

Chapter 2

Fundamentals of Machine Learning

2.1 Learning Methods

Learning is a fundamental capability of neural networks. Learning rules are algorithms for finding suitable weights \mathbf{W} and/or other network parameters. Learning of a neural network can be viewed as a nonlinear optimization problem for finding a set of network parameters that minimize the cost function for given examples. This kind of parameter estimation is also called a *learning or training algorithm*.

Neural networks are usually trained by epoch. An epoch is a complete run when all the training examples are presented to the network and are processed using the learning algorithm only once. After learning, a neural network represents a complex relationship, and possesses the ability for generalization. To control a learning process, a criterion is defined to decide the time for terminating the process. The complexity of an algorithm is usually denoted as $O(m)$, indicating that the order of number of floating-point operations is m .

Learning methods are conventionally divided into supervised, unsupervised, and reinforcement learning; these schemes are illustrated in Fig. 2.1. \mathbf{x}_p and \mathbf{y}_p are the input and output of the p th pattern in the training set, $\hat{\mathbf{y}}_p$ is the neural network output for the p th input, and E is an error function. From a statistical viewpoint, unsupervised learning learns the pdf of the training set, $p(\mathbf{x})$, while supervised learning learns about the pdf of $p(\mathbf{y}|\mathbf{x})$. Supervised learning is widely used in classification, approximation, control, modeling and identification, signal processing, and optimization. Unsupervised learning schemes are mainly used for clustering, vector quantization, feature extraction, signal coding, and data analysis. Reinforcement learning is usually used in control and artificial intelligence.

In logic and statistical inference, transduction is reasoning from observed, specific (training) cases to specific (test) cases. In contrast, induction is reasoning from observed training cases to general rules, which are then applied to the test cases. Machine learning falls into two broad classes: inductive learning or transductive learning. Inductive learning pursues the standard goal in machine learning, which is to accurately classify the entire input space. In contrast, transductive learning focuses

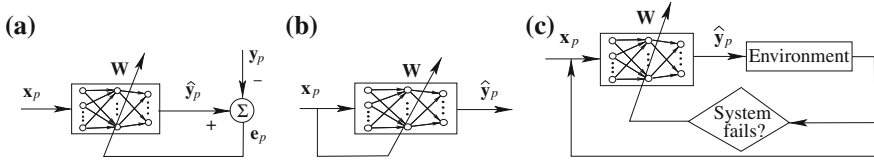


Fig. 2.1 Learning methods. **a** Supervised learning. $e_p = \hat{y}_p - y_p$. **b** Unsupervised learning. **c** Reinforcement learning

on a predefined target set of unlabeled data, the goal being to label the specific target set.

Multitask learning improves the generalization performance of learners by leveraging the domain-specific information contained in the related tasks [30]. Multiple related tasks are learned simultaneously using a shared representation. In fact, the training signals for extra tasks serve as an inductive bias [30].

In order to learn accurate models for rare cases, it is desirable to use data and knowledge from similar cases; this is known as transfer learning. Transfer learning is a general method for speeding up learning. It exploits the insight that generalization may occur not only within tasks, but also across tasks. The core idea of transfer is that experience gained in learning to perform one source task can help improve learning performance in a related, but different, target task [154]. Transfer learning is related in spirit to case-based and analogical learning. A theoretical analysis based on an empirical Bayes perspective exhibits that the number of labeled examples required for learning with transfer is often significantly smaller than that required for learning each target independently [154].

Supervised Learning

Supervised learning adjusts network parameters by a direct comparison between the actual network output and the desired output. Supervised learning is a closed-loop feedback system, where the error is the feedback signal. The error measure, which shows the difference between the network output and the output from the training samples, is used to guide the learning process. The error measure is usually defined by the mean squared error (MSE)

$$E = \frac{1}{N} \sum_{p=1}^N \|y_p - \hat{y}_p\|^2, \quad (2.1)$$

where N is the number of pattern pairs in the sample set, y_p is the output part of the p th pattern pair, and \hat{y}_p is the network output corresponding to the pattern pair p . The error E is calculated anew after each epoch. The learning process is terminated when E is sufficiently small or a failure criterion is met.

To decrease E toward zero, a gradient-descent procedure is usually applied. The gradient-descent method always converges to a local minimum in a neighborhood of the initial solution of network parameters. The LMS and BP algorithms are two most popular gradient-descent based algorithms. Second-order methods are based on the computation of the Hessian matrix.

Multiple-instance learning [46] is a variation of supervised learning. In multiple-instance learning, the examples are bags of instances, and the bag label is a function of the labels of its instances. Typically, this function is the Boolean OR. A unified theoretical analysis for multiple-instance learning and a PAC-learning algorithm are introduced in [121].

Deductive reasoning starts from a cause to deduce the consequence or effects. Inductive reasoning allows us to deduce possible causes from the consequence. The *inductive learning* is a special class of the supervised learning techniques, where given a set of $\{\mathbf{x}_i, f(\mathbf{x}_i)\}$ pairs, we determine a hypothesis $h(\mathbf{x}_i)$ such that $h(\mathbf{x}_i) \approx f(\mathbf{x}_i), \forall i$. In inductive learning, given many positive and negative instances of a problem the learner has to form a concept that supports most of the positive but no negative instances. This requires a number of training instances to form a concept in inductive learning. Unlike this, *analogical learning* can be accomplished from a single example; for instance, given a training instance of plural of fungus as fungi, one can determine the plural of bacillus: bacillus \rightarrow bacilli.

Unsupervised Learning

Unsupervised learning involves no target values. It tries to autoassociate information from the inputs with an intrinsic reduction of data dimensionality or total amount of input data. Unsupervised learning is solely based on the correlations among the input data, and is used to find the significant patterns or features in the input data without the help of a teacher. Unsupervised learning is particularly suitable for biological learning in that it does not rely on a teacher and it uses intuitive primitives like neural competition and cooperation.

A criterion is needed to terminate the learning process. Without a stopping criterion, a learning process continues even when a pattern, which does not belong to the training patterns set, is presented to the network. The network is adapted according to a constantly changing environment. Hebbian learning, competitive learning, and the SOM are the three well-known unsupervised learning approaches. Generally speaking, unsupervised learning is slow to settle into stable conditions.

In Hebbian learning, learning is a purely local phenomenon, involving only two neurons and a synapse. The synaptic weight change is proportional to the correlation between the pre- and post-synaptic signals. Many neural networks for PCA and associative memory are based on Hebbian learning. In competitive learning, the output neurons of a neural network compete for the right to respond. The SOM is also based on competitive learning. Competitive learning is directly related to clustering. The Boltzmann machine uses a stochastic training technique known as simulated annealing, which can be treated as a special type of unsupervised learning based on the inherent property of a physical system.

Reinforcement Learning

Reinforcement learning is a class of computational algorithms that specifies how an artificial agent (e.g., a real or simulated robot) can learn to select actions in order to maximize the total expected reward [11]. This computed difference, termed reward-prediction error, has been shown to correlate very well with the phasic activity of dopamine-releasing neurons projecting from the substantia nigra in nonhuman primates [124].

Reinforcement learning is a special case of supervised learning, where the exact desired output is unknown. The teacher supplies only feedback about success or failure of an answer. This is cognitively more plausible than supervised learning since a fully specified correct answer might not always be available to the learner or even the teacher. It is based only on the information as to whether or not the actual output is close to the estimate. Reinforcement learning is a learning procedure that *rewards* the neural network for its *good* output result and *punishes* it for the *bad* output result. Explicit computation of derivatives is not required. This, however, presents a slower learning process. For a control system, if the controller still works properly after an input, the output is judged as *good*; otherwise, it is considered as *bad*. The evaluation of the binary output, called *external reinforcement*, is used as the error signal.

Semi-supervised Learning and Active Learning

In many machine learning applications, such as bioinformatics, web and text mining, text categorization, database marketing, spam detection, face recognition, and video-indexing, abundant amounts of unlabeled data can be cheaply and automatically collected. However, manual labeling is often slow, expensive, and error-prone. When only a small number of labeled samples are available, unlabeled samples could be used to prevent the performance degradation due to overfitting.

The goal of semi-supervised learning is to employ a large collection of unlabeled data jointly with a few labeled examples for improving generalization performance. Some semi-supervised learning methods are based on some assumptions that relate the probability $P(x)$ to the conditional distribution $P(Y = 1|X = x)$. Semi-supervised learning is related to the problem of transductive learning. Two typical semi-supervised learning approaches are learning with the cluster assumption [147] and learning with the manifold assumption [18]. The cluster assumption requires that data within the same cluster are more likely to have the same label. The most prominent example is the transductive SVM [147].

Universum data are given a set of unlabeled examples and do not belong to either class of the classification problem of interest. Contradiction happens when two functions in the same equivalence class have different signed outputs on a sample from the Universum. Universum learning is conceptually different from semi-supervised learning or transduction [147], because the Universum data is not from the same distribution as the labeled training data. Universum learning implements a trade-off between explaining training samples (using large margin hyperplanes) and maximizing the number of contradictions (on the Universum).

In active learning, or so-called *pool-based active learning*, the labels of data points are initially hidden, and the learner must pay for each label he wishes to be revealed. The goal of active learning is to actively select the most informative examples for manual labeling in these learning tasks, that is, designing input signals for optimal generalization [55]. Based on conditional expectation of the generalization error, a pool-based active learning method effectively copes with model misspecification by weighting training samples according to their importance [135]. Reinforcement learning can be regarded as a form of active learning. At this point, a query mechanism proactively asks for the labels of some of the unlabeled data.

Examples of situations in which active learning can be employed are web searching, email filtering, and relevance feedback for a database or website. The first two examples involve induction. The goal is to create a classifier that works well on unseen future instances. The third situation is an example of transduction [147]. The learner's performance is assessed on the remaining instances in the database rather than a totally independent test set.

The query-by-committee algorithm [56] is an active learning algorithm for classification, which uses a prior distribution over hypotheses. In this algorithm, the learner observes a stream of unlabeled data and makes spot decisions about whether or not to ask for each point's label. If the data is drawn uniformly from the surface of the unit sphere in R^d , and the hidden labels correspond perfectly to a homogeneous (i.e., through the origin) linear separator from this same distribution, then it is possible to achieve generalization error ϵ after seeing $O((d/\epsilon) \log(1/\epsilon))$ points and requesting just $O(d \log(1/\epsilon))$ labels: an exponential improvement over the usual $O(d/\epsilon)$ sample complexity of learning linear separators in a supervised setting. The query-by-committee algorithm involves random sampling from intermediate version spaces; the complexity of the update step scales polynomially with the number of updates performed.

An information-based approach for active data selection is presented in [89]. In [134], a two-stage sampling scheme for reducing both the bias and variance is given, and based on it, two active learning methods are given.

In a framework for batch mode active learning [75], a number of informative examples are selected for manual labeling in each iteration. The key feature is to reduce the redundancy among the selected examples such that each example provides unique information for model updating. The set of unlabeled examples that can efficiently reduce the Fisher information of the classification model is chosen [75].

2.2 Learning and Generalization

From an approximation viewpoint, learning is a hypersurface reconstruction based on existing examples, while generalization means estimating the value on the hypersurface where there is no example. Mathematically, the learning process is a nonlinear curve-fitting process, while generalization is the interpolation and extrapolation of the input data.

The goal of training neural networks is not to learn an exact representation of the training data itself, but rather to build a statistical model of the process which generates the data. The problem of reconstructing the mapping is said to be *well-posed* if an input always generates a unique output, and the mapping is continuous. Learning is an ill-posed inverse problem. Given examples of an input-output mapping, an approximate solution is required to be found for the mapping. The input data may be noisy or imprecise, and also may be insufficient to uniquely construct the mapping. The regularization technique can transform an ill-posed problem into a well-posed one so as to stabilize the solution by adding some auxiliary non-negative functional for constraints [105, 139].

When a network is overtrained with too many examples, parameters or epochs, it may produce good results for the training data, but has a poor generalization capability. This is the overfitting phenomenon, and is illustrated in Fig. 2.2. In statistics, overfitting applies to the situation wherein a model possesses too many parameters, and fits the noise in the data rather than the underlying function. A simple network with smooth input-output mapping usually has a better generalization capability. Generally, the generalization capability of a network is jointly determined by the size of the training pattern set, the complexity of the problem, and the architecture of the network.

Example 2.1 To approximate a noisy cosine function, with 20 random samples, we employ a 1-30-1 feedforward network. The result is plotted in Fig. 2.2. The noisy samples is represented by the “o” symbols, and the true network response is given by the solid line. Clearly, the learned network is overfitted, and it does not generalize well. Notice that if the number of parameters in the network is much smaller than the total number of points in the training set, then there is little or no worry of overfitting.

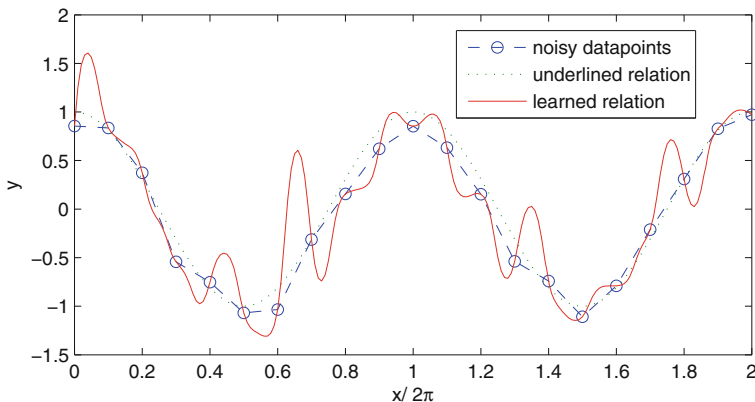


Fig. 2.2 Proper fitting and overfitting. *Dashed line* corresponds to proper fitting, and *solid line* corresponds to overfitting

For a given network topology, we can estimate the minimal size of the training set for successfully training the network. For conventional curve-fitting techniques, the required number of examples usually grows with the dimensionality of the input space, namely, the *curse of dimensionality*. Feature extraction can reduce input dimensionality and thus improve the generalization capability of the network.

The training set should be sufficiently large and diverse so that it could represent the problem well. For good generalization, the size of the training set, N , should be at least several times larger than the network's capacity, i.e., $N \gg \frac{N_w}{N_y}$, where N_w the total number of weights or free parameters, and N_y the number of output components [149].

2.2.1 Generalization Error

The generalization error of a trained network can be decomposed into two parts, namely, an *approximation error* that is due to a finite number of parameters of the approximation scheme used and an unknown level of noise in the training data, and an *estimation error* that is due to a finite number of data available [97]. For a feedforward network with J_1 input nodes and a single output node, a bound on the generalization error is associated with the order of hypothesis parameters N_P and the number of examples N [97]

$$O\left(\frac{1}{N_P}\right) + O\left(\left[\frac{N_P J_1 \ln(N_P N) - \ln \delta}{N}\right]^{1/2}\right), \text{ with probability } p > 1 - \delta, \quad (2.2)$$

where $\delta \in (0, 1)$ is the confidence parameter, and N_P is proportional to the number of parameters, such as N_P centers in an RBF network, or N_P sigmoidal hidden units in an MLP. The first term corresponds to the bound on the approximation error, and the second to that on the estimation error.

As N_P increases, the approximation error decreases since a larger model is used; however, the estimation error increases due to overfitting (or alternatively, more data). Thus, one cannot reduce the upper bounds on both the error components simultaneously. Given the amount of data available, the optimal size of the model for the tradeoff between the approximation and estimation errors is selected as $N_P \propto N^{\frac{1}{3}}$ [97]. After suitably selecting N_P and N , the generalization error for feedforward networks should be $O\left(\frac{1}{N_P}\right)$. This result is similar to that for an MLP with sigmoidal functions [10].

2.2.2 Generalization by Stopping Criterion

Generalization can be controlled during training. Overtraining can be avoided by stopping the training before the absolute minimum is reached. Neural networks

trained with iterative gradient-based methods tend to learn a mapping in the hierarchical order of its increasing components of frequency. When training is stopped at an appropriate point, the network will not learn the high-frequency noise. While the training error will always decrease, the generalization error will decrease to a minimum and then begins to rise again as the network is being overtrained. Training should stop at the optimum stopping point. The generalization error is defined in the same form as the learning error, but on a separate validation set of data. Early stopping is the default method for improving generalization.

Example 2.2 In order to use early stopping technique, the available data is divided into three subsets. The first subset is the training set. The second subset is the validation set. The error on the validation set is monitored during the training process. The validation error normally decreases during the initial phase of training, as does the error on the training set. However, when the network begins to overfit the data, the error on the validation set typically begins to rise. When the validation error increases for a specified number of iterations, training is stopped, and the weights and biases at the minimum of the validation error are returned. The error on the test set is not used during training, but it is used to compare different models. It is also useful to plot the error on the test set during the training process. If the error on the test set reaches a minimum at a significantly different iteration number than the error on the validation set, this might indicate a poor division of the dataset. From Example 2.1, the 40 data samples are divided by 60, 20, and 20 % of samples as the training, validation, and test sets. The relation is illustrated in Fig. 2.3.

Early stopping is implemented with crossvalidation to decide when to stop. Three early-stopping criteria are defined and empirically compared in [106]. Slower stopping criteria, which stop later than others, on average lead to small improvements in generalization, but result in a much longer training time [106].

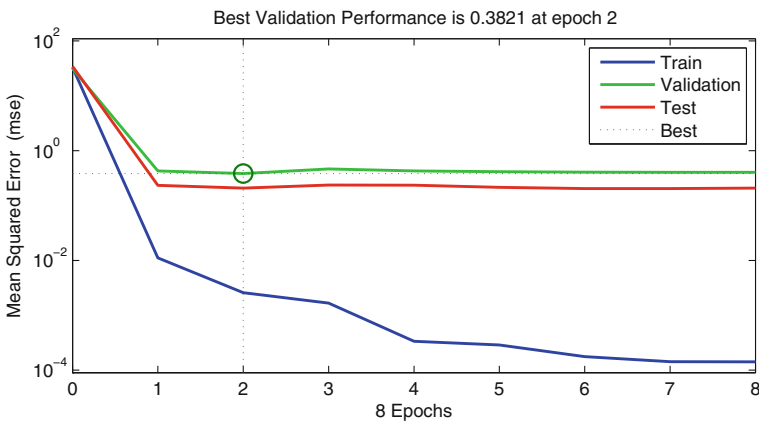


Fig. 2.3 Learning and generalization. When the network is overtrained, its generalization performance degrades

Statistical analysis for the three-layer MLP has been performed in [4]. As far as the generalization performance is concerned, exhaustive learning is satisfactory when $N > 30N_w$. When $N < N_w$, early stopping can really prevent overtraining. When $N < 30N_w$, overtraining may also occur. In the latter two cases, crossvalidation can be used to stop training.

An optimized approximation algorithm [87] avoids overfitting in function approximation applications. The algorithm utilizes a quantitative stopping criterion based on the estimation of the signal-to-noise-ratio figure (SNRF). Using SNRF, overfitting can be automatically detected from the training error only.

2.2.3 Generalization by Regularization

Regularization is a reliable method for improving generalization. The target function is assumed to be smooth, and small changes in the input do not cause large changes in the output. A constraint term E_c , which penalizes poor generalization, is added to the standard training cost function E

$$E_T = E + \lambda_c E_c, \quad (2.3)$$

where λ_c is a positive value that balances the tradeoff between error minimization and smoothing. In contrast to the early-stopping criterion method, the regularization method is applicable to both the iterative gradient-based techniques and the one-step linear optimization such as the singular value decomposition (SVD) technique.

Network-pruning techniques such as the weight-decay technique also help to improve generalization [23, 108]. At the end of training, there are some weights significantly different from zero, while some other weights are close to zero. Those connections with small weights can be removed from the network. Biases should be excluded from the penalty term so that the network yields an unbiased estimate of the true target mean.

Early stopping has a behavior similar to that of a simple weight-decay technique in the case of the MSE function [22]. The quantity $\frac{1}{\eta t}$, where η is the learning rate and t is the iteration index, plays the role of λ_c . The effective number of weights, that is, the number of weights whose values differ significantly from zero, grows as training proceeds.

Training with a small amount of jitter in the input while keeping the same output can improve generalization. With jitter, the learning problem is equivalent to a smoothing regularization with the noise variance playing the role of the regularization parameter [22, 108]. Training with jitter thus allows regularization within the conventional layered feedforward network architecture. Although large networks are generally trained rapidly, they tend to generalize poorly due to insufficient constraints. Training with jitter helps to prevent overfitting.

In [76], noise is added to the available training set to generate an unlimited source of training samples. This is interpreted as a kernel estimate of the probability density

that describes the training vector distribution. It helps to enhance the generalization performance, speed up the BP, and reduce the possibility of local minima entrapment.

In [73], each weight is encoded with a short bit-length to decrease the complexity of the model. The amount of information in a weight can be controlled by adding Gaussian noise and the noise level can be adapted during learning to optimize the tradeoff between the expected squared error of the network and the amount of information in the weights.

Weight sharing is to control several weights by a single parameter [119]. This reduces the number of free parameters in a network, and thus improves generalization. A soft weight-sharing method is implemented in [98] by adding a regularization term to the error function, where the learning algorithm decides which of the weights should be tied together.

Regularization decreases the representation capability of the network, but increases the bias (bias–variance dilemma [59]). The principle of regularization is to choose a well-defined regularizer to decrease the variance by affecting the bias as little as possible [22].

2.2.4 Fault Tolerance and Generalization

Fault tolerance is strongly associated with generalization. Input noise during training improves generalization ability [23], and synaptic noise during training improves fault tolerance [94]. When fault tolerance is improved, the generalization ability is usually better [52], and vice versa [42]. The lower the weight magnitude, the higher the fault tolerance [20, 42] and the generalization ability [20, 82]. Based on the Vapnik-Chervonenkis (VC) dimension, it is qualitatively explained in [102] why adding redundancy can improve fault tolerance and generalization.

Fault tolerance is related to a uniform distribution of the learning among the different neurons, but the BP algorithm does not guarantee this good distribution [52]. Just as input noise is introduced to enhance the generalization ability, the perturbation of weights during training also increases the fault tolerance of MLPs [52, 94]. Saliency, used as a measurement of fault tolerance to weight deviations [94], is computed from the diagonal elements of the Hessian matrix of the error with respect to the weight values. A low value of saliency implies a higher fault tolerance.

An analysis of the influence of weight and input perturbations in an MLP is made in [20]. The measurements introduced are explicitly related to the MSE degradation in the presence of perturbations, thus constituting a selection criterion between different alternatives of weight configurations. Quantitative measurements of fault tolerance, noise immunity, and generalization ability are provided, from which several previous conjectures are deduced. The methodology is also applicable to the study of the tolerance to perturbations of other layered networks such as the RBF networks.

When training the MLP with on-line node fault injection, hidden nodes randomly output zeros during training. The trained MLP is able to tolerate random node fault.

The convergence of the algorithm is proved in [136]. The corresponding objective functions consist of an MSE term, a regularizer term, and a weight decay term.

Six common fault/noise-injection-based online learning algorithms, namely, injecting additive input noise, injecting additive/multiplicative weight noise, injecting multiplicative node noise, injecting multiweight fault (random disconnection of weights), injecting multinode fault during training, and weight decay with injecting multinode fault, are investigated in [74] for RBF networks. The convergence of the six online algorithms is shown to be almost sure, and their true objective functions being minimized are derived. For injecting additive input noise during training, the objective function is identical to that of the Tikhonov regularizer approach. For injecting additive/multiplicative weight noise during training, the objective function is the simple mean square training error; thus, injecting additive/multiplicative weight noise during training cannot improve the fault tolerance of an RBF network. Similar to injective additive input noise, the objective functions of other fault/noise-injection-based online algorithms contain an MSE term and a specialized regularization term.

2.2.5 Sparsity Versus Stability

Stability establishes the generalization performance of an algorithm [26]. Sparsity and stability are two desired properties of learning algorithms. Both properties lead to good generalization ability. These two properties are fundamentally at odds with each other and this no-free-lunch theorem is proved in [152, 153]: A sparse algorithm cannot be stable and vice versa. A sparse algorithm can have nonunique optimal solutions and is therefore ill-posed. If an algorithm is sparse, then its uniform stability is lower bounded by a nonzero constant. This also shows that any algorithmically stable algorithm cannot be sparse. Thus, one has to trade off sparsity and stability in designing a learning algorithm.

In [152, 153], L_1 -regularized regression (LASSO) is shown to be not stable, while L_2 -regularized regression is known to have strong stability properties and is therefore not sparse. Sparsity promoting algorithms include LASSO, L_1 -norm SVM, deep belief network, and sparse PCA.

2.3 Model Selection

Occam's razor was formulated by William of Occam in the late Middle Ages. Occam's razor principle states: "No more things should be presumed to exist than are absolutely necessary." That is, if two models of different complexity fit the data approximately equally well, the simpler one usually is a better predictive model. From models approximating the noisy data, the ones that have minimal complexity should be chosen.

The objective of model selection is to find a model that is as simple as possible that fits a given dataset with sufficient accuracy, and has a good generalization capability to unseen data. The generalization performance of a network gives a measure of the quality of the chosen model. Model-selection approaches can be generally grouped into four categories: crossvalidation, complexity criteria, regularization, and network pruning/growing.

The generalization error of a learning method can be estimated via either crossvalidation or bootstrap. In crossvalidation methods, many networks of different complexity are trained and then tested on an independent validation set. The procedure is computationally demanding and/or requires additional data withheld from the total pattern set. In complexity criterion-based methods, training of many networks is required and hence, computationally demanding, though a validation set is not required. Regularization methods are more efficient than crossvalidation techniques, but the results may be suboptimal since the penalty terms damage the representation capability of the network. Pruning/growing methods can be under the framework of regularization, which often makes restrictive assumptions, resulting in networks that are suboptimal.

2.3.1 Crossvalidation

Crossvalidation is a standard model-selection method in statistics [78]. The total pattern set is randomly partitioned into a training set and a validation (test) set. The major part of the total pattern set is included in the training set, which is used to train the network. The remaining, typically, 10–20%, is included in the validation set and is used for validation. When only one sample is used for validation, the method is called *leave-one-out* crossvalidation. Methods on conducting crossvalidation are given in [106]. This kind of hold-out estimate of performance lacks computational efficiency due to the repeated training, but with lower variance of the estimate.

Let \mathcal{D}_i and $\bar{\mathcal{D}}_i$, $i = 1, \dots, m$, be the data subsets of the total pattern set arising from the i th partitioning, which are, respectively, used for training and testing. The crossvalidation process trains the algorithm m times, and is actually to find a suitable model by minimizing the log-likelihood function

$$E_{cv} = -\frac{1}{m} \sum_{i=1}^m \ln (L(\hat{\mathbf{W}}(\bar{\mathcal{D}}_i) | \mathcal{D}_i)), \quad (2.4)$$

where $\hat{\mathbf{W}}(\bar{\mathcal{D}}_i)$ denotes the maximum-likelihood (ML) parameter estimates on $\bar{\mathcal{D}}_i$, and $L(\hat{\mathbf{W}}(\bar{\mathcal{D}}_i) | \mathcal{D}_i)$ is the likelihood evaluated on the dataset \mathcal{D}_i .

Validation uses data different from the training set, thus the validation set is independent from the estimated model. This helps to select the best one among the different model parameters. Since this dataset is independent from the estimated model, the generalization error obtained is a fair estimate. Sometimes it is not optimal

if we train the network to perfection on a given pattern set due to the ill-posedness of the finite training pattern set. Crossvalidation helps to generate good generalization of the network, when N , the size of the training set, is too large. Crossvalidation is effective for finding a large network with a good generalization performance.

The popular K -fold crossvalidation [133] employs a nonoverlapping test set selection scheme. The data universe \mathcal{D} is divided into K nonoverlapping data subsets of the same size. Each data subset is then used as a test set, with the remaining $K - 1$ folds acting as a training set, and an error value is calculated by testing the classifier in the remaining fold. Finally, the K -fold crossvalidation estimation of the error is the average value of the errors committed in each fold. Thus, the K -fold crossvalidation error estimator depends on two factors: the training set and the partitioning into folds. Estimating the variance of K -fold crossvalidation can be done from independent realizations or from dependent realizations whose correlation is known. K -fold crossvalidation produces dependent test errors. Consequently, there is no universal unbiased estimator of the variance of K -fold crossvalidation that is valid under all distributions [19].

The variance estimators of the K -fold crossvalidation estimator of the generalization error presented in [91] are almost unbiased in the cases of smooth loss functions and the absolute error loss. The problem of variance estimation is approached as a problem in approximating the moments of a statistic. The estimators depend on the distribution of the errors and on the knowledge of the learning algorithm. Overall, a test set that use 25 % of the available data seems to be a reasonable compromise in selecting among the various forms of K -fold crossvalidation [91].

The leave-many-out variants of crossvalidation perform better than the leave-one-out versions [104]. Empirically, both types of crossvalidation can exhibit high variance in small samples, but this may be alleviated by increasing the level of resampling. Used appropriately, leave-many-out crossvalidation is, in general, more robust than leave-one-out crossvalidation [104].

The moment approximation estimator [91] performs better in terms of both the variance and the bias than the Nadeau-Bengio estimator [95]. The latter is computationally simpler than the former for general loss functions, as it does not require the computation of the derivatives of the loss function; but it is not an appropriate one to be used for nonrandom test set selection.

Crossvalidation and bootstrapping are both resampling methods. Resampling varies the training set numerous times based on one set of available data. One fundamental difference between crossvalidation and bootstrapping is that bootstrapping resamples the available data at random with replacement, whereas crossvalidation resamples the available data at random without replacement. Crossvalidation methods never evaluate the trained networks over examples that appear in the training set, whereas bootstrapping methods typically do that. Crossvalidation methods split the data such that a sample does not appear in more than one validation set. Crossvalidation is commonly used for estimating generalization error, whereas bootstrapping finds widespread use in estimating error bars and confidence intervals. Crossvalidation is commonly believed to be more accurate (less biased) than bootstrapping, but to have a higher variance than bootstrapping does in small samples [104].

2.3.2 Complexity Criteria

An efficient approach for improving the generalization performance is to construct a small network using a parsimonious principle. Statistical model selection with information criteria such as Akaike's final prediction error criterion [1], Akaike information criterion (AIC) [3], Schwartz's Bayesian information criterion (BIC) [125], and Rissanen's minimum description length (MDL) principle [112] are popular and have been widely used for model selection of neural networks. Although the motivations and approaches for these criteria may be very different from one another, most of them can be expressed as a function with two components, one for measuring the training error and the other for penalizing the complexity. These criteria penalize large-size models.

A possible approach to model order selection consists of minimizing the Kullback–Leibler discrepancy between the true pdf of the data and the pdf (or likelihood) of the model, or equivalently maximizing the relative Kullback–Leibler information, which is sometimes called the relative Kullback–Leibler information. Maximizing the asymptotic approximation of the relative Kullback–Leibler information with n , the number of variables, is equivalent to minimizing the AIC function of n . AIC is derived by maximizing an asymptotically unbiased estimate of the relative Kullback–Leibler information I . The BIC rule can be derived from an asymptotically unbiased estimate of the relative Kullback–Leibler information [132]. BIC is the penalized ML method.

The AIC and BIC criteria can be, respectively, represented by

$$E_{\text{AIC}} = -\frac{1}{N} \ln (L_N (\widehat{\mathbf{W}}_N)) + \frac{N_P}{N}, \quad (2.5)$$

$$E_{\text{BIC}} = -\frac{1}{N} \ln (L_N (\widehat{\mathbf{W}}_N)) + \frac{N_P}{2N} \ln N, \quad (2.6)$$

where $L_N (\widehat{\mathbf{W}}_N)$ is the likelihood estimated for a training set of size N and model parameters $\widehat{\mathbf{W}}_N$, and N_P is the number of parameters in the model. More specifically, the two criteria can be expressed by [132]

$$\text{AIC}(N_P) = R_{\text{emp}}(N_P) + \frac{2N_P}{N} \hat{\sigma}^2, \quad (2.7)$$

$$\text{BIC}(N_P) = R_{\text{emp}}(N_P) + \frac{N_P}{N} \hat{\sigma}^2 \ln N, \quad (2.8)$$

where $\hat{\sigma}^2$ denotes an estimate of noise variance, and the empirical risk is given by

$$R_{\text{emp}}(N_P) = \frac{1}{N} \sum_{i=1}^N (y_i - f(\mathbf{x}_i, N_P))^2, \quad (2.9)$$

and the noise variance can be estimated, for a linear estimator with N_P parameters, as

$$\hat{\sigma}^2 = \frac{N}{N - N_P} \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2. \quad (2.10)$$

This leads to the following form of AIC known as final prediction error [2]:

$$\text{FPE}(N_P) = \frac{1 + \frac{N_P}{N}}{1 - \frac{N_P}{N}} R_{\text{emp}}(N_P). \quad (2.11)$$

The MDL principle stems from coding theory to find as short a description as possible of a database with as few symbols as possible [112, 113]. The description length of the model characterizes the information needed for simultaneously encoding a description of the model and a description of the prediction errors of the model. The best model is the one with the minimum description length. The total description length E_{MDL} has three terms: code cost for coding the input vectors, model cost for defining the reconstruction method, and reconstruction error due to reconstruction of the input vector from its code. The description length is described by the number of bits. Existing unsupervised learning algorithms such as the competitive learning and PCA can be explained using the MDL principle [73]. Good generalization can be achieved by encoding the weights with short bit-lengths by penalizing the amount of information they contain using the MDL principle [73]. The MDL measure can be regarded as an approximation of the Bayesian measure, and thus has a Bayesian interpretation. BIC rule has also been obtained by an approach based on coding arguments and the MDL principle.

Generalization error Err is characterized by the sum of the training (approximation) error err and the degree of optimism OP inherent in a particular estimate [61], that is, $Err = err + OP$. Complexity criteria such as BIC can be used for estimating OP .

2.4 Bias and Variance

The generalization error can be represented by the sum of the *bias* squared plus the *variance* [59]. Most existing supervised learning algorithms suffer from the bias-variance dilemma [59]. That is, the requirements for small bias and small variance are conflicting and a tradeoff must be made.

Let $f(\mathbf{x}; \hat{\mathbf{w}})$ be the best model in model space. Thus, $\hat{\mathbf{w}}$ does not depend on the training data. The bias and variance can be defined by [22]

$$\text{bias} = E_{\mathcal{S}}(f(\mathbf{x})) - f(\mathbf{x}; \hat{\mathbf{w}}), \quad (2.12)$$

$$\text{var} = E_{\mathcal{S}} \left((f(\mathbf{x}) - E_{\mathcal{S}}(f(\mathbf{x})))^2 \right), \quad (2.13)$$

where $f(\mathbf{x})$ is the function to be estimated, and $E_{\mathcal{S}}$ denotes the expectation operation over all possible training sets. Bias is caused by an inappropriate choice of the size of a class of models when the number of training samples is assumed infinite, while the variance is the error caused by the finite number of training samples.

Example 2.3 An illustration of the concepts of bias and variance in the two-dimensional space is shown in Fig. 2.4. $f(x; \hat{\mathbf{w}})$ is the underlying function; $f_1(x)$ and $f_2(x)$ are used to approximate $f(x; \hat{\mathbf{w}})$: $f_1(x)$ is an exact interpolation of the data points, while $f_2(x)$ is a fixed function independent of the data points. For $f_1(x)$, the bias is zero at the data points and is small in the neighborhood of the data points, while the variance is the variance of the noise on the data, which could be significant; for $f_2(x)$, the bias is high while the variance is zero.

The generalized error can be decomposed into a sum of the bias and variance

$$\begin{aligned}
 & E_{\mathcal{S}} \left([f(\mathbf{x}) - f(\mathbf{x}, \hat{\mathbf{w}})]^2 \right) \\
 &= E_{\mathcal{S}} \left(\{ [f(\mathbf{x}) - E_{\mathcal{S}}(f(\mathbf{x}))] + [E_{\mathcal{S}}(f(\mathbf{x})) - f(\mathbf{x}, \hat{\mathbf{w}})] \}^2 \right) \\
 &= E_{\mathcal{S}} \left([f(\mathbf{x}) - E_{\mathcal{S}}(f(\mathbf{x}))]^2 \right) + E_{\mathcal{S}} \left([E_{\mathcal{S}}(f(\mathbf{x})) - f(\mathbf{x}, \hat{\mathbf{w}})]^2 \right) \\
 &\quad + 2E_{\mathcal{S}} \left([f(\mathbf{x}) - E_{\mathcal{S}}(f(\mathbf{x}))] [E_{\mathcal{S}}(f(\mathbf{x})) - f(\mathbf{x}, \hat{\mathbf{w}})] \right) \\
 &= (\text{Bias})^2 + \text{Var}.
 \end{aligned} \tag{2.14}$$

A network with a small number of adjustable parameters gives poor generalization on new data, since the model has very little flexibility and thus yields underfitting with a high bias and low variance. In contrast, a network with too many adjustable

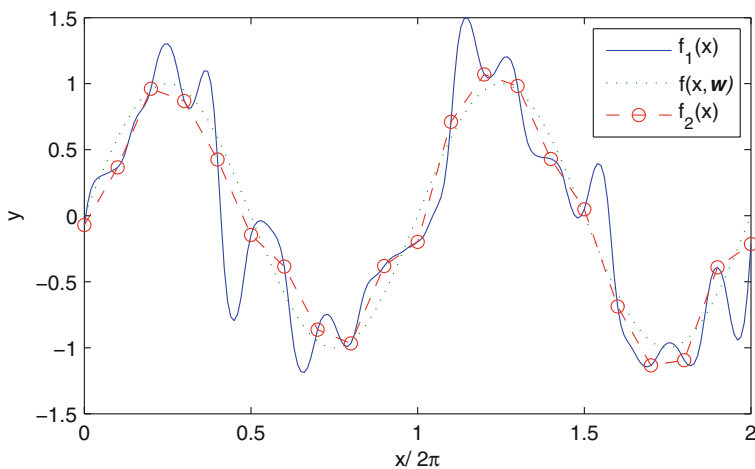


Fig. 2.4 Bias and variance. Circles denote examples from a training set

parameters also gives a poor generalization performance, since it is too flexible and fits too much of the noise on the training data, thus yielding overfitting with a low bias but high variance. The best generalization performance is achieved by balancing bias and variance, which optimizes the complexity of the model through either finding a model with an optimal size or by adding a regularization term in an objective function. For nonparametric methods, most complexity criteria based techniques operate on the variance term in order to get a good compromise between the contributions made by the bias and variance to the error. When the number of hidden cells is increased, the bias term is likely to be reduced, whereas the variance would increase.

For three-layer feedforward networks with N_P hidden sigmoidal units, the bias and variance are upper bounded explicitly [10] by $O\left(\frac{1}{N_P}\right)$ and $O\left(\frac{N_P J_1 \ln N}{N}\right)$, respectively, where N is the size of the training set and J_1 is the dimensionality of the feature vectors. Thus when N_P is large, the bias is small. However, when N is finite, a network with an excessively large space complexity will overfit the training set. The average performance can decrease as N_P gets larger. As a result, a tradeoff needs to be made between the bias and variance.

While unbiasedness is a beneficial quality of a model selection criterion, a low variance is at least as important, as a nonnegligible variance introduces the potential for overfitting in model selection as well as in training the model [33]. The effects of this form of overfitting are often comparable to differences in performance between learning algorithms [33]. This could be ameliorated by regularization of the model selection criterion [32].

2.5 Robust Learning

When the training data is corrupted by large noise, such as outliers, conventional learning algorithms may not yield acceptable performance since a small number of outliers have a large impact on the MSE. An outlier is an observation that deviates significantly from the other observations; this may be due to erroneous measurements or noisy data from the tail of the noise distribution functions. When noise becomes large or outliers exist, the networks may try to fit those improper data and thus, the learned systems are corrupted. The Student- t distribution has heavier tails than the Gaussian distribution and is therefore less sensitive to any departure of the empirical distribution from Gaussianity. For nonlinear regression, the techniques of robust statistics [77] can be applied to deal with the outliers. The M -estimator is derived from the ML estimator to deal with situations, where the exact probability model is unknown. The M -estimator replaces the conventional squared error term by the so-called *loss functions*. The loss function is used to degrade the effects of those outliers in learning. A difficulty is the selection of the scale estimator of the loss function in the M -estimator.

The cost function of a robust learning algorithm is defined by

$$E_r = \sum_{i=1}^N \sigma(\epsilon_i; \beta), \quad (2.15)$$

where $\sigma(\cdot)$ is the loss function, which is a symmetric function with a unique minimum at zero, $\beta > 0$ is the scale estimator, known as the *cutoff parameter*, ϵ_i is the estimated error for the i th training pattern, and N is the size of the training set. The loss function can be typically selected as one of the following functions:

- The logistic function [77]

$$\sigma(\epsilon_i; \beta) = \frac{\beta}{2} \ln \left(1 + \frac{\epsilon_i^2}{\beta} \right). \quad (2.16)$$

- Huber's function [77]

$$\sigma(\epsilon_i; \beta) = \begin{cases} \frac{1}{2} \epsilon_i^2, & |\epsilon_i| \leq \beta \\ \beta |\epsilon_i| - \frac{1}{2} \beta^2, & |\epsilon_i| > \beta \end{cases}. \quad (2.17)$$

- Talwar's function [43]

$$\sigma(\epsilon_i; \beta) = \begin{cases} \frac{1}{2} \epsilon_i^2, & |\epsilon_i| \leq \beta \\ \frac{1}{2} \beta^2, & |\epsilon_i| > \beta \end{cases}. \quad (2.18)$$

- Hampel's tanh estimator [35]

$$\sigma(\epsilon_i; \beta_1, \beta_2) = \begin{cases} \frac{1}{2} \epsilon_i^2, & |\epsilon_i| \leq \beta_1 \\ \frac{1}{2} \beta_1^2 - \frac{2c_1}{c_2} \ln \frac{1+e^{c_2(\beta_2-|\epsilon_i|)}}{1+e^{c_2(\beta_2-\beta_1)}} - c_1 (|\epsilon_i| - \beta_1), & \beta_1 < |\epsilon_i| \leq \beta_2 \\ \frac{1}{2} \beta_1^2 - \frac{2c_1}{c_2} \ln \frac{2}{1+e^{c_2(\beta_2-\beta_1)}} - c_1 (\beta_2 - \beta_1), & |\epsilon_i| > \beta_2 \end{cases}. \quad (2.19)$$

In the tanh estimator, β_1 and β_2 are two cutoff points, and constants c_1 and c_2 adjust the shape of the influence function (to be defined in (2.21)). When $c_1 = \frac{\beta_1}{\tan(c_2(\beta_2-\beta_1))}$, the influence function is continuous. In the interval of the two cutoff points, the influence function can be represented by a hyperbolic tangent relation.

Using the gradient-descent method, the weights are updated by

$$\Delta w_{jk} = -\eta \frac{\partial E_r}{\partial w_{jk}} = -\eta \sum_{i=1}^N \varphi(\epsilon_i; \beta) \frac{\partial \epsilon_i}{\partial w_{jk}}, \quad (2.20)$$

where η is a learning rate or step size, and $\varphi(\cdot)$, called the *influence function*, is given by

$$\varphi(\epsilon_i; \beta) = \frac{\partial \sigma(\epsilon_i; \beta)}{\partial \epsilon_i}. \quad (2.21)$$

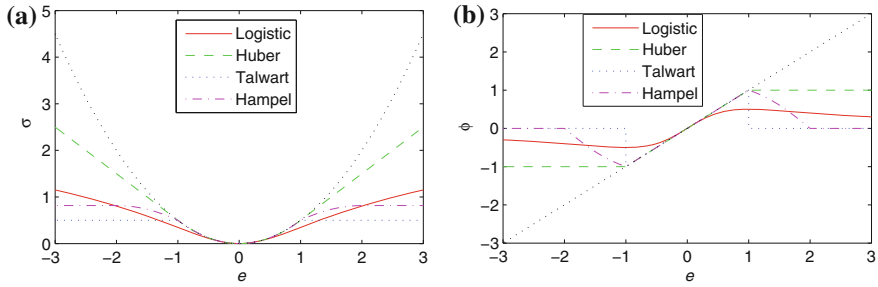


Fig. 2.5 Loss functions and their respective influence functions. For logistic, Huber's, and Talwart's functions, $\beta = 1$. For Hampel's tanh estimator, $\beta_1 = 1$, $\beta_2 = 2$, $c_2 = 1$, and $c_1 = 1.313$. **a** Loss functions σ . **b** Influence functions φ

The conventional MSE function corresponds to $\sigma(\epsilon_i) = \frac{1}{2}\epsilon_i^2$ and $\varphi(\epsilon_i; \beta) = \epsilon_i$. To suppresses the effect of large errors, loss functions used for robust learning are defined such that $\varphi(\epsilon_i; \beta)$ is sublinear.

Example 2.4 The loss functions given above and their respective influence functions are illustrated in Fig. 2.5.

τ -estimator [137] can be viewed as an M -estimator with an adaptive bounded influence function $\varphi(\cdot)$ given by the weighted average of two functions $\varphi_1(\cdot)$ and $\varphi_2(\cdot)$, with $\varphi_1(\cdot)$ corresponding to a very robust estimate and $\varphi_2(\cdot)$ to a highly efficient estimate. τ -estimator simultaneously has a high breakdown point and a high efficiency under Gaussian errors.

When the initial weights are not properly selected, the loss functions may not be able to correctly discriminate against the outliers. The selection of β is also a problem, and one approach is to select β as the median of the absolute deviation (MAD)

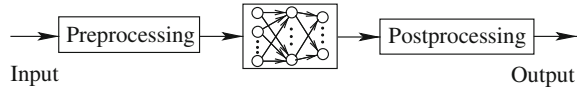
$$\beta = c \times \text{median}(|\epsilon_i - \text{median}(\epsilon_i)|) \quad (2.22)$$

with c chosen as 1.4826 [77]. Some other methods for selecting β are based on using the median of all errors [77], or counting out a fixed percentage of points as outliers [35].

2.6 Neural Network Processors

A typical architecture of a neural network processor is illustrated in Fig. 2.6. It is composed of three components: input preprocessing, a neural network for performing inversion, and output postprocessing. Input preprocessing is used to remove redundant and/or irrelevant information in order to achieve a small network and to reduce the dimensionality of the signal parameter space, thus improving the generalization

Fig. 2.6 Architecture of a neural network processor



capability of the network. Postprocessing the output of the network generates the desired information.

Preprocessing is to transform the raw data into a new representation before being presented to a neural network. If the input data is preprocessed at the training stage, accordingly at the generalizing stage, the input data also needs to be preprocessed before being passed on to the neural network. Similarly, if the output data is preprocessed at the training stage, the network output at the generalization stage is also required to be postprocessed to generate the target output corresponding to the raw output patterns.

For high-dimensional data, dimensionality reduction is the key to cope with the curse of dimensionality. Preprocessing has a significant influence on the generalization performance of a neural network. This process removes the redundancy in the input space and reduces the space of the input data, thus usually resulting in a reduction in the amount or the dimensionality of the input data. This helps to alleviate the problem of the curse of dimensionality. A network with preprocessed inputs may be constrained by a smaller dataset, and thus one needs only to train a small network, which also achieves a better generalization capability.

Preprocessing usually takes the form of linear or nonlinear transformation of the raw input data to generate input data for the network. It can also be based on the prior knowledge of the network architecture, or the problem itself. When preprocessing removes redundant information in the input data, it also results in a loss of information. Thus, preprocessing should retain as much relevant information as possible.

The data are sometimes called *features*, and preprocessing for input raw data can be either feature selection or feature extraction. Feature selection concentrates on selecting from the original set of features a smaller subset of salient features, while feature extraction is to combine the original features in such a way as to produce a new reduced set of salient features. The raw data may be orders of magnitude in range, and linear scaling and data whitening of the raw data are usually employed as a preprocessing step.

When some examples of the raw data suffer from missing components, one simple treatment is to discard those examples from the dataset. This, however, is applicable only when the data is abundant, the percentage of examples with missing components is small, and the mechanism for loss of data is independent of the data itself. However, this may lead to a biased subset of the data. Methods should replace the missing value with a value substituted according to various criteria. For function approximation problems, one can represent any variable with a missing value as a regression over the other variables using the available data, and then find the missing value by interpolating the regression function. For density estimation problems,

the ML solution to problems with the missing data can be found by applying an expectation-maximization (EM) algorithm.

Feature extraction reduces the dimension of the features by orthogonal transforms. The extracted features do not have any physical meaning. In comparison, feature selection decreases the size of the feature set or reduces the dimension of the features by discarding the raw information according to a criterion.

Feature Selection

Feature selection is to select the best subset or the best subspace of the features out of the original set, since irrelevant features degrade the performance. A criterion is required to evaluate each subset of the features so that an optimum subset can be selected. The selection criterion should be the same as that for assessing the complete system, such as the MSE criterion for function approximation and the misclassification rate for classification. Theoretically, the global optimum subset of the features can only be selected by an exhaustive search of all the possible subsets of the features.

Feature selection algorithms can be categorized as either filter or wrapper approaches. During the process of feature selection, the generalization ability of a subset of features needs to be estimated. This type of feature selection is called a wrapper method [79]. The problem of searching the best r variables is solved by means of a greedy algorithm based on backward selection [79]. The filter approach basically pre-selects the features, and then applies the selected feature subset to the clustering algorithm. Filter-based greedy algorithms using the sequential selection of the feature with the best criterion value are computationally more efficient than wrappers. In general, the wrapper method outperforms the filter method, but at the expense of training a large number of classifiers.

Some nonexhaustive search methods such as the branch and bound procedure, sequential forward selection, and sequential backward elimination are discussed in [22]. Usually, backward selection is slower but is more stable in selecting optimal features than forward selection. Backward selection starts from all the features and deletes one feature at a time, which deteriorates the selection criterion the least, until the selection criterion reaches a specified value. In contrast, forward selection starts from an empty set of features and adds one feature at a time that improves the selection criterion the most.

Mutual information based feature selection is a common method for feature selection [15, 54]. The mutual information measures the arbitrary dependence between random variables, whereas linear relations, such as the correlation-based methods, are prone to mistakes. By calculating the mutual information, the importance levels of the features are ranked based on their ability to maximize the evaluation criterion. Relevant inputs are found by estimating the mutual information between the inputs and the desired outputs. The normalized mutual information feature selection [54] does not require a user-defined parameter.

Feature Extraction

Feature extraction is usually conducted by using orthogonal transforms, though the Gram–Schmidt orthonormalization (GSO) is more suitable for feature selection.

This is due to the fact that the physically meaningless features in the Gram–Schmidt space can be linked back to the same number of variables of the measurement space, resulting in no dimensionality reduction. In situations where the features are used for pattern understanding and analysis, the GSO transform provides a good option.

The advantage of employing an orthogonal transform is that the correlations among the candidate features are decomposed so that the significance of the individual features can be evaluated independently. PCA is a well-known orthogonal transform. Taking all the data into account, PCA computes vectors that have the largest variance associated with them. The generated PCA features may not have clear physical meanings. Dimensionality reduction is achieved by dropping the variables with insignificant variance. Projection pursuit [57] is a general approach to feature extraction, which extracts features by repeatedly choosing projection vectors and then orthogonalizing.

PCA is often used to select inputs, but it is not always useful, since the variance of a signal is not always related to the importance of the variable, for example, for non-Gaussian signals. An improvement on PCA is provided by nonlinear generalizations of PCA, which extend the ability of PCA to incorporate nonlinear relationships in the data. ICA can extract the statistically independent components from the input dataset. It is to estimate the mutual information between the signals by adjusting the estimated matrix to give outputs that are maximally independent [8]. The dimensions to remove are those that are independent of the output. LDA searches for those vectors in the underlying space that best discriminate among the classes (rather than those that best describe the data). In [83], the proposed scheme for linear feature extraction in classification is based on the maximization of the mutual information between the features extracted and the classes.

For time/frequency-continuous signal systems such as speech-recognition systems, the fixed time-frequency resolution FFT power spectrum, and the multiresolution discrete wavelet transform and wavelet packets are usually used for feature extraction. The features used are chosen from the Fourier or wavelet coefficients having high energy. The cepstrum and its time derivative remain a most commonly used feature set [103]. These features are calculated by taking the discrete cosine transform (DCT) of the logarithm of the energy at the output of a Mel filter and are commonly called *Mel frequency cepstral coefficients (MFCC)*. In order to have the temporal information, the first and second time derivatives of the MFCC are taken.

2.7 Criterion Functions

The MSE is by far the most popular measure of error. This error measure ensures that a large error receives much greater attention than a small error. The MSE criterion is optimal and results in an ML estimation of the weights if the distributions of the feature vectors are Gaussian [120]. This is desired for most applications. In some situations, other error measures such as the mean absolute error, maximum absolute error, and median squared error, may be preferred.

The logarithmic error function, which takes the form of the instantaneous relative entropy or Kullback–Leibler divergence criterion, has some merits over the MSE function [16]

$$E_p(\mathbf{W}) = \frac{1}{2} \sum_{i=1}^{J_M} \left[(1 + y_{p,i}) \ln \left(\frac{1 + y_{p,i}}{1 + \hat{y}_{p,i}} \right) + (1 - y_{p,i}) \ln \left(\frac{1 - y_{p,i}}{1 - \hat{y}_{p,i}} \right) \right] \quad (2.23)$$

for the tanh activation function, where $y_{p,i} \in (-1, 1)$. For the logistic activation function, the criterion can be written as [92]

$$E_p(\mathbf{W}) = \frac{1}{2} \sum_{i=1}^{J_M} \left[y_{p,i} \ln \left(\frac{y_{p,i}}{\hat{y}_{p,i}} \right) + (1 - y_{p,i}) \ln \left(\frac{1 - y_{p,i}}{1 - \hat{y}_{p,i}} \right) \right], \quad (2.24)$$

where $y_{p,i} \in (0, 1)$. In the latter case, $y_{p,i}$, $\hat{y}_{p,i}$, $1 - y_{p,i}$, and $1 - \hat{y}_{p,i}$ are regarded as probabilities. These criteria take zero only when $y_{p,i} = \hat{y}_{p,i}$, $i = 1, \dots, J_M$, and are strictly positive otherwise. Another criterion function obtained by simplifying (2.24) via omitting the constant terms related to the patterns is [131]

$$E_p(\mathbf{W}) = -\frac{1}{2} \sum_{i=1}^{J_M} [y_{p,i} \ln \hat{y}_{p,i} + (1 - y_{p,i}) \ln (1 - \hat{y}_{p,i})]. \quad (2.25)$$

The problem of loading a set of training examples onto a neural network is NP-complete [25, 130]. As a consequence, existing algorithms cannot be guaranteed to learn the optimal solution in polynomial time. In the case of one neuron, the logistic function paired with the MSE function can lead to $\left(\frac{N}{J_1}\right)^{J_1}$ local minima, for N training patterns and an input dimension of J_1 [6], while with the entropic error function, the error function is convex and thus has only one minimum [16, 131]. The use of the entropic error function considerably reduces the total number of local minima.

The BP algorithm derived from the entropy criteria can partially solve the flat-spot problem. These criteria do not add computation load to calculate the error function. They, however, remarkably reduce the training time, and alleviate the problem of getting stuck at local minima by reducing the density of local minima [92]. Besides, the entropy-based BP is well suited to probabilistic training data, since it can be viewed as learning the correct probabilities of a set of hypotheses represented by the outputs of the neurons.

Traditionally, classification problems are learned through error backpropagation by providing a vector of hard 0/1 target values to represent the class label of a particular pattern. Minimizing an error function with hard target values tends to a saturation of weights, leading to overfitting. The magnitude of the weights plays a more important role in generalization than the number of hidden nodes [12]. Overfitting might be reduced by keeping the weights smaller.

The cross-entropy cost function can also be derived from the ML principle

$$E_{CE} = - \sum_{i=1}^N \sum_{k=1}^C t_{k,i} \ln(y_{k,i}) \quad (2.26)$$

for training set $\{\mathbf{x}_i, \mathbf{t}_i\}$, C classes and N samples, and $t_{k,i} \in \{0, 1\}$.

Marked reductions on convergence rates and density of local minima are observed due to the characteristic steepness of the cross-entropy function [92, 131]. As a function of the absolute errors, MSE tends to produce large relative errors for small output values. As a function of the relative errors, cross-entropy is expected to estimate more accurately small probabilities [62, 72, 131]. When a neural network is trained using MSE or cross-entropy minimization, its outputs approximate the posterior probabilities of class membership. Thus, in the presence of large datasets, it tends to produce optimal solutions in the Bayes sense. However, minimization of the error function does not necessarily imply misclassification minimization in practice. Suboptimal solutions may occur due to flat regions in weight space. Thus, minimization of these error functions does not imply misclassification minimization.

The MSE function can be obtained by the ML principle assuming the independence and Gaussianity of the target data. However, the Gaussianity assumption of the target data in classification is not valid, due to its discrete nature of class labels. Thus, the MSE function is not the most appropriate one for data classification problems. Nevertheless, when using a 1-out-of- C coding scheme for the targets, with large N and a number of samples in each class, the MSE trained outputs of the network approximate the posterior probabilities of the class membership [62]. The cross-entropy error function and other entropy-based functions are suitable for training neural network classifiers, because when interpreting the outputs as probabilities this is the optimal solution.

Classification-based (CB) error functions [109] heuristically seek to directly minimize classification error by backpropagating network error only on misclassified patterns. In so doing, they perform relatively minimal updates to network parameters in order to discourage premature weight saturation and overfitting. CB3 is a CB approach that learns the error function to be used while training [110]. This is accomplished by learning pattern confidence margins during training, which are used to dynamically set output target values for each training pattern. In fact, CB3 saves time by omitting the error backpropagation step for correctly classified patterns with sufficient output confidences. The number of epochs required to converge is similar for CB3 and cross-entropy training, generally about half as many epochs required for MSE training.

The MSE criterion can be generalized into the Minkowski- r metric [64]

$$E_p = \frac{1}{r} \sum_{i=1}^{J_M} |\hat{y}_{p,i} - y_{p,i}|^r. \quad (2.27)$$

When $r = 1$, the metric is called the *city block* metric. The Minkowski- r metric corresponds to the MSE criterion for $r = 2$. A small value of r ($r < 2$) reduces the influence of large deviations, thus it can be used in the case of outliers. In contrast, a large r weights large deviations, and generates a better generation surface when the noise is absent in the data or when the data clusters in the training set are compact.

A generalized error function embodying complementary features of other functions, which can emulate the behavior of other error functions by adjustment of a single real-valued parameter, is proposed in [129]. Many other criterion functions can be used for deriving learning algorithms, including those based on robust statistics [77] or regularization [105].

2.8 Computational Learning Theory

Machine learning makes predictions about the unknown underlying model based on a training set drawn from hypotheses. Due to the finite training set, learning theory cannot provide absolute guarantees of performance of the algorithms. The performance of learning algorithms is commonly bounded by probabilistic terms. Computational learning theory is a statistical tool for the analysis of machine learning algorithms, that is, for characterizing learning and generalization. Computational learning theory addresses the problem of optimal generalization capability for supervised learning. Two popular formalisms of approaches to computational learning theory are the *VC theory* [143] and the probably approximately correct (PAC) learning [142]. Both approaches are nonparametric and distribution-free learning models.

The VC theory [143], known as the statistical learning theory, is a dependency-estimation method with finite data. Necessary and sufficient conditions for consistency and fast convergence are obtained based on the empirical risk minimization (ERM) principle. Uniform convergence for a given class of approximating functions is associated with the capacity of the function class considered [143]. The VC dimension of a function class quantifies its classification capabilities [143]. It indicates the cardinality of the largest set for which all possible binary-valued classifications can be obtained using functions from the class. The capacity and complexity of the function class is measured in terms of the VC dimension. The ERM principle has been practically applied in the SVM [146]. The VC theory provides a general measure of complexity, and gives associated bounds on the optimism.

PAC learning [142] aims to find a hypothesis that is a good approximation to an unknown target concept with a high probability. The PAC learning paradigm is intimately associated with the ERM principle. A hypothesis that minimizes the empirical error, based on a sufficiently large sample, will approximate the target concept with a high probability. The generalization ability of network training can be established estimating the VC dimension of neural architectures. Boosting [122] is a PAC learning-inspired method for supervised learning.

2.8.1 Vapnik-Chervonenkis Dimension

The VC dimension is a combinatorial characterization of the diversity of functions that can be computed by a given neural architecture. It can be viewed as a generalization of the concept of capacity first introduced by Cover [44]. The VC dimension can be regarded as a measure of the capacity or expressive power of a network. VC dimension is the measure of model complexity (capacity) used in VC theory. For linear estimators, the VC dimension is equivalent to the number of model parameters, but is hard to obtain for other types of estimators.

Definition 2.1 (VC dimension) A subset \mathcal{S} of the domain \mathcal{X} is shattered by a class of functions or neural network \mathcal{N} if every function $f : \mathcal{S} \rightarrow \{0, 1\}$ can be computed on \mathcal{N} . The VC dimension of \mathcal{N} is defined as the maximal size of a set $\mathcal{S} \subseteq \mathcal{X}$ that is shattered by \mathcal{N}

$$\dim_{VC}(\mathcal{N}) = \max \{|\mathcal{S}| \mid \mathcal{S} \subseteq \mathcal{X} \text{ is shattered by } \mathcal{N}\}, \quad (2.28)$$

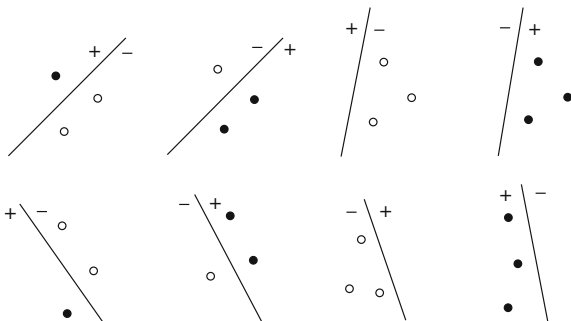
where $|\mathcal{S}|$ denotes the cardinality of \mathcal{S} .

For example, for a neural network with the relation $f(\mathbf{x}, \mathbf{w}, \theta) = \text{sgn}(\mathbf{w}^T \mathbf{x} + \theta)$, it can shatter at most any three points in \mathcal{X} , thus its VC dimension is 3. This is shown in Fig. 2.7. The points are in general position, that is, they are linearly independent.

A hard-limiter function with threshold θ_0 is typically used as the activation function for binary neurons. The basic function of the McCulloch–Pitts neuron has a linear relation applied by a threshold operation, hence called a *linear threshold gate (LTG)*. A neural network with LTG has a VC dimension of $O(N_w \log N_w)$ [14], where N_w is the number of weights in a network. The VC dimension has been generalized for neural networks with real-valued output, and the VC dimension of various neural networks has been studied in [14].

The VC dimension can be used to estimate the number of training examples for a good generalization capability. The Boolean VC dimension of a neural network \mathcal{N} , written $\dim_{BVC}(\mathcal{N})$, is defined as the VC dimension of the class of Boolean functions

Fig. 2.7 Shatter any three points in \mathcal{X} into two classes



that is computed by \mathcal{N} . The VC dimension is a property of a set of functions $\{f(\alpha)\}$, and can be defined for various classes of function f .

The VC dimension for the set of functions $\{f(\alpha)\}$ is defined as the maximum number of training points that can be shattered by $\{f(\alpha)\}$. If the VC dimension is d , then there exists at least one set of d points that can be shattered, but in general it will not be true that every set of d points can be shattered.

Feedforward networks with threshold and logistical activation functions have VC dimensions of $O(N_w \ln N_w)$ [17] and $O(N_w^2)$ [80], respectively. Sufficient sample sizes are, respectively, estimated by using the PAC paradigm and the VC dimension for feedforward networks with sigmoidal neurons [127] and feedforward networks with LTGs [17]. These bounds on sample sizes are dependent on the error rate of hypothesis ϵ and the probability of failure δ . A practical size of the training set for good generalization is $N = O\left(\frac{N_w}{\epsilon}\right)$ [70], where ϵ specifies the accuracy. For example, for an accuracy level of 90 %, $\epsilon = 0.1$.

It is not possible to obtain the analytic estimates of the VC dimension in most cases. Hence, a proposal is to measure the VC dimension of an estimator experimentally by fitting the theoretical formula to a set of experimental measurements of the frequency of errors on artificially generated datasets of varying sizes [145]. However, with this approach it may be difficult to obtain an accurate estimate of the VC dimension due to the variability of random samples in the experimental procedure. In [126], this problem is addressed by proposing an improved design procedure for specifying the measurement points (i.e., the sample size and the number of repeated experiments at a given sample size). This leads to a nonuniform design structure as opposed to the uniform design structure used in [145]. The proposed optimized design structure leads to a more accurate estimation of the VC dimension using the experimental procedure. A more accurate estimation of VC dimension leads to improved complexity control using analytic VC-generalization bounds and, hence, better prediction accuracy.

2.8.2 Empirical Risk-Minimization Principle

Assume that a set of N samples, $\{(\mathbf{x}_i, y_i)\}$, are independently drawn and identically distributed (iid) samples from some unknown probability distribution $p(\mathbf{x}, y)$. Assume a machine defined by a set of possible mappings $\mathbf{x} \rightarrow f(\mathbf{x}, \alpha)$, where α contains adjustable parameters. When α is selected, the machine is called a *trained machine*.

The expected risk is the expectation of the generalization error for a trained machine, and is given by

$$R(\alpha) = \int L(y, f(\mathbf{x}, \alpha)) dp(\mathbf{x}, y), \quad (2.29)$$

where $L(y, f(\mathbf{x}, \alpha))$ is the loss function, measuring the discrepancy between the output pattern y and the output of the learning machine $f(\mathbf{x}, \alpha)$. The loss function can be defined in different forms for different purposes:

$$L(y, f(\mathbf{x}, \alpha)) = \begin{cases} 0, & y = f(\mathbf{x}, \alpha) \\ 1, & y \neq f(\mathbf{x}, \alpha) \end{cases} \quad (\text{for classification}), \quad (2.30)$$

$$L(y, f(\mathbf{x}, \alpha)) = (y - f(\mathbf{x}, \alpha))^2 \quad (\text{for regression}), \quad (2.31)$$

$$L(p(\mathbf{x}, \alpha)) = -\ln p(\mathbf{x}, \alpha) \quad (\text{for density estimation}). \quad (2.32)$$

The empirical risk $R_{\text{emp}}(\alpha)$ is defined to be the measured mean error on a given training set

$$R_{\text{emp}}(\alpha) = \frac{1}{N} \sum_{i=1}^N L(y_i, f(\mathbf{x}_i, \alpha)). \quad (2.33)$$

The ERM principle aims to approximate the loss function by minimizing the empirical risk (2.33) instead of the risk (2.29), with respect to model parameters.

When the loss function takes the value 0 or 1, with probability $1 - \delta$, there is the upper bound called the *VC bound* [146]:

$$R(\alpha) \leq R_{\text{emp}}(\alpha) + \sqrt{\frac{d \left(\ln \frac{2N}{d} + 1 \right) - \ln \frac{\delta}{4}}{N}}, \quad (2.34)$$

where d is the VC dimension of the machine. The second term on the right-hand side is called the *VC confidence*, which monotonically increases with increasing d . Reducing d leads to a better upper bound on the actual error. The VC confidence depends on the class of functions, whereas the empirical risk and actual risk depend on the particular function obtained by the training procedure.

For regression problems, a practical form of the VC bound is used [147]:

$$R(d) \leq R_{\text{emp}}(d) \left(1 - \sqrt{p - p \ln p + \frac{\ln n}{2n}} \right)^{-1}, \quad (2.35)$$

where $p = \frac{d}{n}$ and d is the VC dimension. The VC bound (2.35) is a special case of the general analytical bound [146] with appropriately chosen practical values of theoretical constants.

The principle of structural risk minimization (SRM) minimizes the risk functional with respect to both the empirical risk and the VC dimension of the set of functions; thus, it aims to find the subset of functions that minimizes the bound on the actual risk. The SRM principle is crucial to obtain good generalization performances for a variety of learning machines, including SVMs. It finds the function that achieves the minimum of the guaranteed risk for the fixed amount of data. To find the guaranteed risk, one has to use bounds, e.g., VC bound, on the actual risk. Empirical comparisons

between AIC, BIC and the SRM method are presented for regression problems in [39], based on VC theory. VC-based model selection consistently outperforms AIC for all the datasets, whereas the SRM and BIC methods show similar predictive performance.

Function Approximation, Regularization, and Risk Minimization

Classical statistics and function approximation/regularization rely on the true model that underlies generated data. In contrast, VC learning theory is based on the concept of risk minimization, and does not use the notion of a true model. The distinction between the three learning paradigms becomes blurred when they are used to motivate practical learning algorithms. Least-squares (LS) minimization for function estimation can be derived using the parametric estimation approach via ML arguments under Gaussian noise assumptions, and it can alternatively be introduced under the risk minimization approach. SVM methodology was originally developed in VC-theory, and later re-introduced in the function approximation/regularization setting [68]. An important conceptual contribution of the VC approach states that generalization (learning) with finite samples may be possible even if accurate function approximation is not [41]. The regularization program does not yield good generalization for finite sample estimation problems.

In the function approximation theory, the goal is to estimate an unknown true target function in regression problems, or posterior probability $P(y|\mathbf{x})$ in classification problems. In VC theory, it is to find the target function that minimizes prediction risk or achieves good generalization. That is, the result of VC learning depends on (unknown) input distribution, while that of function approximation does not. The important concept of margin was originally introduced under the VC approach, and later explained and interpreted as a form of regularization. However, the notion of margin is specific to SVM, and it does not exist under the regularization framework. Any of the methodologies (including SRM and SVM) can be regarded as a special case of regularization.

2.8.3 Probably Approximately Correct Learning

The PAC learning paradigm is concerned with learning from examples of a target function called *concept*, by choosing from a set of functions known as the *hypothesis space*, a function that is meant to be a good approximation to the target.

Let \mathcal{C}_n and \mathcal{H}_n , $n \geq 1$, respectively, be a set of target concepts and a set of hypotheses over the instance space $\{0, 1\}^n$, where $\mathcal{C}_n \subseteq \mathcal{H}_n$ for $n \geq 1$. When there exists a polynomial-time learning algorithm that achieves low error with high confidence in approximating all concepts in a class $\mathcal{C} = \{\mathcal{C}_n\}$ by the hypothesis space $\mathcal{H} = \{\mathcal{H}_n\}$ if enough training data is available, the class of concepts \mathcal{C} is said to be *PAC learnable* by \mathcal{H} or simply PAC learnable. Uniform convergence of the empirical

error of a function towards the real error on all possible inputs guarantees that all training algorithms that yield a small training error are PAC. A function class is PAC learnable if and only if the capacity in terms of the VC dimension is finite.

In this framework, we are given a set of inputs and a hypothesis space of functions that maps the inputs onto $\{0, 1\}$. Assume that there is an unknown but usually fixed probability distribution on the inputs, and the aim is to find a good approximation to a particular target concept from the hypothesis space, given only a random sample of the training examples and the value of the target concept on these examples.

The sample complexity of a learning algorithm, N_{PAC} , is defined as the smallest number of samples required for learning \mathcal{C} by \mathcal{H} , that achieve a given approximation accuracy ϵ with a probability $1 - \delta$. Any consistent algorithm that learns \mathcal{C} by \mathcal{H} has a sample complexity with the upper bound [5, 69]

$$N_{\text{PAC}} \leq \frac{1}{\epsilon(1 - \sqrt{\epsilon})} \left(2 \dim_{\text{VC}}(\mathcal{H}_n) \ln \frac{6}{\epsilon} + \ln \frac{2}{\delta} \right), \quad \forall 0 < \delta < 1. \quad (2.36)$$

In other words, with probability of at least $1 - \delta$, the algorithm returns a hypothesis $h \in \mathcal{H}_n$ with an error less than ϵ .

In terms of the cardinality of \mathcal{H}_n , denoted $|\mathcal{H}_n|$, it can be shown [69, 144] that the sample complexity is upper bounded by

$$N_{\text{PAC}} \leq \frac{1}{\epsilon} \left(\ln |\mathcal{H}_n| + \ln \frac{1}{\delta} \right). \quad (2.37)$$

For most hypothesis spaces on Boolean domains, the second bound gives a better bound. On the other hand, most hypothesis spaces on real-valued attributes are infinite, so only the first bound is applicable. PAC learning is particularly useful for obtaining upper bounds on sufficient training sample size. Linear threshold concepts (perceptrons) are PAC learnable on both Boolean and real-valued instance spaces [69].

2.9 No-Free-Lunch Theorem

Before the no-free-lunch theorem [150] was proposed, people intuitively believed that there exists some universally beneficial algorithms for search, and many people actually made efforts to design some algorithms. The no-free-lunch theorem asserts that there is no universally beneficial algorithm.

The no-free-lunch theorem states that no search algorithm is better than another in locating an extremum of a cost function when averaged over the set of all possible discrete functions. That is, all search algorithms achieve the same performance as random enumeration, when evaluated over the set of all functions.

Theorem 2.1 (No-free-lunch theorem) *Given the set of all functions \mathcal{F} and a set of benchmark functions \mathcal{F}_1 , if algorithm A_1 is better on average than algorithm A_2 on \mathcal{F}_1 , then algorithm A_2 must be better than algorithm A_1 on $\mathcal{F} - \mathcal{F}_1$.*

The performance of any algorithm is determined by the knowledge concerning the cost function. Thus, it is meaningless to evaluate the performance of an algorithm without specifying the prior knowledge. Practical problems always contain priors such as smoothness, symmetry, and i.i.d. samples. For example, although neural networks are usually deemed a powerful approach for classification, they cannot solve all classification problems. For some arbitrary classification problems, other methods may be efficient.

The no-free-lunch theorem was later extended to coding methods, early stopping [31], avoidance of overfitting, and noise prediction [90]. Again, it has been asserted that no one method is better than the others for all problems.

Following the no-free-lunch theorem, the inefficiency of leave-one-out crossvalidation was demonstrated on a simple problem in [158]. In response to [158], in [63] the strict leave-one-out crossvalidation was shown to yield the expected results on this simple problem, thus leave-one-out crossvalidation is not subject to the no-free-lunch criticism [63]. Nonetheless, it is concluded in [114] that the statistical tests are preferable to crossvalidation for linear as well as for nonlinear model selection.

2.10 Neural Networks as Universal Machines

The power of neural networks stems from their representation capability. On the one hand, feedforward networks are proved to offer the capability of universal function approximation. On the other hand, recurrent networks using the sigmoidal activation function are Turing equivalent [128] and simulates a universal Turing machine; Thus, recurrent networks can compute whatever function any digital computer can compute.

2.10.1 Boolean Function Approximation

Feedforward networks with binary neurons can be used to represent logic or Boolean functions. In binary neural networks, the input and output values for each neuron are Boolean variables, denoted by binary (0 or 1) or bipolar (-1 or $+1$) representation. For J_1 independent Boolean variables, there are 2^{J_1} combinations of these variables. This leads to a total of $2^{2^{J_1}}$ different Boolean functions of J_1 variables. An LTG can discriminate between two classes.

The function counting theorem [44, 65] gives the number of linearly separable dichotomies of m points in general position in \mathcal{R}^n . It essentially estimates the separating capability of an LTG.

Theorem 2.2 (Function counting theorem) *The number of linearly separable dichotomies of m points in general position in R^n is*

$$C(m, n) = \begin{cases} 2 \sum_{i=0}^n \binom{m-1}{i}, & m > n + 1 \\ 2^m, & m \leq n + 1 \end{cases}. \quad (2.38)$$

A set of m points in R^n is said to be in *general position* if every subset of m or fewer points is linearly independent.

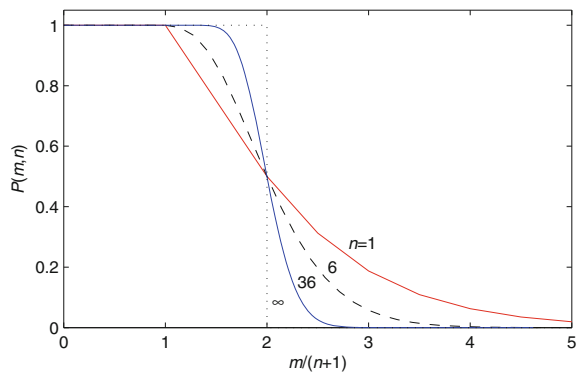
The total number of possible dichotomies of m points is 2^m . Under the assumption of 2^m equiprobable dichotomies, the probability of a single LTG with n inputs to separate m points in general position is given by

$$P(m, n) = \frac{C(m, n)}{2^m} = \begin{cases} \frac{2}{2^m} \sum_{i=0}^n \binom{m-1}{i}, & m > n + 1 \\ 1, & m \leq n + 1 \end{cases}. \quad (2.39)$$

The fraction $P(m, n)$ is the probability of linear dichotomy. Thus, if $\frac{m}{n+1} \leq 1$, $P = 1$; if $1 < \frac{m}{n+1} < 2$ and $n \rightarrow \infty$, $P \rightarrow 1$. At $\frac{m}{n+1} = 2$, $P = \frac{1}{2}$. Usually, $m = 2(n + 1)$ is used to characterize the statistical capability of a single LTG. Equation (2.39) is plotted in Fig. 2.8.

A three-layer ($J_1 - 2^{J_1 - 1}$) feedforward LTG network can represent any Boolean function with J_1 arguments [45, 93]. To realize an arbitrary function $f : R^{J_1} \rightarrow \{0, 1\}$ defined on N arbitrary points in R^{J_1} , the lower bound for the number of hidden nodes is derived as $O\left(\frac{N}{J_1 \log_2 \frac{N}{J_1}}\right)$ for $N \geq 3J_1$ and $J_1 \rightarrow \infty$ [16, 65]; for N points in general position, the lower bound is $\frac{N}{2J_1}$ when $J_1 \rightarrow \infty$ [65]. Networks with two or more hidden layers are found to be potentially more size efficient than networks with a single hidden layer [65].

Fig. 2.8 The probability of linear dichotomy of m points in n dimensions



Binary Radial Basis Function

For three-layer feedforward networks, if the activation function of the hidden neurons is selected as the binary RBF or generalized binary RBF and the output neurons are selected as LTGs, one obtains binary or generalized binary RBF networks. Binary or generalized binary RBF network can be used for the mapping of Boolean functions.

The parameters of the generalized binary RBF neuron are the center $\mathbf{c} \in R^n$ and the radius $r \geq 0$. The activation function $\phi : R^n \rightarrow \{0, 1\}$ is defined by

$$\phi(\mathbf{x}) = \begin{cases} 1, & \|\mathbf{x} - \mathbf{c}\|_{\mathbf{A}} \leq r \\ 0, & \text{otherwise} \end{cases}, \quad (2.40)$$

where \mathbf{A} is any real, symmetric and positive-definite matrix, and $\|\cdot\|_{\mathbf{A}}$ is the weighted Euclidean norm. When \mathbf{A} is the identity matrix \mathbf{I} , the neuron becomes a binary RBF neuron.

Every Boolean function computed by the LTG can also be computed by any generalized binary RBF neuron, and generalized binary RBF neurons are more powerful than LTGs [58]. As an immediate consequence, in any neural network, any LTG that receives only binary inputs can be replaced by a generalized binary RBF neuron having any norm, without any loss of the computational power of the neural network.

Given a J_1 - J_2 -1 feedforward network, whose output neuron is an LTG; we denote the network as $\mathcal{N}_1, \mathcal{N}_2$, and \mathcal{N}_3 , when the J_2 hidden neurons are respectively selected as LTGs, binary RBF neurons, and generalized binary RBF neurons. The VC dimensions of the three networks have the relation [58]

$$\dim_{\text{BVC}}(\mathcal{N}_1) = \dim_{\text{BVC}}(\mathcal{N}_2) \leq \dim_{\text{BVC}}(\mathcal{N}_3). \quad (2.41)$$

When $J_1 \geq 3$ and $J_2 \leq \frac{2^{J_1+1}}{J_1^2 + J_1 + 2}$, the lower bound for the three neural networks is given as [13, 58]

$$\dim_{\text{BVC}}(\mathcal{N}_1) = J_1 J_2 + 1. \quad (2.42)$$

2.10.2 Linear Separability and Nonlinear Separability

Definition 2.2 (Linearly separable) Assume that there is a set \mathcal{X} of N patterns \mathbf{x}_i of J_1 dimensions, each belonging to one of two classes \mathcal{C}_1 and \mathcal{C}_2 . If there is a hyperplane that separates all the samples of \mathcal{C}_1 from \mathcal{C}_2 , then such a classification problem is said to be linearly separable.

A single LTG can realize linearly separable dichotomy function, characterized by a linear separating surface (hyperplane)

$$\mathbf{w}^T \mathbf{x} + w_0 = 0, \quad (2.43)$$

where \mathbf{w} is a J_1 -dimensional vector and w_0 is a bias toward the origin. For a pattern, if $\mathbf{w}^T \mathbf{x} + w_0 > 0$, it belongs to \mathcal{C}_1 ; if $\mathbf{w}^T \mathbf{x} + w_0 < 0$, it belongs to \mathcal{C}_2 .

Definition 2.3 (φ separable) A dichotomy $\{\mathcal{C}_1, \mathcal{C}_2\}$ of set \mathcal{X} is said to be φ -separable if there exists a mapping $\varphi : R^{J_1} \rightarrow R^{J_2}$ that satisfies a separating surface [44]

$$\mathbf{w}^T \varphi(\mathbf{x}) = 0 \quad (2.44)$$

such that $\mathbf{w}^T \varphi(\mathbf{x}) > 0$ if $\mathbf{x} \in \mathcal{C}_1$ and $\mathbf{w}^T \varphi(\mathbf{x}) < 0$ if $\mathbf{x} \in \mathcal{C}_2$. Here \mathbf{w} is a J_2 -dimensional vector.

A linearly inseparable dichotomy can become nonlinearly separable. As shown in Fig. 2.9, the two linearly inseparable dichotomies become φ -separable.

The nonlinearly separable problem can be realized by using a polynomial threshold gate, which changes the linear term in the LTG into high-order polynomials. The function counting theorem is applicable to polynomial threshold gates; and it still holds true if the set of m points is in general position in φ -space, that is, the set of m points is in φ -general position.

Example 2.5 Some examples of linearly separable classes and linearly inseparable classes in two-dimensional space are illustrated in Fig. 2.9. (a) Two linearly separable classes with $x_1 - x_2 = 0$ as the delimiter. (b) and (c) are linearly inseparable classes, where (b) is the exclusive-or problem. Note that the linearly inseparable classification problems in cases (b) and (c) become nonlinearly separable, when the separating surfaces are ellipses.

Higher-order neurons (or Σ - Π units) are simple but powerful extensions of linear neuron models. They introduce the concept of nonlinearity by incorporating monomials, that is, products of input variables, as a hidden layer. Higher-order neurons with k monomials in n variables are shown to have VC dimension at least $nk + 1$ [123].

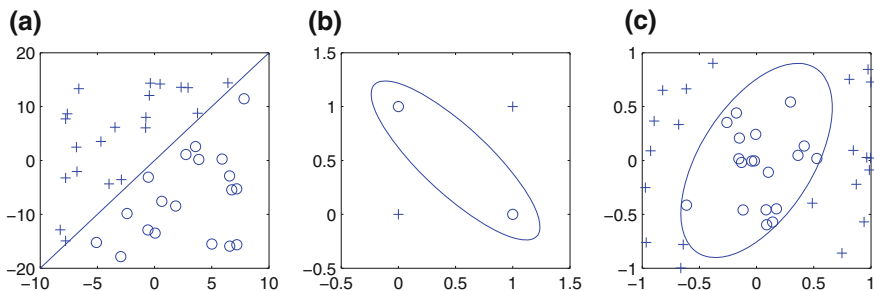


Fig. 2.9 a Linearly separable, b linearly inseparable, and c nonlinearly separable classification in two-dimensional space. Dots and circles denote patterns of different classes

2.10.3 Continuous Function Approximation

A three-layer feedforward network with a sufficient number of hidden units can approximate any continuous function to any degree of accuracy. This is guaranteed by Kolmogorov's theorem [71, 81].

Theorem 2.3 (Kolmogorov) *Any continuous real-valued function $f(x_1, \dots, x_n)$ defined on $[0, 1]^n$, $n \geq 2$, can be represented in the form*

$$f(x_1, \dots, x_n) = \sum_{j=1}^{2n+1} h_j \left(\sum_{i=1}^n \psi_{ij}(x_i) \right), \quad (2.45)$$

where h_j and ψ_{ij} are continuous functions of one variable, and ψ_{ij} are monotonically increasing functions independent of f .

Kolmogorov's theorem is the famous solution to Hilbert's 13th problem. According to Kolmogorov's theorem, a continuous multivariate function on a compact set can be expressed using superpositions and compositions of a finite number of single-variable functions. Based on Kolmogorov's theorem, Hecht-Nielsen provided a theorem that is directly related to neural networks [71].

Theorem 2.4 (Hecht-Nielsen) *Any continuous real-valued mapping $f: [0, 1]^n \rightarrow \mathbb{R}^m$ can be approximated to any degree of accuracy by a feedforward network with n input nodes, $2n + 1$ hidden units, and m output units.*

The Weierstrass theorem asserts that any continuous real-valued multivariate function can be approximated to any accuracy using a polynomial. The Stone–Weierstrass theorem [118] is a generalization of the Weierstrass theorem, and is usually used for verifying a model's approximation capability to dynamic systems.

Theorem 2.5 (Stone–Weierstrass) *Let \mathcal{F} be a set of real continuous functions on a compact domain \mathcal{U} of n dimensions. Let \mathcal{F} satisfy the following criteria*

1. Algebraic closure: \mathcal{F} is closed under addition, multiplication, and scalar multiplication. That is, for any two $f_1, f_2 \in \mathcal{F}$, we have $f_1 f_2 \in \mathcal{F}$ and $a_1 f_1 + a_2 f_2 \in \mathcal{F}$, where a_1 and a_2 are any real numbers.
2. Separability on \mathcal{U} : for any two different points $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{U}$, $\mathbf{x}_1 \neq \mathbf{x}_2$, there exists $f \in \mathcal{F}$ such that $f(\mathbf{x}_1) \neq f(\mathbf{x}_2)$;
3. Not constantly zero on \mathcal{U} : for each $\mathbf{x} \in \mathcal{U}$, there exists $f \in \mathcal{F}$ such that $f(\mathbf{x}) \neq 0$.

Then \mathcal{F} is a dense subset of $\mathcal{C}(\mathcal{U})$, the set of all continuous real-valued functions on \mathcal{U} . In other words, for any $\varepsilon > 0$ and any function $g \in \mathcal{C}(\mathcal{U})$, there exists $f \in \mathcal{F}$ such that $|g(\mathbf{x}) - f(\mathbf{x})| < \varepsilon$ for any $\mathbf{x} \in \mathcal{U}$.

To date, numerous attempts have been made in searching for suitable forms of activation functions and proving the corresponding network's universal approximation

capabilities. Universal approximation to a given nonlinear functional under certain conditions can be realized by using the classical Volterra series or the Wiener series.

2.10.4 Winner-Takes-All

The winner-takes-all (WTA) competition is widely observed in both inanimate and biological media and society. Theoretical analysis [88] shows that WTA is a powerful computational module in comparison with threshold gates and sigmoidal gates (i.e., McCulloch–Pitts neurons). An optimal quadratic lower bound is given in [88] for computing WTA in any feedforward circuit consisting of threshold gates. Arbitrary continuous functions can be approximated by circuits employing a single soft WTA gate as their only nonlinear operation [88].

Theorem 2.6 (Maass, 1 [88]) *Assume that WTA with $n \geq 3$ inputs is computed by some arbitrary feedforward circuit C consisting of threshold gates with arbitrary weights. Then C consists of at least $\binom{n}{2} + n$ threshold gates.*

Theorem 2.7 (Maass, 2 [88]) *Any two-layer feedforward circuit C (with m analog or binary input variables and one binary output variable) consisting of threshold gates can be simulated by a circuit consisting of a single k -winner-take-all gate (k -WTA) applied to n weighted sums of the input variables with positive weights, except for some set $S \subseteq R^m$ of inputs that has measure 0.*

Any boolean function $f : \{0, 1\}^m \rightarrow \{0, 1\}$ can be computed by a single k -WTA gate applied to weighted sums of the input bits. If C has polynomial size and integer weights and the size is bounded by a polynomial in m , then n can be bounded by a polynomial in m , and all weights in the simulating circuit are natural numbers and the circuit size is bounded by a polynomial in m .

For real valued input (x_1, \dots, x_n) , a soft-WTA has its output (r_1, \dots, r_n) , analog numbers r_i reflecting the relative position of x_i within the ordering of x_i . Soft-WTA is plausible as computational function of cortical circuits with lateral inhibition. Single gates from a fairly large class of soft-WTA gates can serve as the only nonlinearity in universal approximators for arbitrary continuous functions.

Theorem 2.8 (Maass, 3 [88]) *Assume that $h : D \rightarrow [0, 1]$ is an arbitrary continuous function with a bounded and closed domain $D \subseteq R^m$. Then for any $\epsilon > 0$ and for any function g satisfying above conditions there exist natural numbers k, n , biases $\alpha_0^j \in R$, and coefficients $\alpha_i^j \geq 0$ for $i = 1, \dots, m, j = 1, \dots, n$, so that the circuit consisting of the soft-WTA gate $\text{soft-WTA}_{n,k}^g$ applied to the n sums $\sum_{i=1}^m \alpha_i^j z_i + \alpha_0^j$ for $j = 1, \dots, n$ computes a function $f : D \rightarrow [0, 1]$ so that $|f(z) - h(z)| < \epsilon$ for all $z \in D$. Thus, circuits consisting of a single soft-WTA gate applied to positive weighted sums of the input variables are universal approximators for continuous functions.*

2.11 Compressed Sensing and Sparse Approximation

A rationale behind sparse coding is the sparse connectivity between neurons in human brain. In the sparse coding model for the primary visual cortex, a small subset of learned dictionary elements will encode most natural images, and only a small subset of the cortical neurons need to be active for representing the high-dimensional visual inputs [99]. In a sparse representation a small number of coefficients contain a large portion of the energy. Sparse representations of signals are of fundamental importance in fields such as blind source separation, compression, sampling, and signal analysis.

Compressed sensing, or compressed sampling, integrates the signal acquisition and compression steps into a single process. It is an alternative to Shannon/Nyquist sampling for the acquisition of sparse or compressible signals that can be well approximated by just $K \ll N$ elements from an N -dimensional basis [28, 48]. Compressed sensing allows perfect recovery of sparse signals (or signals sparse in some basis) using only a small number of random measurements. In practice, signals tend to be compressible, rather than sparse. Compressible signals are well approximated by sparse signals.

Modeling a target signal as a sparse linear combination of atoms (elementary signals) drawn from a dictionary (a fixed collection), known as sparse coding, has become a popular paradigm in many fields, including signal processing, statistics, and machine learning. Many signals like audio, images, and video can be efficiently represented by sparse coding. Sparse coding is also a type of matrix factorization technique. The goal of sparse coding is to learn an over-complete basis set that represents each data point as a sparse combination of the basis vectors.

2.11.1 Compressed Sensing

Compressed sensing relies on two fundamental properties: Compressibility of the data and acquiring incoherent measurements. A signal \mathbf{x} is said to be *compressible* if there exists a dictionary Φ such that $\alpha = \Phi^T \mathbf{x}$ are sparsely distributed. In compressed sensing, a signal $\mathbf{x} \in \mathcal{C}^N$ is acquired by collecting data of linear measurements

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{n}, \quad (2.46)$$

where the random matrix \mathbf{A} is an $M \times N$ “sampling” or measurement matrix, with the number of measurements M in $\mathbf{y} \in \mathcal{C}^M$ smaller than the number of samples N in \mathbf{x} : $M < N$, and \mathbf{n} is a noise term. It focuses on underdetermined problems where the forward operator $\mathbf{A} \in \mathcal{C}^{M \times N}$ has unit-norm columns and forms an incomplete basis with $M \ll N$.

In compressed sensing, random distributions for generating \mathbf{A} have to satisfy the so-called restricted isometry property (RIP) in order to preserve the information in sparse and compressible signals, and ensure a stable recovery of both sparse and

compressible signals \mathbf{x} [28]. A large class of random matrices have the RIP with high probability.

Definition 2.4 (*K-restricted isometry property (K-RIP)*, [28]) The $M \times N$ matrix \mathbf{A} is said to satisfy the K -restricted isometry property (K -RIP) if there exists a constant $\delta \in (0, 1)$ such that

$$(1 - \delta)\|\mathbf{x}\|_2^2 \leq \|\mathbf{A}\mathbf{x}\|_2^2 \leq (1 + \delta)\|\mathbf{x}\|_2^2 \quad (2.47)$$

holds for any K -sparse vector \mathbf{x} of length N . A vector \mathbf{x} is said to be K -sparse when $\|\mathbf{x}\|_0 \leq K$, where $\|\cdot\|_0 : R^N \rightarrow R$ is L_0 -norm, which returns the number of nonzero elements in its argument, i.e., when \mathbf{x} has at most K nonzero entries. The minimum of all constants $\delta \in (0, 1)$ that satisfy (2.47) is referred to as the restricted isometry constant δ_K .

In other words, K -RIP ensures that all submatrices of \mathbf{A} of size $M \times K$ are close to an isometry, and therefore distance (and information) preserving. The goal is to push M as close as possible to K in order to perform as much signal compression during acquisition as possible. RIP measures the orthogonality of column vectors of a dictionary.

Compressed sensing can achieve stable recovery of compressible, noisy signals through the solution of the computationally tractable L_1 regularized inverse problem

$$\min_{\mathbf{x}} \|\mathbf{x}\|_1 \quad \text{subject to} \quad \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2 \leq \epsilon^2. \quad (2.48)$$

LP is the reconstruction method that achieves the best sparsity-undersampling tradeoff, but having a high computational cost for large-scale applications. LASSO [138] and approximate message-passing [50] are well-known low-complexity reconstruction procedures.

The popular least absolute selection and shrinkage operator (LASSO) minimizes a weighted sum of the residual norm and a regularization term $\|\mathbf{x}\|_1$. LASSO has the ability to reconstruct sparse solutions when sampling occurs far below the Nyquist rate, and also to recover the sparsity pattern exactly with probability one, asymptotically as the number of observations increases. The approximate message-passing algorithm [50] is an iterative-thresholding algorithm, leading to the sparsity-undersampling tradeoff equivalent to that of the corresponding LP procedure while running dramatically faster.

Standard compressed sensing dictates that robust signal recovery is possible from $O(K \log(N/K))$ measurements. A model-based compressed sensing theory [9] provides concrete guidelines on how to create model-based recovery algorithms with provable performance guarantees. Wavelet trees and block sparsity are integrated into two compressed sensing recovery algorithms (Matlab toolbox, <http://dsp.rice.edu/software>) and they are proved to offer robust recovery from just $O(K)$ measurements [9].

2.11.2 Sparse Approximation

With a formulation similar to that of compressed sensing, sparse approximation has a different objective. Assume that a target signal $\mathbf{y} \in R^M$ can be represented exactly (or at least approximated with sufficient accuracy) by a linear combination of exemplars in the overcomplete dictionary $\mathbf{A} = (\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_N)$:

$$\mathbf{y} = \mathbf{A}\mathbf{x}, \quad (2.49)$$

where \mathbf{A} is a real $M \times N$ matrix with $N \gg M$ whose columns have unit Euclidean norm: $\|\mathbf{a}_j\|_2 = 1$, $j = 1, 2, \dots, N$, and $\mathbf{x} \in R^N$. In fact, since the dictionary is overcomplete, any vector can be represented as a linear combination of vectors from the dictionary.

Sparsity has emerged as a fundamental type of regularization. Sparse approximation [36] seeks an approximate solution to (2.49) while requiring that the number K of nonzero entries of \mathbf{x} is only a few relative to its dimension N . Compressive sensing is a specific type of sparse approximation problem.

Although the system of linear equations in (2.49) has no unique solution, if \mathbf{x} is sufficiently sparse, \mathbf{x} can be uniquely determined by solving [48]

$$\mathbf{x} = \arg \min_{\tilde{\mathbf{x}} \in R^N} \|\tilde{\mathbf{x}}\|_0 \quad \text{subject to} \quad \mathbf{y} = \mathbf{A}\tilde{\mathbf{x}}. \quad (2.50)$$

The combinatorial problem (2.50) is NP-hard [96].

With weak conditions on \mathbf{A} , the solution of the L_0 -norm minimization given by (2.50) is equal to the solution of an L_1 -norm minimization [49]

$$\mathbf{x} = \arg \min_{\tilde{\mathbf{x}} \in R^N} \|\tilde{\mathbf{x}}\|_1 \quad \text{subject to} \quad \mathbf{y} = \mathbf{A}\tilde{\mathbf{x}}. \quad (2.51)$$

This convex minimization problem is the same as that given by (2.48). This indicates that the problems of recovering sparse signals from compressed measurements and constructing sparse approximation are the same in nature.

L_1 regularization gives rise to convex optimization and are thus widely used for generating results with sparse loadings, whereas L_0 regularization does not and furthermore yields NP-hard problems. However, sparsity is better achieved with L_0 penalties based on prediction and false discovery rate arguments [85]. The relation between L_1 and L_0 has been studied in the compressed sensing literature [48]. Under the RIP condition the L_1 and L_0 solutions are equal [49]. However, L_1 regularization may cause biased estimation for large coefficients since it over-penalizes true large coefficients [157].

As an extension of sparse approximation, the recovery of a data matrix from a sampling of its entries is considered in [29]. It is proved that a matrix $\mathbf{X} \in R^{m \times n}$ of rank r , $r \leq \min(m, n)$, can be perfectly recovered from a number k of entries selected uniformly at random from the matrix with very high probability if k obeys a certain condition [29]. The matrix completion problem is formulated as finding

the matrix with minimum nuclear norm that fits the data. This can be solved using iterative singular-value thresholding [27].

2.11.3 LASSO and Greedy Pursuit

The convex minimization problem given by (2.51) or (2.48) can be cast as an LS problem with L_1 penalty, also referred to as LASSO [138]

$$\mathbf{x} = \arg \min_{\tilde{\mathbf{x}} \in \mathbb{R}^N} \{\|\mathbf{A}\tilde{\mathbf{x}} - \mathbf{y}\|_2^2 + \lambda \|\tilde{\mathbf{x}}\|_1\} \quad (2.52)$$

with regularization parameter λ . Public domain software packages exist to solve problem (2.52) efficiently.

LASSO is probably the most popular supervised-learning technique that has been proposed to recover sparse signals from high-dimensional measurements. LASSO shrinks certain regression coefficients to zero, giving interpretable models that are sparse. It minimizes the sum of squared errors, given a fixed bound on the sum of absolute value of the regression coefficients. LASSO and many L_1 -regularized regression methods typically need to set a regularization parameter.

LASSO solves a robust optimization problem. The sparsity and consistency of LASSO are shown based on its robustness interpretation [152]. Furthermore, the robust optimization formulation is shown to be related to kernel density estimation. A no-free-lunch theorem is proved in [152] which states that sparsity and algorithmic stability contradict each other, and hence LASSO is not stable. An asymptotic analysis shows that the asymptotic variances of some of the robust versions of LASSO estimators are stabilized in the presence of large variance noise, compared with the unbounded asymptotic variance of the ordinary LASSO estimator [37].

The LS ridge regression model estimates linear regression coefficients, with the L_2 ridge regularization on coefficients. To better identify important features in the data, LASSO uses the L_1 penalty instead of using the L_2 ridge regularization. LASSO, or L_1 regularized LS, has been explored extensively for its remarkable sparsity properties. For sparse, high-dimensional regression problems, marginal regression, where each dependent variable is regressed separately on each covariate, computes the estimates roughly two orders of magnitude faster than the LASSO solutions [60].

A greedy algorithm is usually used for solving the convex minimization problem given by (2.51) or (2.48). Basis pursuit [36] is a greedy sparse approximation technique for decomposing a signal into a superposition of dictionary elements (basis functions), which has the smallest L_1 norm of coefficients among all such decompositions. It is implemented as pdco and SolveBP in the SparseLab toolbox (<http://sparselab.stanford.edu>). A similar method [53] is implemented as SolveLasso in the SparseLab toolbox.

Orthogonal matching pursuit (OMP) [101] is the simplest effective greedy algorithm for sparse approximation. At each iteration, a column of \mathbf{A} that is maximally correlated with the residual is chosen, the index of this column is added to the list,

and then the vestige of columns in the list is eliminated from the measurements, generating a new residual for the next iteration. OMP adds one new element of the dictionary and makes one orthogonal projection at each iteration. Generalized OMP [148] generalizes OMP by identifying multiple indices per iteration. Similarly, the orthogonal super greedy algorithm [86] adds multiple new elements of the dictionary and makes one orthogonal projection at each iteration. The performance of orthogonal multimatching pursuit, a counterpart of the orthogonal super greedy algorithm in the compressed sensing setting, is analyzed in [86] under RIP conditions.

In order to solve the sparse approximation problem, a single sufficient condition is developed in [140] under which both OMP and basis pursuit can recover an exactly sparse signal. For every input signal, OMP can calculate a sparse approximant whose error is only a small factor worse than the optimal error which can be attained with the same number of terms [140]. OMP can reliably recover a signal with K nonzero entries in dimension N given $O(K \ln N)$ random linear measurements of that signal [141].

A sparse LMS algorithm [38] takes advantage of the sparsity of the underlying signal for system identification. This is done by incorporating two sparsity constraints into the quadratic cost function of the LMS algorithm. A recursive L_1 -regularized LS algorithm is developed in [7] for the estimation of a sparse tap-weight vector in the adaptive filtering setting. It exploits noisy observations of the tap-weight vector output stream and produces its estimate using an EM-type algorithm. Recursive L_1 -regularized LS converges to a near-optimal estimate in a stationary environment. It has significant improvement over the RLS algorithm in terms of MSE, but with lower computational requirements.

2.12 Bibliographical Notes

Some good books on machine and statistical learning are Duda et al. [51], Bishop [22], Bishop [24], Ripley [111], Cherkassky and Mulier [40], Vapnik [147], and Hastie et al. [68].

Functional data analysis

Functional data analysis is an extension of the traditional data analysis to functional data [107]. Functional data analysis characterizes a series of data points as a single piece of data. Examples of functional data are spectra, temporal series, and spatiotemporal images. Functional data are usually represented by regular or irregular sampling as lists of input-output pairs. Functional data analysis is closely related with the multivariate statistics and regularization. Many statistical methods, such as PCA, multivariate linear modeling and CCA, can be applied within this framework. Conventional neural network models have been extended to functional data inputs, such as the RBF network [115], the MLP [116], SVMs [117], and k -NN method [21].

Parametric, semiparametric and nonparametric classification

Pattern classification techniques with numerical inputs can be generally classified into parametric, semiparametric, and nonparametric groups. The parametric and semiparametric classifiers need certain amount of a priori information about the structure of the data in the training set. Parametric techniques assume that the form of the pdf is known in advance except for a vector of parameters, which has to be estimated from the sample of realizations. In this case, smaller sample size can yield good performance if the form of the pdf is properly selected. When some insights about the form of the pdf are available, parametric techniques offer the most valid and efficient approach to density estimation. Semiparametric techniques consider models having a number of parameters not growing with the sample size, though greater than that involved in parametric techniques.

Nonparametric techniques aim to retrieve the behavior of the pdf without imposing any a priori assumption on it; therefore, they require a sample size significantly higher than the dimension of the domain of the random variable. Density estimation methods using neural networks or SVMs fall into the category of nonparametric techniques. The Parzen's windows approach [100] is a nonparametric method for estimating the pdf of a finite set of patterns; it has a very high computational cost due to the very large number of kernels required for its representation. A decision tree such as C5.0 (<http://www.rulequest.com/see5-info.html>) is an efficient nonparametric method. A decision tree is a hierarchical data structure implementing the divide-and-conquer strategy. It is a supervised learning method, and can be used for both classification and regression.

Complexity of circuits

Complexity theory of circuits strongly suggests that deep architectures can be much more efficient than their shallow counterparts, in terms of computational elements and parameters required to represent some functions. Theoretical results on circuit complexity theory have shown that shallow digital circuits can be exponentially less efficient than deeper ones [66]. An equivalent result has been proved for architectures whose computational elements are linear threshold units [67]. Any Boolean function can be represented by a two-layer circuit of logic gates. However, most Boolean functions require an exponential number of logic gates (with respect to the input size) to be represented by a two-layer circuit. For example, the parity function, which can be efficiently represented by a circuit of depth $O(\log n)$ (for n input bits) needs $O(2^n)$ gates to be represented by a depth two circuit [155].

Categorical data

Categorical data can generally be classified into ordinal data and nominal data. Ordinal and nominal data both have a set of possible states, and the value of a variable will be in one of those possible states. The difference between them is that the states in ordinal data are ordered but are unordered in nominal data. A nominal variable can only have two matching results, either match or does not match. For instance, hair color is a nominal variable that may have four states: *black*, *blond*, *red*, and *brown*. Service quality assessment is an ordinal variable that may have five states: *very good*, *good*, *medium*, *poor*, *very poor*.

Ordinal regression is generally defined as the task where some input sample vectors are ranked on an ordinal scale. Ordinal regression is commonly formulated as a multiclass problem with ordinal constraints. The aim is to predict variables of ordinal scale. In contrast to traditional metric regression problems, these ranks are of finite types and the metric distances between the ranks are not defined. The naive idea is to transform the ordinal scales into numerical values and then solve the problem as a standard regression problem.

Occam's razor

A widely accepted interpretation of Occam's razor is: "Given two classifiers with the same training error, the simpler classifier is more likely to generalize better." Domingos [47] rejects this interpretation and proposes that model complexity is only a confounding factor usually correlated with the number of models from which the learner selects. It is thus hypothesized that the risk of overfitting (poor generalization) follows only from the number of model tests rather than the complexity of the selected model. The confusion between the two factors arises from the fact that a learning algorithm usually conducts a greater amount of testing to fit a more complex model. Experiments results on real-life datasets confirm Domingos' hypothesis [156]. In particular, the experiments test the following assertions. (i) Models selected from a larger set of tested candidate models overfit more than those selected from a smaller set (assuming constant model complexity). (ii) More complex models overfit more than simpler models (assuming a constant number of candidate models tested). According to Domingos' hypothesis, the first assertion should be true and the second should be false.

Learning from imbalanced data

Learning from imbalanced data is a challenging problem. It is the problem of learning a classification rule from data that are skewed in favor of one class. Many real-world datasets are imbalanced and the majority class has much more training patterns than the minority class. The resultant hyperplane will be shifted towards the majority class. However, the minority class is often the most interesting one for the task.

For the imbalanced datasets, a classifier may fail. The remedies can be divided into two categories. The first category processes the data before feeding them into the classifier, such as the oversampling and undersampling techniques, combining oversampling with undersampling, and synthetic minority oversampling technique (SMOTE) [34]. The oversampling technique duplicates the positive data by interpolation while undersampling technique removes the redundant negative data to reduce the imbalanced ratio. They are classifier-independent approaches. The second category belongs to the algorithm-based approach such as different error cost algorithms [84], and class-boundary-alignment algorithm [151]. The different cost algorithms suggest that by assigning heavier penalty to the smaller class, the skew of the optimal separating hyperplane can be corrected.

Problems

2.1 A distance measure, or metric, between two points, must satisfy three conditions:

- Positivity: $d(\mathbf{x}, \mathbf{y}) \geq 0$ and $d(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$;
- Symmetry: $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$;
- Triangle inequality: $d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z}) \geq d(\mathbf{x}, \mathbf{z})$.

- Show that the Euclidean distance, the city block distance, and the maximum value distance are metrics.
- Show that the squared Euclidean distance is not a metric.
- How about the Hamming distance?

2.2 Is it possible to use a single neuron to approximate the function $f(x) = x^2$?

2.3 Are the following set of points linearly separable?

Class 1: (0,0,0,0), (1,0,0,1), (0,1,0,1); class 2: (1,1,1,1), (1,1,0,0), (1,0,1,0).

2.4 For a K -class problem, the target \mathbf{t}_k , $k = 1, \dots, K$, is a vector of all zeros but for a one in the k th position, show that classifying a pattern to the largest element of $\hat{\mathbf{y}}$, if $\hat{\mathbf{y}}$ is normalized, is equivalent to choosing the closest target, $\min_k \|\mathbf{t}_k - \hat{\mathbf{y}}\|$.

2.5 Given a set of N samples $(\mathbf{x}_k, \mathbf{t}_k)$, $k = 1, \dots, N$, derive the optimal least squares parameter $\hat{\mathbf{w}}$ for the total training loss

$$E = \sum_{n=1}^N (t_n - \mathbf{w}^T \mathbf{x}_n)^2.$$

Compare the expression with that derived from the average loss.

2.6 Assume that we have a class of functions $\{f(\mathbf{x}, \boldsymbol{\alpha})\}$ indexed by a parameter vector $\boldsymbol{\alpha}$, with $\mathbf{x} \in \mathbb{R}^p$, f being an indicator function, taking value 0 or 1. If $\boldsymbol{\alpha} = (\alpha_0, \alpha_1)$ and f is the linear indicator function $I(\alpha_0 + \alpha_1 x > 0)$, then the complexity of the class f is the number of parameters $p + 1$.

The indicator function $I(\sin(\alpha x) > 0)$ can shatter (separate) an arbitrarily large number of points by choosing an appropriately high frequency α . Show that the set of functions $\{I(\sin(\alpha x) > 0)\}$ can shatter the following points on the line: $x_1 = 2^{-1}, \dots, x_M = 2^{-M}, \forall M$. Hence the VC dimension of the class $\{I(\sin(\alpha x) > 0)\}$ is infinite.

2.7 For an input vector \mathbf{x} with p components and a target y , the projection pursuit regression model has the form

$$f(\mathbf{x}) = \sum_{m=1}^M g_m(\mathbf{w}_m^T \mathbf{x}),$$

where \mathbf{w}_m , $m = 1, 2, \dots, M$, are unit p -vectors of unknown parameters. The functions g_m are estimated along with the direction \mathbf{w}_m using some flexible smoothing method. Neural networks are just nonlinear statistical models. Show how neural networks resemble the projection pursuit regression model.

2.8 The XOR operation is a linearly inseparable problem. Show that a quadratic threshold gate

$$y = \begin{cases} 1, & \text{if } g(x) = \sum_{i=1}^n w_i x_i + \sum_{i=1}^n \sum_{j=1}^n w_{ij} x_i x_j \geq T \\ 0, & \text{otherwise} \end{cases}$$

can be used to separate them. Give an example for $g(x)$, and plot the separating surface.

2.9 Plot four points that are in general position. Show how they are separated by separating lines.

References

1. Akaike, H. (1969). Fitting autoregressive models for prediction. *Annals of the Institute of Statistical Mathematics*, 21, 425–439.
2. Akaike, H. (1970). Statistical prediction information. *Annals of the Institute of Statistical Mathematics*, 22, 203–217.
3. Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19, 716–723.
4. Amari, S., Murata, N., Muller, K. R., Finke, M., & Yang, H. (1996). Statistical theory of overtraining-Is cross-validation asymptotically effective? In D. S. Touretzky, M. C. Mozer & M. E. Hasselmo (Eds.), *Advances in neural information processing systems* (Vol. 8, pp. 176–182). Cambridge, MA: MIT Press.
5. Anthony, M., & Biggs, N. (1992). *Computational learning theory*. Cambridge, UK: Cambridge University Press.
6. Auer, P., Herbster, M., & Warmuth, M. K. (1996). Exponentially many local minima for single neurons. In D. S. Touretzky, M. C. Mozer & M. E. Hasselmo (Eds.), *Advances in neural information processing systems* (Vol. 8, pp. 316–322). Cambridge, MA: MIT Press.
7. Babadi, B., Kalouptsidis, N., & Tarokh, V. (2010). SPARLS: The sparse RLS algorithm. *IEEE Transactions on Signal Processing*, 58(8), 4013–4025.
8. Back, A. D., & Trappenberg, T. P. (2001). Selecting inputs for modeling using normalized higher order statistics and independent component analysis. *IEEE Transactions on Neural Networks*, 12(3), 612–617.
9. Baraniuk, R. G., Cevher, V., Duarte, M. F., & Hegde, C. (2010). Model-based compressive sensing. *IEEE Transactions on Information Theory*, 56(4), 1982–2001.
10. Barron, A. R. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3), 930–945.
11. Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13, 834–846.
12. Bartlett, P. L. (1998). The sample complexity of pattern classification with neural networks: The size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory*, 44(2), 525–536.
13. Bartlett, P. L. (1993). Lower bounds on the Vapnik-Chervonenkis dimension of multi-layer threshold networks. In *Proceedings of the 6th Annual ACM Conference on Computational Learning Theory* (pp. 144–150). New York: ACM Press.
14. Bartlett, P. L., & Maass, W. (2003). Vapnik-Chervonenkis dimension of neural nets. In M. A. Arbib (Ed.), *The handbook of brain theory and neural networks* (2nd ed., pp. 1188–1192). Cambridge, MA: MIT Press.

15. Battiti, R. (1994). Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on Neural Networks*, 5(4), 537–550.
16. Baum, E. B., & Wilczek, F. (1988). Supervised learning of probability distributions by neural networks. In D. Z. Anderson (Ed.), *Neural information processing systems* (pp. 52–61). New York: American Institute of Physics.
17. Baum, E. B., & Haussler, D. (1989). What size net gives valid generalization? *Neural Computation*, 1, 151–160.
18. Belkin, M., Niyogi, P., & Sindhwani, V. (2006). Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7, 2399–2434.
19. Bengio, Y., & Grandvalet, Y. (2004). No unbiased estimator of the variance of K -fold cross-validation. *Journal of Machine Learning Research*, 5, 1089–1105.
20. Bernier, J. L., Ortega, J., Ros, E., Rojas, I., & Prieto, A. (2000). A quantitative study of fault tolerance, noise immunity, and generalization ability of MLPs. *Neural Computation*, 12, 2941–2964.
21. Biau, G., Bunea, F., & Wegkamp, M. (2005). Functional classification in Hilbert spaces. *IEEE Transactions on Information Theory*, 51, 2163–2172.
22. Bishop, C. M. (1995). *Neural networks for pattern recognition*. New York: Oxford Press.
23. Bishop, C. M. (1995). Training with noise is equivalent to Tikhonov regularization. *Neural Computation*, 7(1), 108–116.
24. Bishop, C. (2006). *Pattern recognition and machine learning*. New York: Springer.
25. Blum, A. L., & Rivest, R. L. (1992). Training a 3-node neural network is NP-complete. *Neural Networks*, 5(1), 117–127.
26. Bousquet, O., & Elisseeff, A. (2002). Stability and Generalization. *Journal of Machine Learning Research*, 2, 499–526.
27. Cai, J.-F., Candes, E. J., & Shen, Z. (2010). A singular value thresholding algorithm for matrix completion. *SIAM Journal of Optimization*, 20(4), 1956–1982.
28. Candes, E. J. (2006). Compressive sampling. In *Proceedings of International Congress on Mathematicians, Madrid, Spain* (Vol. 3, pp. 1433–1452).
29. Candes, E. J., & Recht, B. (2009). Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9, 717–772.
30. Caruana, R. (1997). Multitask learning. *Machine Learning*, 28, 41–75.
31. Cataltepe, Z., Abu-Mostafa, Y. S., & Magdon-Ismael, M. (1999). No free lunch for early stopping. *Neural Computation*, 11, 995–1009.
32. Cawley, G. C., & Talbot, N. L. C. (2007). Preventing over-fitting during model selection via Bayesian regularisation of the hyper-parameters. *Journal of Machine Learning Research*, 8, 841–861.
33. Cawley, G. C., & Talbot, N. L. C. (2010). On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11, 2079–2107.
34. Chawla, N., Bowyer, K., & Kegelmeyer, W. (2002). SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16, 321–357.
35. Chen, D. S., & Jain, R. C. (1994). A robust backpropagation learning algorithm for function approximation. *IEEE Transactions on Neural Networks*, 5(3), 467–479.
36. Chen, S. S., Donoho, D. L., & Saunders, M. A. (1999). Atomic decomposition by basis pursuit. *SIAM Journal of Scientific Computing*, 20(1), 33–61.
37. Chen, X., Wang, Z. J., & McKeown, M. J. (2010). Asymptotic analysis of robust LASSOs in the presence of noise with large variance. *IEEE Transactions on Information Theory*, 56(10), 5131–5149.
38. Chen, Y., Gu, Y., & Hero, A. O., III. (2009). Sparse LMS for system identification. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)* (pp. 3125–3128). Taipei, Taiwan.
39. Cherkassky, V., & Ma, Y. (2003). Comparison of model selection for regression. *Neural Computation*, 15, 1691–1714.

40. Cherkassky, V., & Mulier, F. (2007). *Learning from data* (2nd ed.). New York: Wiley.
41. Cherkassky, V., & Ma, Y. (2009). Another look at statistical learning theory and regularization. *Neural Networks*, 22, 958–969.
42. Chiu, C., Mehrotra, K., Mohan, C. K., & Ranka, S. (1994). Modifying training algorithms for improved fault tolerance. In *Proceedings of IEEE International Conference on Neural Networks* (Vol. 4, pp. 333–338).
43. Cichocki, A., & Unbehauen, R. (1992). *Neural networks for optimization and signal processing*. New York: Wiley.
44. Cover, T. M. (1965). Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, 14, 326–334.
45. Denker, J. S., Schwartz, D., Wittner, B., Solla, S. A., Howard, R., Jackel, L., et al. (1987). Large automatic learning, rule extraction, and generalization. *Complex Systems*, 1, 877–922.
46. Dietterich, T. G., Lathrop, R. H., & Lozano-Perez, T. (1997). Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89, 31–71.
47. Domingos, P. (1999). The role of Occam's razor in knowledge discovery. *Data Mining and Knowledge Discovery*, 3, 409–425.
48. Donoho, D. L. (2006). Compressed sensing. *IEEE Transactions on Information Theory*, 52(4), 1289–1306.
49. Donoho, D. L. (2006). For most large underdetermined systems of linear equations the minimal l_1 -norm solution is also the sparsest solution. *Communications on Pure and Applied Mathematics*, 59, 797–829.
50. Donoho, D. L., Maleki, A., & Montanari, A. (2009). Message-passing algorithms for compressed sensing. *Proceedings of the National Academy of Sciences of the USA*, 106(45), 18914–18919.
51. Duda, R., Hart, P., & Stork, D. (2000). *Pattern classification* (2nd ed.). New York: Wiley.
52. Edwards, P. J., & Murray, A. F. (1998). Towards optimally distributed computation. *Neural Computation*, 10, 997–1015.
53. Efron, B., Hastie, T., Johnstone, I., & Tibshirani, R. (2004). Least angle regression. *Annals of Statistics*, 32(2), 407–499.
54. Estevez, P. A., Tesmer, M., Perez, C. A., & Zurada, J. M. (2009). Normalized mutual information feature selection. *IEEE Transactions on Neural Networks*, 20(2), 189–201.
55. Fedorov, V. V. (1972). *Theory of optimal experiments*. San Diego, CA: Academic Press.
56. Freund, Y., Seung, H. S., Shamir, E., & Tishby, N. (1997). Selective sampling using the query by committee algorithm. *Machine Learning*, 28, 133–168.
57. Friedman, J. H., & Tukey, J. W. (1974). A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, 23(9), 881–889.
58. Friedrichs, F., & Schmitt, M. (2005). On the power of Boolean computations in generalized RBF neural networks. *Neurocomputing*, 63, 483–498.
59. Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1), 1–58.
60. Genovese, C. R., Jin, J., Wasserman, L., & Yao, Z. (2012). A comparison of the lasso and marginal regression. *Journal of Machine Learning Research*, 13, 2107–2143.
61. Ghodsi, A., & Schuurmans, D. (2003). Automatic basis selection techniques for RBF networks. *Neural Networks*, 16, 809–816.
62. Gish, H. (1990). A probabilistic approach to the understanding and training of neural network classifiers. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)* (pp. 1361–1364).
63. Goutte, C. (1997). Note on free lunches and cross-validation. *Neural Computation*, 9(6), 1245–1249.
64. Hanson, S. J., & Burr, D. J. (1988). Minkowski back-propagation: Learning in connectionist models with non-Euclidean error signals. In D. Z. Anderson (Ed.), *Neural Information processing systems* (pp. 348–357). New York: American Institute of Physics.

65. Hassoun, M. H. (1995). *Fundamentals of artificial neural networks*. Cambridge, MA: MIT Press.
66. Hastad, J. T. (1987). *Computational limitations for small depth circuits*. Cambridge, MA: MIT Press.
67. Hastad, J., & Goldmann, M. (1991). On the power of small-depth threshold circuits. *Computational Complexity*, 1, 113–129.
68. Hastie, T., Tibshirani, R., & Friedman, J. (2005). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Berlin: Springer.
69. Haussler, D. (1990). Probably approximately correct learning. In *Proceedings of 8th National Conference on Artificial Intelligence* (Vol. 2, pp. 1101–1108). Boston, MA.
70. Haykin, S. (1999). *Neural networks: A comprehensive foundation* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.
71. Hecht-Nielsen, R. (1987). Kolmogorov's mapping neural network existence theorem. In *Proceedings of the 1st IEEE International Conference on Neural Networks* (Vol. 3, pp. 11–14). San Diego, CA.
72. Hinton, G. E. (1989). Connectionist learning procedure. *Artificial Intelligence*, 40, 185–234.
73. Hinton, G. E., & van Camp, D. (1993). Keeping neural networks simple by minimizing the description length of the weights. In *Proceedings of the 6th Annual ACM Conference on Computational Learning Theory* (pp. 5–13). Santa Cruz, CA.
74. Ho, K. I.-J., Leung, C.-S., & Sum, J. (2010). Convergence and objective functions of some fault/noise-injection-based online learning algorithms for RBF networks. *IEEE Transactions on Neural Networks*, 21(6), 938–947.
75. Hoi, S. C. H., Jin, R., & Lyu, M. R. (2009). Batch mode active learning with applications to text categorization and image retrieval. *IEEE Transactions on Knowledge and Data Engineering*, 21(9), 1233–1248.
76. Holmstrom, L., & Koistinen, P. (1992). Using additive noise in back-propagation training. *IEEE Transactions on Neural Networks*, 3(1), 24–38.
77. Huber, P. J. (1981). *Robust statistics*. New York: Wiley.
78. Janssen, P., Stoica, P., Soderstrom, T., & Eykhoff, P. (1988). Model structure selection for multivariable systems by cross-validation. *International Journal of Control*, 47, 1737–1758.
79. Kohavi, R., & John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, 97, 273–324.
80. Koiran, P., & Sontag, E. D. (1996). Neural networks with quadratic VC dimension. In D. S. Touretzky, M. C. Mozer, & M. E. Hasselmo (Eds.), *Advances in neural information processing systems* (Vol. 8, pp. 197–203). Cambridge, MA: MIT Press.
81. Kolmogorov, A. N. (1957). On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk USSR*, 114(5), 953–956.
82. Krogh, A., & Hertz, J. A. (1992). A simple weight decay improves generalization. In *Proceedings of Neural Information and Processing Systems (NIPS) Conference* (pp. 950–957). San Mateo, CA: Morgan Kaufmann.
83. Leiva-Murillo, J. M., & Artes-Rodriguez, A. (2007). Maximization of mutual information for supervised linear feature extraction. *IEEE Transactions on Neural Networks*, 18(5), 1433–1441.
84. Lin, Y., Lee, Y., & Wahba, G. (2002). Support vector machines for classification in nonstandard situations. *Machine Learning*, 46, 191–202.
85. Lin, D., Pitler, E., Foster, D. P., & Ungar, L. H. (2008). In defense of l_0 . In *Proceedings of International Conference on Machine Learning: Workshop of Sparse Optimization and Variable Selection*. Helsinki, Finland.
86. Liu, E., & Temlyakov, V. N. (2012). The orthogonal super greedy algorithm and applications in compressed sensing. *IEEE Transactions on Information Theory*, 58(4), 2040–2047.
87. Liu, Y., Starzyk, J. A., & Zhu, Z. (2008). Optimized approximation algorithm in neural networks without overfitting. *IEEE Transactions on Neural Networks*, 19(6), 983–995.

88. Maass, W. (2000). On the computational power of winner-take-all. *Neural Computation*, 12, 2519–2535.
89. MacKay, D. (1992). Information-based objective functions for active data selection. *Neural Computation*, 4(4), 590–604.
90. Magdon-Ismael, M. (2000). No free lunch for noise prediction. *Neural Computation*, 12, 547–564.
91. Markatou, M., Tian, H., Biswas, S., & Hripcsak, G. (2005). Analysis of variance of cross-validation estimators of the generalization error. *Journal of Machine Learning Research*, 6, 1127–1168.
92. Matsuoka, K., & Yi, J. (1991). Backpropagation based on the logarithmic error function and elimination of local minima. In *Proceedings of the International Joint Conference on Neural Networks* (pp. 1117–1122). Seattle, WA.
93. Muller, B., Reinhardt, J., & Strickland, M. (1995). *Neural networks: An introduction* (2nd ed.). Berlin: Springer.
94. Murray, A. F., & Edwards, P. J. (1994). Synaptic weight noise euring MLP training: Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training. *IEEE Transactions on Neural Networks*, 5(5), 792–802.
95. Nadeau, C., & Bengio, Y. (2003). Inference for the generalization error. *Machine Learning*, 52, 239–281.
96. Natarajan, B. K. (1995). Sparse approximate solutions to linear systems. *SIAM Journal of Computing*, 24(2), 227–234.
97. Niyogi, P., & Girosi, F. (1999). Generalization bounds for function approximation from scattered noisy data. In *Advances in computational mathematics* (Vol. 10, pp. 51–80). Berlin: Springer.
98. Nowlan, S. J., & Hinton, G. E. (1992). Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4), 473–493.
99. Olshausen, B. A., & Field, D. J. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381, 607–609.
100. Parzen, E. (1962). On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(1), 1065–1076.
101. Pati, Y. C., Rezaifar, R., & Krishnaprasad, P. S. (1993). Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Proceedings of the 27th Annual Asilomar Conference on Signals, Systems, and Computers* (Vol. 1, pp. 40–44).
102. Phatak, D. S. (1999). Relationship between fault tolerance, generalization and the Vapnik-Cervonenkis (VC) dimension of feedforward ANNs. In *Proceedings of International Joint Conference on Neural Networks* (Vol. 1, pp. 705–709).
103. Picone, J. (1993). Signal modeling techniques in speech recognition. *Proceedings of the IEEE*, 81(9), 1215–1247.
104. Plutowski, M. E. P. (1996). *Survey: Cross-validation in theory and in practice*. Research Report, Princeton, NJ: Department of Computational Science Research, David Sarnoff Research Center.
105. Poggio, T., & Girosi, F. (1990). Networks for approximation and learning. *Proceedings of the IEEE*, 78(9), 1481–1497.
106. Prechelt, L. (1998). Automatic early stopping using cross validation: Quantifying the criteria. *Neural Networks*, 11, 761–767.
107. Ramsay, J., & Silverman, B. (1997). *Functional data analysis*. New York: Springer.
108. Reed, R., Marks, R. J., II, & Oh, S. (1995). Similarities of error regularization, sigmoid gain scaling, target smoothing, and training with jitter. *IEEE Transactions on Neural Networks*, 6(3), 529–538.
109. Rimer, M., & Martinez, T. (2006). Classification-based objective functions. *Machine Learning*, 63(2), 183–205.
110. Rimer, M., & Martinez, T. (2006). CB3: an adaptive error function for backpropagation training. *Neural Processing Letters*, 24, 81–92.

111. Ripley, B. D. (1996). *Pattern recognition and neural networks*. Cambridge, UK: Cambridge University Press.
112. Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14(5), 465–477.
113. Rissanen, J. (1999). Hypothesis selection and testing by the MDL principle. *Computer Journal*, 42(4), 260–269.
114. Rivals, I., & Personnaz, L. (1999). On cross-validation for model selection. *Neural Computation*, 11(4), 863–870.
115. Rossi, F., & Conan-Guez, B. (2005). Functional multi-layer perceptron: A non-linear tool for functional data analysis. *Neural Networks*, 18, 45–60.
116. Rossi, F., Delannay, N., Conan-Guez, B., & Verleysen, M. (2005). Representation of functional data in neural networks. *Neurocomputing*, 64, 183–210.
117. Rossi, F., & Villa, N. (2006). Support vector machine for functional data classification. *Neurocomputing*, 69, 730–742.
118. Royden, H. L. (1968). *Real analysis* (2nd ed.). New York: Macmillan.
119. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Foundation* (Vol. 1, pp. 318–362). Cambridge, MA: MIT Press.
120. Rumelhart, D. E., Durbin, R., Golden, R., & Chauvin, Y. (1995). Backpropagation: The basic theory. In Y. Chauvin & D. E. Rumelhart (Eds.), *Backpropagation: Theory, architecture, and applications* (pp. 1–34). Hillsdale, NJ: Lawrence Erlbaum.
121. Sabato, S., & Tishby, N. (2012). Multi-instance learning with any hypothesis class. *Journal of Machine Learning Research*, 13, 2999–3039.
122. Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5, 197–227.
123. Schmitt, M. (2005). On the capabilities of higher-order neurons: A radial basis function approach. *Neural Computation*, 17, 715–729.
124. Schultz, W. (1998). Predictive reward signal of dopamine neurons. *Journal of Neurophysiology*, 80(1), 1–27.
125. Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*, 6, 461–464.
126. Shao, X., Cherkassky, V., & Li, W. (2000). Measuring the VC-dimension using optimized experimental design. *Neural Computing*, 12, 1969–1986.
127. Shawe-Taylor, J. (1995). Sample sizes for sigmoidal neural networks. In *Proceedings of the 8th Annual Conference on Computational Learning Theory* (pp. 258–264). Santa Cruz, CA.
128. Siegelmann, H. T., & Sontag, E. D. (1995). On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1), 132–150.
129. Silva, L. M., de Sa, J. M., & Alexandre, L. A. (2008). Data classification with multilayer perceptrons using a generalized error function. *Neural Networks*, 21, 1302–1310.
130. Sima, J. (1996). Back-propagation is not efficient. *Neural Networks*, 9(6), 1017–1023.
131. Solla, S. A., Levin, E., & Fleisher, M. (1988). Accelerated learning in layered neural networks. *Complex Systems*, 2, 625–640.
132. Stoica, P., & Selen, Y. (2004). A review of information criterion rules. *IEEE Signal Processing Magazine*, 21(4), 36–47.
133. Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society Series B*, 36, 111–147.
134. Sugiyama, M., & Ogawa, H. (2000). Incremental active learning for optimal generalization. *Neural Computation*, 12, 2909–2940.
135. Sugiyama, M., & Nakajima, S. (2009). Pool-based active learning in approximate linear regression. *Machine Learning*, 75, 249–274.
136. Sum, J. P.-F., Leung, C.-S., & Ho, K. I.-J. (2012). On-line node fault injection training algorithm for MLP networks: Objective function and convergence analysis. *IEEE Transactions on Neural Networks and Learning Systems*, 23(2), 211–222.
137. Tabatabai, M. A., & Argyros, I. K. (1993). Robust estimation and testing for general nonlinear regression models. *Applied Mathematics and Computation*, 58, 85–101.

138. Tibshirani, R. (1996). Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society Series B*, 58(1), 267–288.
139. Tikhonov, A. N. (1963). On solving incorrectly posed problems and method of regularization. *Doklady Akademii Nauk USSR*, 151, 501–504.
140. Tropp, J. A. (2004). Greed is good: Algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, 50, 2231–2242.
141. Tropp, J. A., & Gilbert, A. C. (2007). Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory*, 53(12), 4655–4666.
142. Valiant, P. (1984). A theory of the learnable. *Communications of the ACM*, 27(11), 1134–1142.
143. Vapnik, V. N., & Chervonenkis, A. J. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & its Applications*, 16, 264–280.
144. Vapnik, V. N. (1982). *Estimation of dependences based on empirical data*. New York: Springer.
145. Vapnik, V., Levin, E., & Le Cun, Y. (1994). Measuring the VC-dimension of a learning machine. *Neural Computation*, 6, 851–876.
146. Vapnik, V. N. (1995). *The nature of statistical learning theory*. New York: Springer.
147. Vapnik, V. N. (1998). *Statistical learning theory*. New York: Wiley.
148. Wang, J., Kwon, S., & Shim, B. (2012). Generalized orthogonal matching pursuit. *IEEE Transactions on Signal Processing*, 60(12), 6202–6216.
149. Widrow, B., & Lehr, M. A. (1990). 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9), 1415–1442.
150. Wolpert, D. H., & Macready, W. G. (1995). *No free lunch theorems for search*, SFI-TR-95-02-010, Santa Fe Institute.
151. Wu, G., & Cheng, E. (2003). Class-boundary alignment for imbalanced dataset learning. In *Proceedings of ICML 2003 Workshop on Learning Imbalanced Data Sets II* (pp. 49–56). Washington, DC.
152. Xu, H., Caramanis, C., & Mannor, S. (2010). Robust regression and Lasso. *IEEE Transactions on Information Theory*, 56(7), 3561–3574.
153. Xu, H., Caramanis, C., & Mannor, S. (2012). Sparse algorithms are not stable: A no-free-lunch theorem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(1), 187–193.
154. Yang, L., Hanneke, S., & Carbonell, J. (2013). A theory of transfer learning with applications to active learning. *Machine Learning*, 90(2), 161–189.
155. Yao, A. (1985). Separating the polynomial-time hierarchy by oracles. In *Proceedings of 26th Annual IEEE Symposium on Foundations Computer Science* (pp. 1–10).
156. Zahalka, J., & Zelezny, F. (2011). An experimental test of Occam’s razor in classification. *Machine Learning*, 82, 475–481.
157. Zhang, C. H. (2010). Nearly unbiased variable selection under minimax concave penalty. *Annals of Statistics*, 38(2), 894–942.
158. Zhu, H. (1996). No free lunch for cross validation. *Neural Computation*, 8(7), 1421–1426.

<http://www.springer.com/978-1-4471-5570-6>

Neural Networks and Statistical Learning

Du, K.-L.; Swamy, M.N.S.

2014, XXVII, 824 p. 166 illus., 68 illus. in color.,

Hardcover

ISBN: 978-1-4471-5570-6