1

BACKGROUND AND PREVIEW

1.1 SUPERVISED, SEQUENTIAL, AND ACTIVE LEARNING

Learning is a process by which the free parameters and the topology of a neural network are adapted through a process of stimulation by the environment in which the network is embedded [Haykin, 2009].

Adjustments on the free parameters are well studied in the adaptive filtering and neural network fields, whereas adaptation of the topology still has much room for investigation. Traditionally, the growing and pruning techniques for multilayer perceptrons are viewed as a heuristic network designing tool rather than as an integral part of learning itself. However, learning a network topology may be equally important, if not more important, than adjusting the free parameters in the network, as exemplified in biological learning where new synaptic connections in the human brain can grow or die. If the environment is changing over time, then the design of an "optimal" network topology beforehand by conventional methods such as Akaike information criterion, Bayesian information criterion, and minimum description length is not possible.[1] Therefore, it will make more sense to adapt the network structure over time as part of the learning process.[2]

Learning may be performed in three basic ways:

- *Supervised learning*, which requires the availability of a collection of desired (target) responses.

Kernel Adaptive Filtering, By Weifeng Liu, José C. Príncipe, and Simon Haykin

- *Unsupervised learning*, which is performed in a self-organized manner, bypassing the need for a desired response.
- *Reinforcement learning*, which learns through a sequence of state–action–reward and has no explicit input–output available.

In this book, we focus on supervised learning. In conceptual terms, we may think there is a teacher (supervisor) who has knowledge of the environment, where that knowledge is represented by input–desired examples (training data). Well-known supervised learning rules include *error-correction learning* like the Widrow–Hoff rule [Widrow and Hoff, 1960] and *memory-based learning* exemplified by k nearest-neighbor classifiers [Cover and Hart, 1967] and radial-basis function networks [Broomhead and Lowe, 1988].

Historically, machine learning has focused on nonsequential learning tasks, where the training data set can be constructed a priori and learning stops once this set is duly processed. Despite its wide applicability, there are many situations where learning takes place over time. A learning task is *sequential* if the training examples become available over time, usually one at a time. A learning algorithm is *sequential* if, for any given training examples

$$\{\mathbf{u}(1), d(1)\}, \{\mathbf{u}(2), d(2)\}, \ldots, \{\mathbf{u}(i), d(i)\}, \ldots$$

it produces a sequence of *hypotheses*

$$\hbar(1), \hbar(2), \ldots \hbar(i), \ldots$$

such that $\hbar(i)$ depends not only on the previous hypothesis $\hbar(i-1)$, but also on the current example $\{\mathbf{u}(i), d(i)\}$ [Giraud-Carrier, 2000]. Note that sometimes one relaxes slightly the above definition to allow the next hypothesis to depend on the previous one and a small subset of new training examples (rather than a single one) to trade off complexity and performance. It has been long argued that learning involves the ability to improve performance over time [Simon, 1983]. Clearly, humans acquire knowledge over time and knowledge is constantly revised, based on newly acquired information. In the study of robotic and intelligent agents, one finds that sequential learning is a must to deal with complex operating environments, which are usually changing and unpredictable [Brazdil, 1991, Tan, 1993]. Moreover, sequential learning algorithms typically require less computation and memory resources to update the hypothesis, and these algorithms may be the only possibility for large applications.

Active learning is another important concept in machine learning. There are possibly two scenarios where active learning finds applications. In the first case, unlabeled data (input without target) are abundant, but labeling data are expensive. For example, it is easy to get raw speech samples for a speech recognizer, whereas labeling these samples is tedious. The idea of active learning here is to construct an accurate recognizer with significantly fewer labels than you would need in a conventional supervised learning framework. For

these problems, no desired signal (or label) is available to quantify the importance of the candidate data sample.

In the second scenario, a large amount of data have already been gathered in pairs of input and target, but a subset of data must be selected for efficient training and sparse representation. This kind of problem commonly occurs in kernel methods and Gaussian process (GP) modeling.[3] For example, in support vector machines, only data that are close to the boundary are important. The rationale is that training data are not equally informative. Especially in a sequential learning setting, depending on the hypothesis (state) of a learning system, the same data may convey a different amount of information. This is understandable by our daily experience. A message conveys the most amount of information when it is first perceived, and its amount of information diminishes with repeated presentations. These kinds of problems are termed "active data selection" or "sparsification" to distinguish them from the first scenario. We use active learning as a general term referring to both cases.

Active sequential learning is natural. For instance, human beings assess incoming data every day based on knowledge and then decide how much resource should be allocated to process it. Intuitively, for a learning machine, one can expect that after processing sufficient samples from the same source, there is little need to learn, because of redundancy. It has been demonstrated that active learning can significantly reduce the computational complexity with equivalent performance. And it can even provide a more accurate and more stable solution in some situations.[4]

1.2 LINEAR ADAPTIVE FILTERS

Linear adaptive filters built around a linear combiner are designed for sequential learning. When we speak of adaptive filters, we have in mind a class of filtering structures, which are equipped with a built-in mechanism that enables such a filter to adjust its free parameters automatically in response to statistical variations in the environment in which the filter operates. This capability is behind a wide range of applications of adaptive filters in such diverse fields as adaptive equalization in communication receivers, adaptive noise cancellation in active noise control, adaptive beamforming in radar and sonar, system identification, and adaptive control, just to mention a few.[5]

The traditional class of supervised adaptive filters rely on *error-correction learning* for their adaptive capability. To illustrate what we mean by this form of learning, consider the filtering structure depicted in Figure 1.1; a tapped-delay-line (transversal) is most commonly used as the filter, on which the adaptation is performed. The filter embodies a set of adjustable parameters (weights), which is denoted by the vector $\mathbf{w}(i-1)$, where i denotes discrete time. An input signal vector, $\mathbf{u}(i)$, applied to the filter at time i, produces the actual response $y(i)$. This actual response is compared with an externally supplied desired response $d(i)$ to produce the error signal $e(i)$. This error signal

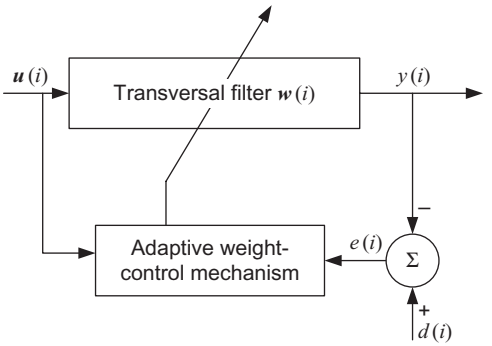

Figure 1.1. Basic structure of a linear adaptive filter.

is, in turn, used to produce an adjustment to the parameter vector $\mathbf{w}(i-1)$ of the filter by an incremental amount denoted by $\Delta\mathbf{w}(i)$. Accordingly, the *updated* parameter vector of the filter assumes the new value

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \Delta\mathbf{w}(i) \tag{1.1}$$

On the next iteration at time $i+1$, $\mathbf{w}(i)$ becomes the latest value of the parameter vector to be updated. The adaptive filtering process is continually repeated in this manner until the filter reaches a condition, whereafter the parameter adjustments become small enough to stop the adaptation. As is clear, the weights here embody the hypothesis in the definition of sequential learning.

Starting from some initial condition denoted by $\mathbf{w}(0)$, the *ensemble-averaged square error*

$$J(i) = \mathbf{E}\left[e^2(i)\right], \quad i = 1, 2, \cdots \tag{1.2}$$

plotted versus time i, traces the *learning curve* of the adaptive filtering process. The expectation in equation (1.2), which is denoted by $\mathbf{E}[\cdot]$, is carried out for an ensemble of different training sets.

An important issue in the design of adaptive filters is to ensure that the learning curve is *convergent* with an increasing number of iterations. Under this condition, we may define the speed of adaptation as the number of iterations i needed for the ensemble-averaged square error, $J(i)$, to reach a relatively "steady-state" value. We may then say that the adaptive filter is *convergent in the mean-square-error sense*.

1.2.1 Least-Mean-Square Algorithm

The simplest and most commonly used form of an adaptive filtering algorithm is the so-called *least-mean-square* (LMS) *algorithm*. Basically, the LMS algorithm operates by minimizing the instantaneous cost function

$$J(i) = \frac{1}{2} e^2(i) \tag{1.3}$$

where the factor 1/2 is introduced to simplify the mathematical formulation. Given that the parameter vector of the filter is $\mathbf{w}(i-1)$, the *prediction error* $e(i)$ is defined by

$$e(i) = d(i) - \mathbf{w}(i-1)^T \mathbf{u}(i) \tag{1.4}$$

Correspondingly, the *instantaneous gradient vector* is given by

$$\frac{\partial}{\partial \mathbf{w}(i-1)} J(i) = -e(i)\mathbf{u}(i) \tag{1.5}$$

Following the instantaneous version of the *method of gradient descent*, the adjustment $\Delta\mathbf{w}(i)$ applied to the algorithm at time i is defined by

$$\Delta\mathbf{w}(i) = \eta e(i)\mathbf{u}(i) \tag{1.6}$$

where η is the *step-size parameter*. Thus, using equation (1.6) in equation (1.1) yields the following update rule for the filter's parameter vector:

$$\underbrace{\mathbf{w}(i)}_{\text{Updated estimate}} = \underbrace{\mathbf{w}(i-1)}_{\text{Old estimate}} + \underbrace{\eta e(i)\mathbf{u}(i)}_{\text{Adjustment}} \tag{1.7}$$

where the prediction error $e(i)$ is itself defined in equation (1.4).

Examining the computations described above, we readily view the basic *simplicity* of the LMS algorithm. Nonetheless, the algorithm can deliver an effective performance, provided that the step-size parameter η is properly chosen. Most importantly, the LMS algorithm is *model independent*, in that no structural restriction was imposed on how the training data were generated. Consequently, the LMS algorithm is known for its robustness. For best performance, the step-size parameter η should be assigned a relatively small value. However, from a practical perspective, such a choice has a serious disadvantage: A small η makes the LMS algorithm converge slowly.

1.2.2 Recursive Least-Squares Algorithm

To overcome the slow rate of adaptation of the LMS algorithm, we may look to another adaptive filtering algorithm: the recursive least-squares (RLS) algorithm.

In a sense, the RLS algorithm follows a rationale similar to the LMS algorithm, in that they are both examples of error-correction learning but with a basic difference:

At every iteration i, the LMS algorithm aims at minimizing the instantaneous value of the squared estimation error $J(i)$ as in equation (1.3), the RLS algorithm aims at minimizing the sum of squared estimation errors up to and including the current time i.

Mathematically, the cost function for the RLS algorithm is defined by

$$J(i) = \sum_{j=1}^{i} \left[d(j) - \mathbf{w}^T \mathbf{u}(j) \right]^2 \tag{1.8}$$

Typically, the RLS algorithm converges to a relatively stable condition an order of magnitude faster than the LMS algorithm. However, this improvement in performance is achieved at some cost, as discussed later in this section.

Considering the operation of the RLS algorithm at time i, let $\mathbf{w}(i-1)$ denote the old estimate of the parameter vector computed at the preceding time instant $i-1$. In an information-theoretic sense, the *state* of the adaptive filter, which is symbolized by the estimate $\mathbf{w}(i-1)$, provides a summary of all the data processed by the RLS algorithm, starting from the initial condition $i = 0$ up to and including time $i-1$. We may, therefore, view the prediction error

$$e(i) = d(i) - \mathbf{w}^T(i-1)\mathbf{u}(i) \tag{1.9}$$

as the new information supplied to the algorithm at time i by the pair of input vector and desired response: $\{\mathbf{u}(i), d(i)\}$. Indeed, it is in light of this statement that the prediction error is referred to as *innovation*.

In the course of working through the derivation of the RLS algorithm, we find that the adjustment supplied to the old estimate $\mathbf{w}(i-1)$ is now defined by

$$\Delta\mathbf{w}(i) = \mathbf{k}(i)e(i) \tag{1.10}$$

where $\mathbf{k}(i)$ is called the *gain vector* of the RLS algorithm. To be specific, $\mathbf{k}(i)$ is defined by

$$\mathbf{k}(i) = \mathbf{P}(i)\mathbf{u}(i) \tag{1.11}$$

where $\mathbf{P}(i)$ is the *state-error correlation matrix*. In fact, the matrix $\mathbf{P}(i)$ is the inverse of the time-averaged correlation matrix, $\mathbf{R}(i)$, of the input vector $\mathbf{u}(i)$, as shown by

$$\mathbf{P}(i) = \mathbf{R}^{-1}(i) \tag{1.12}$$

with $\mathbf{R}(i)$ itself being defined by

$$\mathbf{R}(i) = \sum_{j=1}^{i} \mathbf{u}(j)\mathbf{u}^T(j) \tag{1.13}$$

Accordingly, the updated estimate of the actual parameter vector $\mathbf{w}(i)$ in the RLS algorithm is defined by

$$\underbrace{\mathbf{w}(i)}_{\text{Updated estimate}} = \underbrace{\mathbf{w}(i-1)}_{\text{Old estimate}} + \underbrace{\mathbf{k}(i)e(i)}_{\text{Adjustment}} \tag{1.14}$$

In closing this brief discussion of the LMS and RLS algorithms, we may compare their individual characteristics as follows:

1. Computational complexity of the LMS algorithm scales linearly with the dimension of the parameter vector $\mathbf{w}$, whereas the computational complexity of the RLS algorithm follows a square law.
2. Being model independent, the LMS algorithm is typically more robust than the RLS algorithm.
3. The convergence rate of the RLS algorithm is typically an order of magnitude faster than the LMS algorithm.
4. Last, but by no means least, the LMS algorithm propagates the estimation error from one iteration to the next, whereas the RLS algorithm propagates the error-covariance matrix. This statement is a further testimony to the simplicity of the LMS algorithm.

1.2.3 Extended Recursive Least-Squares Algorithm

The extended recursive least-squares (EX-RLS) algorithm is a special case of a more general algorithm called the *Kalman filter* [Kalman, 1960, Haykin, 2002]. Compared with RLS and LMS, a distinctive feature of the extended recursive least-squares algorithm is that its mathematical formulation is described in terms of *state-space concepts*. Although RLS also embodies the concept of state, the state in RLS is time invariant. Consider a linear dynamic system described by the signal-flow graph shown in Figure 1.2. The state vector

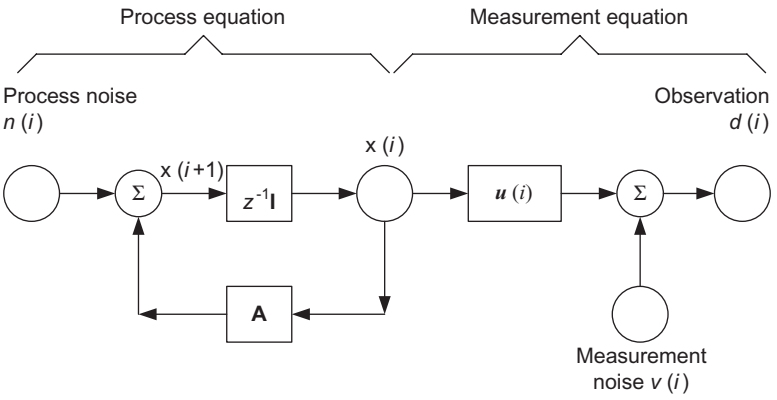

Figure 1.2. Signal-flow graph representation of a linear dynamic system.

denoted by $\mathbf{x}(i)$ in Figure 1.2 is defined as the minimal set of data that is sufficient to describe the unforced dynamic behavior of the system uniquely. In other words, the state comprises the fewest data on the past of the system that are needed to predict its future behavior. Typically, the state $\mathbf{x}(i)$ is unknown and we need to use a set of observations to estimate it.

In mathematical terms, the signal-flow graph of Figure 1.2 embodies the following pair of equations:

1. *A process equation*

$$\mathbf{x}(i+1) = \mathbf{A}\mathbf{x}(i) + \mathbf{n}(i) \tag{1.15}$$

In this equation, the L-by-1 vector $\mathbf{n}(i)$ represents *process noise*, modeled as a zero-mean, white-noise process whose correlation matrix is defined by

$$\mathbf{E}\left[\mathbf{n}(i)\mathbf{n}(j)^T\right] = \begin{cases} \mathbf{Q}_1, & i = j \\ \mathbf{0}, & i \neq j \end{cases} \tag{1.16}$$

The process equation (1.15) models an unknown physical stochastic phenomenon denoted by the L-by-1 state vector $\mathbf{x}(i)$ as the output of a linear dynamic system excited by the white noise $\mathbf{n}(i)$, as depicted in the left-hand portion of Figure 1.2. The linear dynamic system is uniquely characterized by the feedback connection of two units: the L-by-L *transition matrix*, denoted by $\mathbf{A}$, and the *memory unit*, denoted by $z^{-1}\mathbf{I}$, where z^{-1} is the unit time delay and $\mathbf{I}$ is the L-by-L identity matrix. In general, $\mathbf{A}$ can be time variant.

2. *A measurement equation*, which describes the observation as

$$d(i) = \mathbf{u}(i)^T \mathbf{x}(i) + v(i) \tag{1.17}$$

where $\mathbf{u}(i)$ is known and called a *measurement vector* in this setting. The *measurement noise* is modeled as a zero-mean, white noise process with a fixed variance q_2. The measurement equation (1.17) relates the observable output of the system $d(i)$ to the state $\mathbf{x}(i)$, as depicted in the right-hand portion of Figure 1.2. Equation (1.17) bears some resemblance to the linear regression setting in RLS with $\mathbf{u}(i)$ as the input and $d(i)$ as the output. The distinctive difference is that the input–output mapping $\mathbf{x}(i)$ in equation (1.17) is *time variant* and governed by the process equation (1.15), whereas in RLS, it is *time invariant*.

It is assumed that $\mathbf{x}(1)$, which is the initial value of the state, is uncorrelated with both $\mathbf{n}(i)$ and $v(i)$ for $i \geq 1$. Two noise processes $\mathbf{n}(i)$ and $v(i)$ are statistically independent. Parameters $\mathbf{A}$, $\mathbf{Q}_1$, and q_2 are known a priori. With all those assumptions, the EX-RLS algorithm is required to solve the following problem:

Use the entire observations $\{\mathbf{u}(1), d(1)\}$, $\{\mathbf{u}(2), d(2)\}$, …, $\{\mathbf{u}(i), d(i)\}$, to find the minimum mean-square estimate of the state $\mathbf{x}(i+1)$.

We use $\mathbf{w}(i-1)$ to denote the optimal estimate of $\mathbf{x}(i)$ given the observations starting at time $j = 1$ and extending up to and including time $i-1$, i.e., $\{\mathbf{u}(1), d(1)\}$, $\{\mathbf{u}(2), d(2)\}$, …, $\{\mathbf{u}(i-1), d(i-1)\}$, and $\mathbf{w}(i)$ to denote the optimal estimate of $\mathbf{x}(i+1)$ given the observations starting at time $j = 1$ and extending up to and including time i. As in RLS, we define the *innovations process* associated with $d(i)$ as

$$e(i) = d(i) - \mathbf{w}^T(i-1)\mathbf{u}(i) \tag{1.18}$$

where $e(i)$ is the prediction error for the observed $d(i)$ and represents the new information in $d(i)$. The variance of the innovations process $e(i)$ is defined by

$$r(i) = \mathbf{E}\left[e(i)^2\right] = \mathbf{u}(i)^T \mathbf{P}(i-1)\mathbf{u}(i) + q_2 \tag{1.19}$$

where $\mathbf{P}(i-1)$ is the *state-error correlation matrix*. To proceed further, we need to introduce an important concept called *Kalman gain* or simply gain vector here, and a fundamental relation between the current optimal estimate and the previous optimal estimate

$$\underbrace{\mathbf{w}(i)}_{\text{Updated estimate}} = \mathbf{A}\,\underbrace{\mathbf{w}(i-1)}_{\text{Old estimate}} + \underbrace{\mathbf{k}(i)e(i)}_{\text{Adjustment}} \tag{1.20}$$

This equation shows that we can compute the minimum mean-square estimate $\mathbf{w}(i)$ of the state of a linear dynamic system by adding to the previous estimate $\mathbf{w}(i-1)$, which is premultiplied by the transition matrix $\mathbf{A}$, a correction term equal to $\mathbf{k}(i)e(i)$. The correction term equals the prediction error $e(i)$ premultiplied by the gain vector $\mathbf{k}(i)$. This equation is similar to equation (1.14) except for the step of premultiplying the estimate $\mathbf{w}(i-1)$ by the transition matrix $\mathbf{A}$. Furthermore, we can express the gain vector $\mathbf{k}(i)$ as

$$\mathbf{k}(i) = \mathbf{A}\mathbf{P}(i-1)\mathbf{u}(i)/r(i) \tag{1.21}$$

The problem remains of finding a recursive way of computing the state-error correlation matrix $\mathbf{P}(i-1)$. By using the famous *Riccati equation*, we have

$$\mathbf{P}(i) = \mathbf{A}\left[\mathbf{P}(i-1) - \mathbf{A}^{-1}\mathbf{k}(i)\mathbf{u}(i)^T\mathbf{P}(i-1)\right]\mathbf{A}^T + \mathbf{Q}_1 \tag{1.22}$$

Therefore, by initializing $\mathbf{w}(0) = \mathbf{0}$ and $\mathbf{P}(0) = \lambda^{-1}\mathbf{I}$, we can compute $r(1)$, $\mathbf{k}(1)$ and then update $\mathbf{w}(1)$, $\mathbf{P}(1)$ with $\{\mathbf{u}(1), d(1)\}$ supplied. The recursion can go on as long as observations are available. Here, λ is a positive number called the *regularization parameter*, which will be discussed in the subsequent chapters.

As mentioned, EX-RLS is a special case of a Kalman filter that is used in a wide range of engineering applications from radar to aerospace engineering, and it is one of the foundations in modern control theory and control systems engineering. In addition, EX-RLS provides a unifying framework for the derivation of the family of RLS filters using the state-space model. For example, by setting $\mathbf{A} = \beta^{-1/2}\mathbf{I}$ and ignoring $\mathbf{n}(i)$, we have the exponentially weighted RLS algorithm. Furthermore, by setting $\beta = 1$, we have the RLS algorithm itself.

Despite the simple structure of the linear adaptive filters (and probably because of it), they enjoy wide applicability and successes in diverse fields such as communications, control, radar, sonar, seismology, and biomedical engineering, among others. The theory of linear adaptive filters has reached a highly mature stage of development [Haykin, 2002]. However, the same cannot be said about nonlinear adaptive filters, as discussed next.

1.3 NONLINEAR ADAPTIVE FILTERS

The limited computational power of linear learning machines was highlighted by Minsky and Papert [1969] in their famous work on *Perceptrons*. In general, complex real-world applications require more expressive hypothesis spaces than linear functions. Now suppose we lift the linearity assumption, and the goal is to learn a continuous arbitrary input–output mapping $f : \mathbb{U} \to \mathbb{R}$ based on a sequence of examples. Apparently the problem of designing a nonlinear adaptive filter is much harder.[6] One simple way to implement a nonlinear adaptive filter is by cascading a static nonlinearity with a linear filter such as in Hammerstein and Wiener models [Wiener, 1958, Billings and Fakhouri, 1982]. However, in this approach, the modeling capability is limited, the choice of the nonlinearity is highly problem dependent, and there are local minima during training. Gabor [1968] tried using the *Volterra series*[7] to bypass the mathematical difficulties of nonlinear adaptive filtering, whereas it is clear that the complexity of the Volterra series explodes exponentially as its modeling capacity increases. An improvement of the Volterra series is the Wiener series, but slow convergence and high complexity still hinder its wide application. Later on, time-lagged multilayer perceptrons, radial-basis function networks, or recurrent neural networks were used to replace the linear combiner and trained in a real-time fashion by stochastic gradient [Lang and Hinton, 1988, Príncipe et al., 1992] (see Figure 1.3). They have a history of successes, but their nonconvex optimization nature prevents their widespread use in online applications.

Other forms of sequential learning can be found in the *Bayesian learning* literature [Winkler, 2003]. Recursive Bayesian estimation is a general probabilistic approach for estimating an unknown probability density function recursively over time using incoming measurements and a mathematical process model. It has a close connection to the Kalman filter in the adaptive

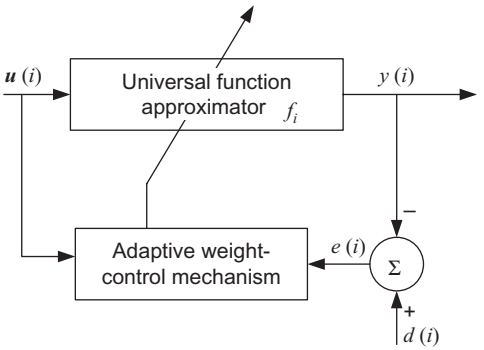

Figure 1.3. Basic structure of a nonlinear adaptive filter.

filtering theory [Kalman, 1960, Roweis and Ghahramani, 1999] but is usually complicated for arbitrary data distributions exemplified by sequential Monte Carlo methods [Doucet et al., 2000a].

Reinforcement learning [Sutton and Barto, 1998] is another area where sequential learning prevails. Reinforcement learning differs from supervised learning in several ways. The most important difference is that there is no presentation of input–output examples. Instead, after choosing an action based on the current state, the algorithm has access to an immediate reward and the subsequent state, but it is not told which action would have been in its best long-term interest. Online performance is crucial in reinforcement learning because the system must act on the environment to evaluate actions (i.e., action evaluation is concurrent with learning). Simply stated, the system learns through a process of reward and punishment without needing to receive a specification of how the task is to be achieved.

In this book, we follow a different course by focusing on a family of nonlinear adaptive filtering algorithms, which have the following features:

- They are universal approximators.
- They have no local minima.
- They have moderate complexity in terms of computation and memory.

In other words, we want to build a nonlinear adaptive filter that possesses the ability of modeling any continuous input–output mapping $y = f(\mathbf{u})$ and obeys the following sequential learning rule:

$$f_i = f_{i-1} + \mathbf{Gain}(i)e(i) \tag{1.23}$$

where f_i denotes the estimate of the mapping at time i and $\mathbf{Gain}(i)$ is a function in general. This sequential learning, which was first studied by Goodwin and Sin [1984] for linear filters, is attractive in practice because the current

estimate consists of two additive parts, namely, the previous estimate and a correction term proportional to the prediction error on new data. This unique incremental nature distinguishes our methods from all the others. Although equation (1.23) seems simple, the algorithm can in fact be motivated by many different objective functions. Also, depending on the precise meanings of **Gain**(i) and $e(i)$, the algorithm can take many different forms. We explore this in detail in the subsequent chapters. This amazing feature is achieved with the underlying linear structure of the reproducing kernel Hilbert space where the algorithms exist, as is discussed next.

1.4 REPRODUCING KERNEL HILBERT SPACES

A *pre-Hilbert space* is an *inner product space* that has an *orthonormal basis* $\{\mathbf{x}_k\}_{k=1}^{\infty}$. Let $\mathbb{H}$ be the largest and most inclusive space of vectors for which the infinite set $\{\mathbf{x}_k\}_{k=1}^{\infty}$ is a basis. Then, vectors not necessarily lying in the original inner product space represented in the form

$$\mathbf{x} = \sum_{k=1}^{\infty} a_k \mathbf{x}_k$$

are said to be spanned by the basis $\{\mathbf{x}_k\}_{k=1}^{\infty}$; the a_k are the coefficients of the representation. Define the new vector

$$\mathbf{y}_n = \sum_{k=1}^{n} a_k \mathbf{x}_k$$

Another vector $\mathbf{y}_m$ may be similarly defined. For $n > m$, we may express the squared Euclidean distance between the vectors $\mathbf{y}_n$ and $\mathbf{y}_m$ as

$$\begin{aligned} \|\mathbf{y}_n - \mathbf{y}_m\|^2 &= \left\| \sum_{k=1}^{n} a_k \mathbf{x}_k - \sum_{k=1}^{m} a_k \mathbf{x}_k \right\|^2 \\ &= \left\| \sum_{k=m+1}^{n} a_k \mathbf{x}_k \right\|^2 \\ &= \sum_{k=m+1}^{n} a_k^2 \end{aligned}$$

where, in the last line, we invoked the orthonormality condition. Therefore, to make the definition of $\mathbf{x}$ meaningful, we need the following to hold:

1. $\sum_{k=m+1}^{n} a_k^2 \to 0$ as both $n, m \to \infty$.
2. $\sum_{k=1}^{m} a_k^2 < \infty$.

In other words, a sequence of vectors $\{\mathbf{y}_k\}_{k=1}^{\infty}$ so defined is a *Cauchy sequence*. Consequently, a vector $\mathbf{x}$ can be expanded on the basis $\{\mathbf{x}_k\}_{k=1}^{\infty}$ if, and only if, $\mathbf{x}$ is a linear combination of the basis vectors and the associated coefficients $\{a_k\}_{k=1}^{\infty}$ are square summable. From this discussion, it is apparent that the space $\mathbb{H}$ is more "complete" than the starting inner product space. We may therefore make the following important statement:

An inner product space $\mathbb{H}$ is complete if every Cauchy sequence of vectors taken from the space $\mathbb{H}$ converges to a limit in $\mathbb{H}$; a complete inner product space is called a Hilbert space.

A *Mercer kernel* [Aronszajn, 1950] is a continuous, symmetric, positive-definite function $\kappa : \mathbb{U} \times \mathbb{U} \rightarrow \mathbb{R}$. $\mathbb{U}$ is the input domain, a subset of $\mathbb{R}^L$. The commonly used kernels include the Gaussian kernel [equation (1.24)] and the polynomial kernel [equation (1.25)]:

$$\kappa(\mathbf{u}, \mathbf{u}') = \exp\left(-a\|\mathbf{u} - \mathbf{u}'\|^2\right) \tag{1.24}$$

$$\kappa(\mathbf{u}, \mathbf{u}') = \left(\mathbf{u}^T\mathbf{u}' + 1\right)^p \tag{1.25}$$

Let $\mathbb{H}$ be any vector space of all real-valued functions of $\mathbf{u}$ that are generated by the kernel $\kappa(\mathbf{u}, \cdot)$. Suppose now two functions $h(\cdot)$ and $g(\cdot)$ are picked from the space $\mathbb{H}$ that are respectively represented by

$$h = \sum_{i=1}^{l} a_i \kappa(\mathbf{c}_i, \cdot)$$

and

$$g = \sum_{j=1}^{m} b_j \kappa(\tilde{\mathbf{c}}_j, \cdot)$$

where the a_i and the b_j are expansion coefficients and both $\mathbf{c}_i$ and $\tilde{\mathbf{c}}_j \in \mathbb{U}$ for all i and j. The *bilinear form* defined as

$$\langle h, g \rangle = \sum_{i=1}^{l} \sum_{j=1}^{m} a_i \kappa(\mathbf{c}_i, \tilde{\mathbf{c}}_j) b_j$$

satisfies the following properties:

1. Symmetry

$$\langle h, g \rangle = \langle g, h \rangle$$

2. Scaling and distributive property

$$\langle (cf + dg), h \rangle = c\langle f, h \rangle + d\langle g, h \rangle$$

3. Squared norm

$$\|f\|^2 = \langle f, f \rangle \geq 0$$

By virtue of these facts, the bilinear term $<h, g>$ is indeed an inner product. There is one additional property that follows directly. Specifically, setting $g(\cdot) = \kappa(\mathbf{u}, \cdot)$, we obtain

$$\begin{aligned} \langle h, \kappa(\mathbf{u}, \cdot) \rangle &= \sum_{i=1}^{l} a_i \kappa(\mathbf{c}_i, \mathbf{u}) \\ &= h(\mathbf{u}) \end{aligned}$$

This property is known as the *reproducing property*. The kernel $\kappa(\mathbf{u}, \mathbf{u}')$, which represents a function of the two vectors $\mathbf{u}, \mathbf{u}' \in \mathbb{U}$, is called a reproducing kernel of the vector space $\mathbb{H}$ if it satisfies the following two conditions:

1. For every $\mathbf{u} \in \mathbb{U}$, $\kappa(\mathbf{u}, \mathbf{u}')$ as a function of the vector $\mathbf{u}'$ belongs to $\mathbb{H}$.
2. It satisfies the reproducing property.

These two conditions are indeed satisfied by the Mercer kernel, thereby endowing it with the designation "reproducing kernel." If the inner product space $\mathbb{H}$, in which the reproducing kernel space is defined, is also complete, then it is called a reproducing kernel Hilbert space (RKHS).

The analytic power of RKHS is expressed in an important theorem called the Mercer theorem. The Mercer theorem [Aronszajn, 1950, Burges, 1998] states that any reproducing kernel $\kappa(\mathbf{u}, \mathbf{u}')$ can be expanded as follows:

$$\kappa(\mathbf{u}, \mathbf{u}') = \sum_{i=1}^{\infty} \varsigma_i \phi_i(\mathbf{u}) \phi_i(\mathbf{u}') \tag{1.26}$$

where ς_i and ϕ_i are the eigenvalues and the eigenfunctions, respectively. The eigenvalues are non-negative. Therefore, a mapping $\boldsymbol{\varphi}$ can be constructed as

$$\begin{aligned} &\boldsymbol{\varphi} : \mathbb{U} \to \mathbb{F} \\ &\boldsymbol{\varphi}(\mathbf{u}) = \left[\sqrt{\varsigma_1} \phi_1(\mathbf{u}), \sqrt{\varsigma_2} \phi_2(\mathbf{u}), \ldots \right] \end{aligned} \tag{1.27}$$

By construction, the dimensionality of $\mathbb{F}$ is determined by the number of strictly positive eigenvalues, which are infinite in the Gaussian kernel case.

In the machine learning literature, $\boldsymbol{\varphi}$ is usually treated as the feature mapping and $\boldsymbol{\varphi}(\mathbf{u})$ is the transformed feature vector lying in the feature space $\mathbb{F}$ (which is an inner product space) (Figure 1.4). By doing so, an important implication is

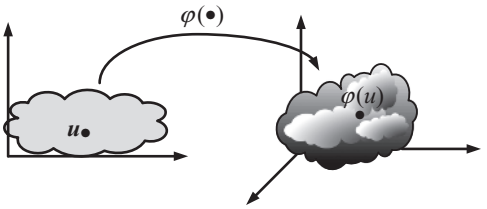

Figure 1.4. Nonlinear map $\boldsymbol{\varphi}(\cdot)$ from the input space to the feature space.

$$\boldsymbol{\varphi}(\mathbf{u})^T \boldsymbol{\varphi}(\mathbf{u}') = \kappa(\mathbf{u}, \mathbf{u}') \tag{1.28}$$

It is easy to check that $\mathbb{F}$ is essentially the same as the RKHS induced by the kernel by identifying $\boldsymbol{\varphi}(\mathbf{u}) = \kappa(\mathbf{u}, \cdot)$, which are the bases of the two spaces, respectively. By slightly abusing the notation, we do not distinguish $\mathbb{F}$ and $\mathbb{H}$ in this book if no confusion is involved.

A concrete example helps here. Let [Cherkassky and Mulier, 1998]

$$\kappa(\mathbf{u}, \mathbf{c}) = \left(1 + \mathbf{u}^T \mathbf{c}\right)^2 \tag{1.29}$$

with $\mathbf{u} = [u_1, u_2]^T$ and $\mathbf{c} = [c_1, c_2]^T$. By expressing the polynomial kernel in terms of monomials of various orders, we have

$$\kappa(\mathbf{u}, \mathbf{c}) = 1 + u_1^2 c_1^2 + 2u_1 u_2 c_1 c_2 + u_2^2 c_2^2 + 2u_1 c_1 + 2u_2 c_2$$

Therefore, the image of the input vector $\mathbf{u}$ in the feature space may be written as

$$\boldsymbol{\varphi}(\mathbf{u}) = \left[1, u_1^2, \sqrt{2}u_1 u_2, u_2^2, \sqrt{2}u_1, \sqrt{2}u_2\right]^T$$

And similarly we have

$$\boldsymbol{\varphi}(\mathbf{c}) = \left[1, c_1^2, \sqrt{2}c_1 c_2, c_2^2, \sqrt{2}c_1, \sqrt{2}c_2\right]^T$$

It is easy to verify that

$$\boldsymbol{\varphi}(\mathbf{u})^T \boldsymbol{\varphi}(\mathbf{c}) = \kappa(\mathbf{u}, \mathbf{c})$$

However, it is hard in general to express $\boldsymbol{\varphi}$ explicitly even for simple polynomial kernels, because the dimensionality of $\boldsymbol{\varphi}$ scales with $O(L^p)$, where L is the dimension of input vectors and p is the order of the polynomial kernel.

1.5 KERNEL ADAPTIVE FILTERS

The *kernel method* is a powerful nonparametric modeling tool. The main idea can be summarized as follows: Transform the input data into a high-dimensional feature space via a reproducing kernel such that the inner product operation in the feature space can be computed efficiently through the kernel evaluations [equation (1.28)]. Then, appropriate linear methods are subsequently applied on the transformed data. As long as an algorithm can be formulated in terms of inner products (or equivalent kernel evaluation), there is no need to perform computations in the high-dimensional feature space. Even though this methodology is called the "kernel trick," we have to point out that the underlying reproducing kernel Hilbert space plays a central role in providing linearity, convexity, and universal approximation capability. Successful examples of this methodology include support vector machines, kernel principal component analysis, and Fisher discriminant analysis.[8]

We start with an example to show why projecting the input into a feature space helps in learning. Consider the target function of a two-dimensional input $\mathbf{u} = [u_1, u_2]^T$.

$$f(u_1, u_2) = a_1 u_1 + a_2 u_2 + a_3 u_1^2 + a_4 u_2^2 \tag{1.30}$$

where a_1, a_2, a_3, and a_4 are some constant coefficients. Apparently, a linear system trying to approximate f by a linear combination of u_1 and u_2 could not exactly model it as written. However, by using the kernel [equation (1.29)] and its mapping $\boldsymbol{\varphi}$, we have a new representation of the input

$$(u_1, u_2) \overset{\varphi}{\mapsto} (x_1, x_2, x_3, x_4, x_5, x_6) = \left(1, u_1^2, \sqrt{2}u_1u_2, u_2^2, \sqrt{2}u_1, \sqrt{2}u_2\right)$$

Now, f can be represented by a linear system of $(x_1, x_2, x_3, x_4, x_5, x_6)$

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = 0 \cdot x_1 + a_3 x_2 + 0 \cdot x_3 + a_4 x_4 + \frac{a_1}{\sqrt{2}} x_5 + \frac{a_2}{\sqrt{2}} x_6$$

The fact that mapping the input into a feature space can simplify the learning task has been well known for a long time in machine learning as exemplified by polynomial regression[9] and Volterra series. The problem is really how to construct this mapping. One may be tempted to add as many features as possible because it is more likely the target functions can be represented using a standard learning algorithm in high-dimensional feature spaces, but this may run into the danger of *overfitting*. Overfitting is a phenomenon where a good fit to the training data is achieved, but the overlearned system performs badly when making test predictions. The difficulties with high-dimensional feature spaces are mainly as follows:

1. The computational complexity explodes with the dimensionality.
2. The generalization performance degrades as the dimensionality increases.

Two approaches can be used to overcome these problems. One is called *feature selection* where only useful features are selected and other features are pruned so that the dimensionality of the feature space is constrained. In our previous example, apparently x_1 and x_3 are two *redundant features* that should be pruned. The other workaround is the kernel method. Because the features are only implicitly constructed and we do not need to work directly in the feature space, the explosion of computational complexity is avoided. The overfitting problem is taken care of by the use of regularization. These properties will become clear when we develop the kernel adaptive filters in the subsequent chapters.

It has been proved [Steinwart, 2001] that in the case of the Gaussian kernel, for any continuous input–output mapping $f : \mathbb{U} \to \mathbb{R}$ and any $\varsigma > 0$, there exist parameters $\{\mathbf{c}_i\}_{i=1}^m$ in $\mathbb{U}$ and real numbers $\{a_i\}_{i=1}^m$ such that

$$\left\| f - \sum_{i=1}^{m} a_i \kappa(\cdot, \mathbf{c}_i) \right\|_2 < \varsigma \tag{1.31}$$

If we denote a vector $\boldsymbol{\omega}$ in $\mathbb{F}$ as

$$\boldsymbol{\omega} = \sum_{i=1}^{m} a_i \boldsymbol{\varphi}(\mathbf{c}_i)$$

then by equations (1.28) and (1.31), we have

$$\left\| f - \boldsymbol{\omega}^T \boldsymbol{\varphi} \right\|_2 < \varsigma$$

This equation implies that the linear model in $\mathbb{F}$ has the *universal approximation property*. Clearly, this property is established from the viewpoint of strict function approximation.

Furthermore, if our problem is to minimize a regularized cost function over a finite data set $\{(\mathbf{u}(i), d(i))\}_{i=1}^N$, then we write

$$\min_f J(f) = \sum_{i=1}^{N} (d(i) - f(\mathbf{u}(i)))^2 + \lambda \|f\|_2^2$$

It has been shown that the optimal solution can be expressed as

$$f = \sum_{i=1}^{N} a_i \kappa(\cdot, \mathbf{u}(i))$$

Table 1.1. Comparison of different nonlinear adaptive filters.

Algorithms	Modeling capacity	Convexity	Complexity
Linear adaptive filters	Linear only	Yes	Very simple
Hammerstein, Wiener models	Limited nonlinearity	No	Simple
Volterra, Wiener series	Universal	Yes	Very high
Time-lagged neural networks	Universal	No	Modest
Recurrent neural networks	Universal	No	High
Kernel adaptive filters	Universal	Yes	Modest
Recursive Bayesian estimation	Universal	No	Very high

for suitable a_i. This result is called the *representer theorem* [Schölkopf et al., 2001]. In other words, although we did consider functions that were expansions in terms of arbitrary points $\mathbf{c}_i$ [see equation (1.31)], it turns out that we can always express the solution in terms of the training points $\mathbf{u}(i)$ only. Hence, the optimization problem over an arbitrarily large number of variables is transformed into one over N variables, where N is the number of training points.

Recently, it also has been shown that Volterra series and Wiener series can be treated just as a special case of a kernel regression framework [Franz and Schölkopf, 2006]. By formulating the Volterra and Wiener series as a linear regression in RKHS, the complexity is now independent of the input dimensionality and the order of nonlinearity.

Based on these advantages and arguments, our strategy is clear: to formulate the classic adaptive filters in RKHS such that we are iteratively solving a convex least-squares problem there. As long as we can formulate these algorithms in terms of inner products, we obtain nonlinear adaptive filters that have the universal approximation property and convexity at the same time. Convexity is an important feature that prevents the algorithms from being stuck in local minima. (See Table 1.1.)

In recent years, many efforts of "kernelizing" adaptive filters have been published in the literature. Frieb and Harrison [1999] first used this idea to derive the kernel ADALINE, which is formulated as a deterministic gradient method based on all the training data (not online). Then, Kivinen et al. [2004] proposed an algorithm called NORMA by directly differentiating a regularized functional cost to get the stochastic gradient. Although the derivation involves advanced mathematics, the results are actually equivalent to a kernel version of the leaky least-mean-square algorithm. At almost the same time, Engel et al. [2004] studied the case of kernel recursive least squares by using the matrix inversion lemma. Later on, Liu et al. [2008] investigated the kernel least-mean-square algorithm and pointed out that the algorithm possesses a self-regularization property. Kernel affine projection algorithms were studied from different perspectives by Liu and Príncipe [2008b], Slavakis and Theodoridis [2008], and Richard et al. [2009]. More recently, the extended

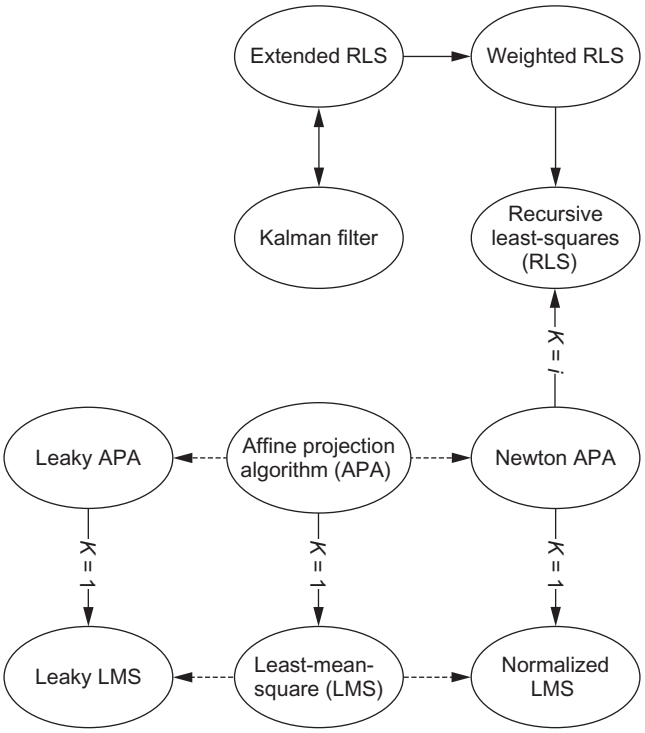

Figure 1.5. Relation among different adaptive filtering algorithms.

kernel recursive least-squares algorithm was presented in Liu et al. [2009], who studied the general state estimation problem in RKHS. After a decade of efforts from many researchers, kernel adaptive filtering is rapidly evolving into an important field of signal processing. This book serves to address previous works and incorporate them in a unifying framework (see Figure 1.5). We present, in detail, the kernel least-mean-square algorithm, the kernel affine projection algorithms, the kernel recursive least-squares algorithm, and the extended kernel recursive least-squares algorithm. Relations among these algorithms, which are illustrated in Figure 1.5, will become clear when we present them in the subsequent chapters.

Kernel adaptive filters provide a generalization of linear adaptive filters because these become a special case of the former when expressed in the dual space. Kernel adaptive filters exhibit a growing memory structure embedded in the filter weights. They naturally create a growing radial-basis function network, learning the network topology and adapting the free parameters directly from data at the same time. The learning rule is a beautiful combination of the error-correction and memory-based learning, and potentially it will have a deep impact on our understanding about the essence of kernel learning theory.

Historically, most kernel methods use block adaptation and are computationally expensive using a large Gram matrix of dimensionality given by the number of data points; therefore, the efficient online algorithms provide the useful flexibility for trading off performance with complexity. And in nonstationary environments, the tracking ability of online algorithms provides an extra advantage.

The combination of sequential learning and memory-based learning requires and at the same time enables, the network to select informative training examples instead of treating all examples equally. Empirical evidence shows that selecting informative examples can reduce drastically the training time and produce much more compact networks with equivalent accuracy. Therefore, in the case of a large and redundant data set, performing kernel online learning algorithms provides a big edge over batch mode methods in terms of efficiency.

The widely used active data selection methods for kernel adaptive filters include the novelty criterion [Platt, 1991] and approximate linear dependency test [Engel et al., 2004]. Both are based on heuristic distance functions while we present a principled and unifying approach. Our criterion is based on a subjective information measure called "surprise". It quantifies how informative the candidate exemplar is relative to the knowledge of the learning system. It turns out that the approximate linear dependency test is a special case and that the novelty criterion is some approximation in this information theoretic framework.

1.6 SUMMARIZING REMARKS

To put the introductory material covered in this chapter into a historical context, it is noteworthy that in a classic paper on the separability of patterns published in 1965, Cover proved that, given a nonlinearly separable pattern-classification problem, there is, in general, a practical benefit to be gained in mapping the input (data) space into a hidden (feature) space of high enough dimensionality. Basically, a nonlinearly separable pattern-classification problem is transformed into a linearly separable one, provided that the following two conditions are satisfied:

1. The transformation from the input space into the feature space is nonlinear.
2. The dimensionality of the feature space is high enough.

Inspired by Cover's insightful ideas just summarized, the following statement was made in Chapter 7 of the first edition of *Neural Networks* [Haykin, 1994, p. 242]:

> In a similar fashion, we may use a nonlinear mapping to transform a difficult nonlinear filtering problem into an easier one that involves linear filtering.

Unfortunately, the extension of Cover's ideas to nonlinear adaptive filtering problems is more difficult because we now have to account for time in an online manner.

Much has been written on the theory and design of nonlinear adaptive filters, going back to an early paper from Gabor et al. [1960]. However, an elegant, unified theory for the algorithmic implementation of nonlinear adaptive filters, which builds on the well-established linear adaptive filters [Widrow and Stearns, 1985, Haykin, 1996], has been lacking. In light of the introductory material presented in this chapter, followed by the detailed expositions presented in the rest of the book on different procedures of implementing kernel adaptive filters, it can be justifiably said that at long last we now have the elegant, unified theory that has been lacking in the literature for much too long.

The new theory presented in this book is elegant because it builds on the Mercer theorem, which is basic to the well-established kernel methods. Moreover, the theory is unified in that it also exploits all the powerful linear adaptive filtering algorithms, namely, the LMS and RLS algorithms and their respective modified versions. Simply stated, the new theory of kernel adaptive filters brings together two different subjects under a single umbrella:

- The Mercer kernel theory, which, in practical terms, manifests itself in the form of memory
- Linear adaptive filter theory, through which the need for adaptation is taken care of

Most importantly, the new theory is developed in a coherent fashion.

ENDNOTES

1. **Model Selection Criteria.** There are mainly three tools for model selection: Akaike information criterion (AIC), Bayesian information criterion (BIC), and minimum description length (MDL). *Akaike's information criterion* was developed by Hirotsugu Akaike under the name of "Akaike information criterion" in 1971 and proposed in Akaike [1974]. AIC is a measure of the goodness of fit of an estimated statistical model, which is defined as

$$\mathrm{AIC} = 2k - 2\ln(L_{\max}) \tag{1.32}$$

where k is the number of the free parameters in the model and $L_{\max}$ is the maximized value of the likelihood function for the model. Given a data set, several competing models may be ranked according to their AIC; the one with the lowest AIC is the best. If the model errors are normally and independently distributed, then AIC may simplify to

$$\mathrm{AIC} = 2k + 2N\ln(\mathrm{MSE}) \tag{1.33}$$

where N is the number of data points and MSE is the mean square error of the data by using the model. Equation (1.33) consists of two terms: measure of the goodness of fit and penalty of the model complexity. In this sense, the AIC methodology attempts to find the model that best explains the data with a minimum of free parameters.

Bayesian information criterion was developed by Schwarz [1978]; it is closely related to the Akaike information criterion. The formula for BIC is

$$\text{BIC} = k \ln(N) - 2\ln(L_{\max}) \tag{1.34}$$

Under the assumption that the model errors are Gaussian distributed, this equation becomes

$$\text{BIC} = k \ln(N) + N \ln(\text{MSE}) \tag{1.35}$$

The *Minimum description length* was introduced by Rissanen [1978] and systematically studied by Grünwald [2007]. The MDL principle is a formalization of Occam's Razor, in which the best hypothesis for a given set of data is the one that leads to the largest compression of the data. The ideal MDL approach requires the estimation of the Kolmogorov complexity, which is uncomputable in general. However, nonideal, practical versions of MDL are widely used in the machine learning community; see for example Grünwald [2007], and Haykin [2009].

2. **Growing and Pruning Neural Networks.** The *adaptive resonance theory* architecture [Grossberg, 1987, Carpenter and Grossberg, 1987] is one of the first growing neural networks. It is a biologically inspired approach rather than a computational design. However, it is often inefficient and the convergence of the iterative processing is not guaranteed.

   The *cascade-correlation learning architecture* [Fahlman and Lebiere, 1990] is another example of the network-growing approach. The procedure begins with a minimal network that has some input and one or more output nodes as indicated by input–output considerations, but no hidden nodes. The hidden neurons are added to the network one by one, thereby obtaining a multilayer structure.

   Another network-growing approach is described in Lee et al. [1990], where a third level of computation is added to the forward pass (function-level adaptation) and the backward pass (parameter-level adaptation). In this third level of computation, the structure of the network is adapted by changing the number of neurons and the structural relationship among neurons in the network. This method of network growing is computationally intensive.

   Platt [1991] proposed a more feasible design called *resource allocating networks*, where the structure of a neural network was dynamically altered to optimize resource allocation. Since then, many researchers have proposed methods of both growing and pruning radial-basis function networks reported in Cheng and Lin [1994], Karayiannis and Mi [1997], and Huang et al. [2005].

   Martinetz and Schulten [1991] started a new strand of research into growing networks by inventing the "neural gas" models, where the topologies were not predetermined and connections between nodes were added as needed. These "neural gas" models are self-organizing systems that use Hebb-like learning rule to learn the distribution of input data.

Two main approaches to network pruning are 1) regularization, of which notable examples are *weight decay* [Hinton and Sejnowski, 1986], *weight elimination* [Weigend et al., 1990], and *approximate smoother* [Moody and Rögnvaldsson, 1997]; and 2) systematic deletion, which includes the *optimal brain damage* procedure [LeCun et al., 1990] and the *optimal brain surgeon* procedure [Hassibi and Stork, 1992].

3. **Sparsification.** In kernel methods and Gaussian process modeling, the complexity is usually directly proportional to the number of training data either at linear scale, quadratic scale, or even cubic scale. Sparsification is a process of selecting only an "important" subset of the training data to train the model, and by so doing the complexity of the algorithm can be reduced greatly. There are two main approaches. One is by *elimination*, as in support vector machines [Vapnik, 1995], regularization networks [Evgeniou et al., 2000], relevance vector machines [Tipping, 2001], and least-squares support vector machines [Suykens et al., 2000]. These algorithms start by considering all training samples as potential centers. And part of the samples are eliminated by solving the optimization problem wherein the associated coefficients become zero. These algorithms are usually computationally expensive, scaling with $O(N^3)$ in time and up to $O(N^2)$ in space, where N is the number of training examples.

   The other approach is by *construction*. Here, the algorithm starts with an empty network and gradually adds centers at each step of the construction process. Because finding the best subset is a combinational optimization problem, these algorithms usually employ various greedy selection strategies, in which at each step the sample that maximizes the amount of some fitness criterion is selected [Seeger and Williams, 2003, Smola and Bartlett, 2001, Quinonero-Candela and Rasmussen, 2005]. Still, the complexity of these algorithms ranges from $O(M^2N)$ to $O(MN^2)$, where $M(<<N)$ is the size of the selected subset (or the number of basis functions). It usually requires multiple passes of the whole training data. If computer memory cannot hold the whole training data, then disk-read operations would slow down the learning speed significantly.

4. **Active learning.** Active learning has been studied under the names of *optimal experiment design* [Lindley, 1956, Fedorov, 1972], *sequential decision making* [El-Gamal, 1991], *query learning* [Campbell et al., 2000], and *selective sampling* [Lindenbaum, 1999] in such diversified fields as economics theory, statistics, and machine learning. The uses of active learning in neural networks are reported in MacKay [1992a], Fukumizu [1996], and Tong and Koller [2000]. Active learning in sequential methods has been studied in Platt [1991], Engel et al. [2004], and Csato and Opper [2002] for regression problems and in Bordes et al. [2005] and Glasmachers [2008] for classification problems.

5. **Linear Adaptive Filters.** The earliest work on adaptive filters may be traced back to the late 1950s, during which time many researchers were working independently on different applications of such filters. As a result, the LMS emerged as a simple and effective algorithm for the operation of adaptive transversal filters. The LMS algorithm was devised by Widrow and Hoff in 1959 in their study of an adaptive linear element, which is commonly referred to in the literature as the *ADALINE* [Widrow and Hoff, 1960].

   Another important algorithm in adaptive filtering theory is the RLS algorithm. The original paper on the standard RLS algorithm is that of Plackett [1950], although many other researchers are believed to have derived and rederived various versions

of the RLS algorithm. In 1974, Godard first used Kalman filter theory successfully to solve adaptive filtering problems, which is known in the literature as the *Godard algorithm*. Then, Sayed and Kailath [1994] established an exact relationship between the RLS algorithm and Kalman filter theory, thereby laying the groundwork for how to exploit the vast literature on Kalman filters for solving linear adaptive filtering problems.

The EX-RLS algorithm was first presented in Haykin et al. [1997]. It is derived as an improvement over the RLS algorithm in terms of tracking ability in nonstationary signal processing. The algorithm is a special case of the *Kalman filter* while derived from the perspective of adaptive signal processing. The Kalman filter is used in a wide range of engineering applications from radar to navigation, and it is an important topic in control theory and control systems engineering. The filter is named after Rudolf E. Kalman based on his seminal papers [Kalman, 1960, Kalman and Bucy, 1961], although Thorvald Nicolai Thiele and Peter Swerling are found to have developed a similar algorithm earlier [Lauritzen, 1981]. A highlight application of the Kalman filter is perhaps its incorporation in the Apollo navigation computer.

Two excellent textbooks for linear adaptive filtering were authored by Haykin [2002] and Sayed [2003].

6. **Nonlinear Adaptive Filters.** The idea of nonlinear adaptive filter was proposed by Gabor in 1954, who was one of the early pioneers of communications theory. Hammerstein and Wiener models are discussed in Wiener [1958], Billings and Fakhouri [1982], and Marmarelis [1993]. Volterra [1887] and later Gabor [1968] studied the use of Volterra series for nonlinear adaptive filtering. The Wiener series was proposed and studied in Wiener [1958] and Barrett [1963] as an improvement over the Volterra series. For the use of time-lagged multilayer perceptrons, radial-basis function networks, and recurrent neural networks for nonlinear adaptive filtering, see Lang and Hinton [1988], Wan [1990], Príncipe et al. [1992], and Haykin [1998].

7. **Volterra Series.** The Volterra series is named after the Spanish mathematician Vito Volterra, who first introduced the notion in 1887 [Volterra, 1887]. The first major application of Volterra's work to nonlinear circuit analysis was done by the mathematician Norbert Wiener at the Massachusetts Institute of Technology, who used them in a general way to analyze several problems including the spectrum of an FM system with a Gaussian noise input [Wiener, 1958].

   The Volterra series is a model for a nonlinear, time-invariant system with memory. It is similar to the Taylor series in terms of nonlinear modeling but differs from the Taylor series in its ability to capture "memory" effects. The Taylor series can be used to approximate the nonlinear response of a system to a given input:

$$y(t) = \sum_{n=0}^{\infty} a_n [x(t)]^n$$

   where the output $y(t)$ depends strictly on the input $x(t)$ at that particular time. In the Volterra series, the output of the nonlinear system depends on all the input that has been applied to the system in the past. This provides the ability to capture the "memory" effect of devices such as capacitors and inductors. A linear, causal system with memory can be described by the convolution representation:

$$y(t) = \int_{-\infty}^{\infty} h(\tau) x(t-\tau) d\tau$$

where $y(t)$ is the output, $x(t)$ is the input, and $h(t)$ is the impulse response of the system. The Volterra series combines these two techniques to describe a nonlinear, dynamic, time-invariant system in the following way:

$$\begin{aligned} y(t) &= \sum_{n=0}^{\infty} \frac{1}{n!} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} k_n(\tau_1, \tau_2, \ldots, \tau_n) x(t-\tau_1) x(t-\tau_2) \cdots x(t-\tau_n) d\tau_1 d\tau_2 \cdots d\tau_n \\ &= k_0 \\ &\quad + \frac{1}{1!} \int_{-\infty}^{\infty} k_1(\tau_1) x(t-\tau_1) d\tau_1 \\ &\quad + \frac{1}{2!} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} k_2(\tau_1, \tau_2) x(t-\tau_1) x(t-\tau_2) d\tau_1 d\tau_2 \\ &\quad + \frac{1}{3!} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} k_3(\tau_1, \tau_2, \tau_3) x(t-\tau_1) x(t-\tau_2) x(t-\tau_3) d\tau_1 d\tau_2 d\tau_3 \\ &\quad + \cdots \end{aligned} \tag{1.36}$$

where the $k_n(\tau_1, \tau_2, \ldots, \tau_n)$ are called the *Volterra kernels* of the system. For $n = 1$, $k_1(\tau_1)$ is the conventional impulse response like in the linear system; for $n > 1$, k_n are regarded as "higher order impulse responses."

The determination of the Volterra kernels are generally complicated. Common methods include the *harmonic input method*, *direct expansion method*, and *powers of transfer function method*. The Volterra series is successful to model systems that exhibit "weak nonlinearity." If the system to be modeled is "strongly nonlinear," then the Volterra series either takes a long time to converge or often diverges. For more details, read Cherry [1994] and Schetzen [2006].

8. **Kernel Methods.** The idea of using kernel functions as inner products in a feature space was introduced into machine learning by the work of Aizerman et al. [1964] on the method of potential functions. Then, Boser et al. [1992] combined the idea with large margin classifiers, leading to the birth of support vector machines and the popularity of kernel in the machine learning community. Schölkopf et al. [1998] derived the first unsupervised learning algorithm in reproducing kernel Hilbert space by introducing the kernel principal components analysis. The description of kernel Fisher discriminant analysis can be found in Mika et al. [1999]. The use of kernels for function approximation dates back to Aronszajn [1950]. Then, Wahba [1990] systematically studied reproducing kernels in approximation and regularization theory. At the same time, Poggio and Girosi [1990] used reproducing kernels in the development of regularization networks. Excellent tutorial books on kernel methods include Cristianini and Shawe-Taylor [2000], Shawe-Taylor and Cristianini [2004], and Schölkopf and Smola [2002]. More advanced reading on support vector machines includes Vapnik's books on statistical learning theory [Vapnik, 1995, 1998].
9. **Polynomial Regression.** The first use of polynomial regression was published by Gergonne [Stigler, 1974]. The polynomial regression is a simple generalization of the linear regression model. The goal of regression analysis is to model the expected value of a dependent variable y in terms of the value of an independent variable x. In simple linear regression, the model is

$$y = a_0 + a_1 x + \varepsilon$$

where ε is random noise with zero mean. Similarly a quadratic model is

$$y = a_0 + a_1 x + a_2 x^2 + \varepsilon$$

In general, we can use an nth-order polynomial to model the mapping between two scalar variables y and x:

$$y = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n + \varepsilon$$

The advantage of the method is it is easy to estimate the linear coefficients a_0, a_1, …, a_n using the least-squares analysis. Polynomial regression belongs to a more general function approximation framework using basis functions, which include splines, radial-basis functions, and wavelets. The drawbacks of polynomial bases are 1) they are correlated and 2) they are nonlocal. These drawbacks lead to practical difficulties in terms of interpretation and stability.