architecture in the theorem.
    - When testing, F act as an **estimator** of new points of the target function; that is, $F = \hat{f}$.
- The bound on the risk N resulting from the use of a multilayer perceptron with m_0 input nodes and m_1 hidden neurons:

  $$E_a(N) = O \left(\frac{C_f^2}{m_1} \right) + O\left(\frac{m_0 m_1}{N} \log N\right)$$

  where C_f is the first absolute moment and it quantifies the smoothness of the function f.

    - Tradeoff
        - Accuracy of best approximation -> m_1 must be large.
        - Accuracy of empirical fit to the approximation -> m_1 should be kept small.

Practical consideration

- The problem with multilayer perceptrons using a **single hidden layer** is that the neurons therein tend to interact with each other globally. In complex situations, this interaction makes it **difficult to improve** the approximation at one point **without worsening it at some other point**.
- With two hidden layers,
    - **Local features** are extracted in the **first hidden layer**.
    - **Global features** are extracted in the **second hidden layer**.

4.13 Cross-Validation 171

- BP encodes an **input–output mapping** (represented by a set of labeled examples) into the **synaptic weights** and **thresholds** of a MLP.
- Network selection problem as choosing the best one within a set of candidate model structures (parameterization).

Cross-validation

- Randomly partition the available data set into a **training sample** and a **test set**.
- The **training sample** is further partitioned into two disjoint subsets:
    - an **estimation** subset, used to select the model;
    - a **validation** subset, used to test or validate the model.
- The training sample is used to assess the performance of **various candidate models** and thereby choose the **best** one.
- To avoid **overfitting**, the model so selected is tested with the **test set**, which is different from the validation subset.
- The usage of cross-validation
    - to determine the multilayer perceptron with **the best number of hidden neurons**;
    - to figure out **when it is best to stop** training.

Model selection

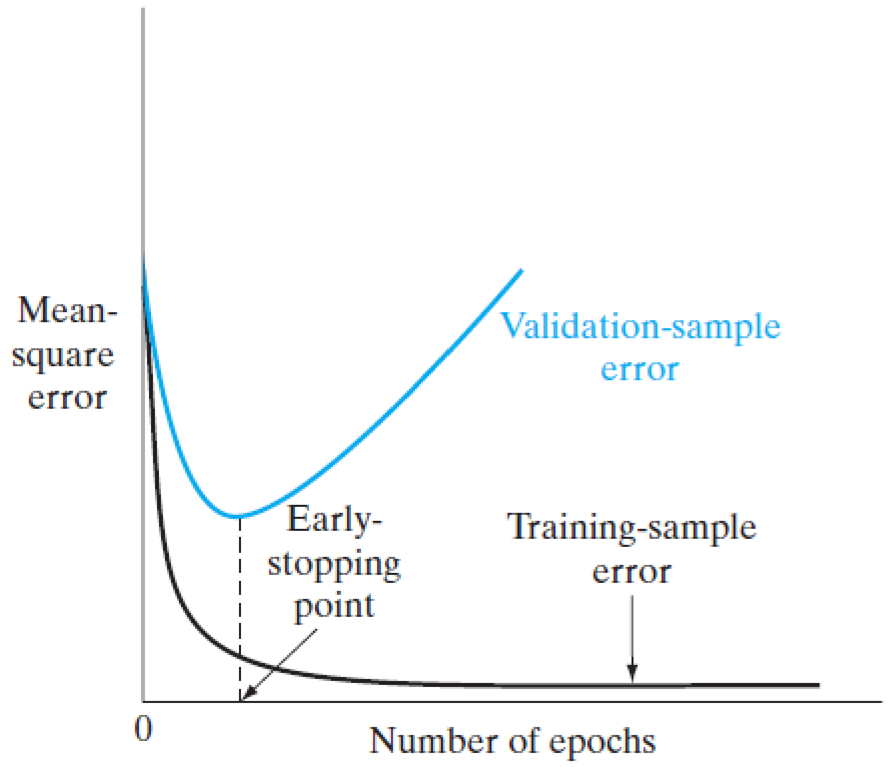
- Terms
  - F_k is the k-th hypothesis with the weight space W_k. $F_k \subset F_{k+1}$ for all $k < n$.
  - $W_{max}(N)$ is the fitting number that is the large enough number of free parameters for the training sample to be fitted adequately. It is reasonable to choose a hypothesis $F(x,w)$ with a set of weights $W \le W_{max}(N)$.
- Sets
  - The training sample S;
  - Estimation subset S' is a set of $(1-r)N$ examples where $0<r<1$;
  - Validation subset S'' is a set of rN examples.
- The size of W is smaller than $W_{max} ((1-r)N)$.
- Cross-validation result

  $$ F_{cv} = \argmin_{k=1,2,\cdots,v} e_t'' (F_k) $$

  where $e_t'' (F_k)$ is the classification error produced by hypothesis F_k when it is tested on the validation subset F''.
- Kearns (1996) reported that $r=0.2$ appears to be sensible choice.

Early-stopping method of training

- The **mean-square error** decreases with an increasing number of epochs used for training: It starts off at a large value, **decreases rapidly**, and then **continues to decrease slowly** as the network makes its way to a local minimum on the error surface.
- If a **good generalization** is the goal, it is not always desirable for the cost function to achieve the minimum. -> prone to overfitting
- CV can be used to identify the onset of overfitting: **estimation-followed-by-validation process**
  - The estimation subset S' is used to train.
  - The training session is stopped every set of epochs.
  - THen the network is tested on the validation subset.
- Example
  - 5 epochs of estimation (training) using S' -> synaptic weights and bias are all fixed and perform in forward mode to compute validation error using the validation subset S''.
  - The process is repeated.
- Validation error may form a convex function. This heuristic suggests that the minimum point on the validation learning curve be used as a sensible criterion for stopping the training session.



4.14 Complexity Regularization and Network Pruning 175

- Because the network design is statistical in nature, an appropriate tradeoff is needed between **reliability of the training data** and **goodness of the model**.

- Total risk

  $$ R(w) = E_a (w) + \lambda E_c (w) $$

  where E_a is the standard **performance metric**, E_c is the **complexity penalty**, and λ is the **regularization parameter**.

Weight-decay procedure

$$ E_c(w) = ||w||^2 $$

- Group of weights
  - weights that have a **significant influence** on the network's performance;
  - weights that have **practically little or no influence** on the network's performance. -> excess weights
- Regularization minimizes the contributes of excess weights by suppressing them, whereas it encourages the weights of significant influence.

Hessian-based network pruning: optimal brain surgeon

- Use information on **second-order derivatives of the error surface** in order to make a **trade-off** between **network complexity** and **training-error performance**.

- Taylor expansion of empirical risk

  $$ E_a (w + \Delta w) = E_a (w) + g^T \Delta w + \frac{1}{2} \Delta^T H \Delta w + O(||\Delta w||^3) $$

- Assumptions
  - **Extremal approximation**: parameters are deleted from the network only after the training process has converged (i.e., the network is fully trained).Then, the parameters will have a set of values corresponding to a local minimum or global minimum of the error surface. -> the **gradient vector** g may be set equal to zero
  - **Quadratic approximation**: the error surface around a local minimum or global minimum is **nearly quadratic**. Hence, the higher-order terms may also be neglected.
- Base function for the pruning procedure

  $$ \Delta E_a = \frac{1}{2} \Delta^T H \Delta w $$

- The goal of OBS (optimal brain surgeon) is to set one of the synaptic weights to zero in order to minimize the incremental increase in E_a.

  $$ S = \Delta E_a - \lambda (e_i^T \Delta w + w_i) = \frac{1}{2} \Delta^T H \Delta w - \lambda (e_i^T \Delta w + w_i) $$

  where λ is the Lagrange multiplier.

- Solution

  $$ \Delta w = -\frac{w_i}{[H^{-1}]{ii}}H^{-1}e_i $$ $$ S_i = \frac{w_i^2}{2[H^{-1}]{ii}} $$

  S_i is called the **saliency** of w_i.

- Computing the inverse Hessian: Hassibi and Stork (1993)

4.15 Virtues and Limitations of Back-Propagation Learning 180

4.16 Supervised Learning Viewed as an Optimization Problem 186

4.17 Convolutional Networks 201

4.18 Nonlinear Filtering 203

4.19 Small-Scale Versus Large-Scale Learning Problems 209