CHAPTER 2
KERNEL LEAST MEAN SQUARE ALGORITHM

The great appeal of developing filters in RKHS is to utilize the linear structure of this space to implement well-established linear adaptive algorithms and to obtain nonlinear filters in the input space. When compared with neural networks, this alternative design approach leads to universal approximation capabilities, convex optimization (i.e. no local minima), and computational complexity that is still reasonable. It holds a unique position by bridging two important areas of adaptive filtering and neural networks. The bottleneck of the RKHS approach to nonlinear filter design is the need for regularization, the need to select the kernel function, and the need to curtail the growth of the filter structure.

In particular, in this chapter we study the kernel least-mean-square algorithm (KLMS), the simplest among the family of the kernel adaptive filters in Figure 1-5. The linear LMS algorithm will be directly mapped into RKHS with an emphasis on the general methodology to formulate linear filters and gradient descent algorithms in terms of inner products that can immediately take advantage of the reproducing property of RKHS and be directly implemented by kernel evaluations. This is one of the critical steps in the overall design; otherwise, the RKHS methodology loses its edge because of the insurmountable computational complexity of operating with an infinite number of parameters.

Another important aspect for understanding is to show how the nonlinear mapping is incrementally constructed during adaptation. The KLMS algorithm naturally creates a growing radial-basis function network, learning network topology and adapting free parameters directly from training data. Kernel filtering is a memory intensive operation just like other kernel methods. However, kernel filtering is online and the filter output is incrementally constructed by using previous samples and prediction errors.

The material presented in this chapter also includes a detailed analysis of the KLMS self-regularization property. Thanks to its gradient descent nature, KLMS does not

need extra solution norm constraint in contrast to the majority of kernel methods. This simplifies even further the implementation, and provides a rather practical nonlinear filter design.

2.1 Least-Mean-Square Algorithm

Suppose the goal is to learn a continuous input-output mapping $f : \mathbb{U} \rightarrow \mathbb{R}$ based on a sequence of input-output examples $\{\mathbf{u}(1), d(1)\}$, $\{\mathbf{u}(2), d(2)\}$,..., $\{\mathbf{u}(N), d(N)\}$. $\mathbb{U}$ is the input domain and is assumed as a subspace of $\mathbb{R}^L$. The output is assumed to be one-dimensional but it is straightforward to generalize the discussion to multidimensional output. N is the size of training data; the problem of sequential learning with infinite training data will be addressed later.

The LMS algorithm assumes a linear model and uses the following procedure

$$\begin{aligned} &\mathbf{w}(0) = 0 \\ &e(i) = d(i) - \mathbf{w}(i-1)^T\mathbf{u}(i) \\ &\mathbf{w}(i) = \mathbf{w}(i-1) + \eta e(i)\mathbf{u}(i) \end{aligned} \tag{2–1}$$

to approximately find the optimal weight $\mathbf{w}^o$, which minimizes the empirical risk:

$$J(\mathbf{w}) = \sum_{i=1}^{N} (d(i) - \mathbf{w}^T\mathbf{u}(i))^2$$

In equation (2–1), $e(i)$ is called the prediction error, η is the step-size parameter and $\mathbf{w}(i)$ is the estimate of the optimal weight at iteration i. LMS can be derived by using the *instantaneous gradient*. The gradient of the cost function with respect to $\mathbf{w}$ is

$$\nabla_{\mathbf{w}} J = -2\sum_{i=1}^{N} \mathbf{u}(i)(d(i) - \mathbf{w}^T\mathbf{u}(i)) \tag{2–2}$$

and the instantaneous gradient at time i is

$$\widehat{\nabla_{\mathbf{w}} J} = -2\mathbf{u}(i)(d(i) - \mathbf{w}(i-1)^T\mathbf{u}(i)) \tag{2–3}$$

by dropping the summation. Finally, according to the method of steepest descent, we may formulate the LMS algorithm as follows:

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \eta \mathbf{u}(i)(d(i) - \mathbf{w}(i-1)^T \mathbf{u}(i)) \tag{2–4}$$

For this reason, the LMS algorithm is sometimes referred to as a "stochastic gradient algorithm". The LMS algorithm is summarized in Algorithm 1, which clearly illustrates the simplicity of the algorithm. As indicated in the algorithm, for the *initialization* of the algorithm, it is customary to set the initial value of the weight vector equal to zero.

Algorithm 1 The Least Mean Square Algorithm

Initialization
$\mathbf{w}(0) = 0$, choose η
Computation
while $\{\mathbf{u}(i), d(i)\}$ available **do**
  $e(i) = d(i) - \mathbf{w}^T(i-1)\mathbf{u}(i)$
  $\mathbf{w}(i) = \mathbf{w}(i-1) + \eta e(i)\mathbf{u}(i)$
end while

At iteration i, given a test point $\mathbf{u}_*$, the output of the system is

$$f(\mathbf{u}_*) = \mathbf{u}_*^T \mathbf{w}(i).$$

Convergence Considerations of the LMS Algorithm

The first criterion for convergence of the LMS algorithm is *convergence of the mean*, described by

$$\mathbf{E}[\mathbf{w}(i)] \to \mathbf{w}^o \quad \text{as } i \to \infty, N \to \infty \tag{2–5}$$

However, this criterion is too weak to be of any practical value, since a sequence of zero-mean, but otherwise arbitrary random vectors converges in this sense.

A more practical convergence criterion is *convergence in the mean square*, described by

$$\mathbf{E}[e(i)^2] \to \text{constant} \quad \text{as } i \to \infty, N \to \infty \tag{2–6}$$

Under the assumption that the step-size parameter η is sufficiently small, it is shown in [Haykin, 2002] that the LMS is convergent in the mean square provided that η satisfies the condition

$$0 < \eta < \frac{1}{\varsigma_{max}} \tag{2–7}$$

where ς_{max} is the *largest eigenvalue* of the correlation matrix $\mathbf{R_u}$, defined by

$$\mathbf{R_u} = \sum_{i=1}^{N} \mathbf{u}(i)\mathbf{u}(i)^T \tag{2–8}$$

In typical applications of the LMS algorithm, knowledge of ς_{max} is not available. To overcome this difficulty, the *trace* of $\mathbf{R_u}$ may be taken as a conservative estimate for ς_{max}. Therefore, we have the following conservative condition

$$0 < \eta < \frac{1}{\mathrm{tr}[\mathbf{R_u}]} \tag{2–9}$$

Misadjustment of the LMS Algorithm

Another important parameter of the LMS algorithm is called the *misadjustment*, which is formally defined as

$$\mathcal{M} = \frac{J(\infty) - J_{min}}{J_{min}} \tag{2–10}$$

where $J(\infty)$ is the limiting constant of the mean square error $\mathbf{E}[e(i)^2]$ as i goes to ∞ and J_{min} is the irreducible error power caused by noise in the observations. In words, the misadjustment is defined as the ratio of the steady-state value of the excess mean-square error to the minimum mean-square error. Under the small step-size theory, we may also write

$$\mathcal{M} = \frac{\eta}{2} \sum_{i=1}^{L} \varsigma_i \tag{2–11}$$

which, by the eigen-decomposition theory, is equivalent to

$$\mathcal{M} = \frac{\eta}{2} \mathrm{tr}[\mathbf{R_u}] \tag{2–12}$$

The misadjustment is a dimensionless parameter that provides a measure of how close the LMS algorithm is to optimality in the mean-square-error sense. The smaller the misadjustment is compared with unity, the more accurate is the adaptive filtering action being performed by the LMS algorithm. It is customary to express misadjustment as a percentage. For example, a misadjustment of 10% means that the LMS algorithm produces a mean-square error (after adaptation is completed) that is 10% greater than the minimum mean-square error J_{min}. Such performance is ordinarily considered to be satisfactory in practice.

Learning Curve

Learning curve is an informative way of examining the convergence behavior of the LMS algorithm, or in general any adaptive filter. We will use the learning curve a great deal in our experiments to compare the performance of different adaptive filters. The learning curve is a plot of the mean square error (MSE), $\mathbf{E}[e(i)^2]$, versus the number of iterations, i. There are mainly two ways to obtain the estimate of $\mathbf{E}[e(i)^2]$, the *ensemble-average* approach and the testing mean square error approach.

To obtain the ensemble-averaged learning curve, we need an ensemble of adaptive filters, with each filter operating with the same configuration settings such as updating rule, step-size parameter and initialization. The input and desired signals are independent for each filter. For each filter, we plot the *sample* learning curve, which is simply the squared value of the estimation error $e(i)^2$ (notice there is no expectation operator here) versus the number of iterations. The sample learning curve so obtained consists of noisy components due to the inherently stochastic nature of the adaptive filter. Then we take the average of these sample learning curves over the ensemble of adaptive filters used in the experiment, thereby smoothing out the effects of noise. The averaged learning curve so obtain is called the ensemble-averaged learning curve. This method is applicable for any environment, stationary or nonstationary.

The other approach is by setting aside a testing data set before the training. For each iteration, we have the weight estimate $\mathbf{w}(i)$. We compute the mean square error on the testing data set by using $\mathbf{w}(i)$. Then we plot the testing MSE versus the number of iterations. This approach only needs one adaptive filter and is computationally cheaper comparing with the ensemble-average approach. However, this method does not apply in situations where the environment is nonstationary.

2.2 Kernel Least Mean Square Algorithm

A *linear finite impulse response filter* is assumed in the LMS algorithm. If the mapping between d and $\mathbf{u}$ is highly nonlinear, very poor performance can be expected from LMS. To overcome the limitation of linearity, we are well motivated to formulate a "similar" algorithm which is capable of learning arbitrary nonlinear mappings. For that purpose, the kernel-induced mapping (1–27) is employed to transform the input $\mathbf{u}(i)$ into a high-dimensional feature space $\mathbb{F}$ as $\boldsymbol{\varphi}(\mathbf{u}(i))$. As we discussed in Chapter 1, $\boldsymbol{\omega}^T\boldsymbol{\varphi}(\mathbf{u})$ is a much more powerful model than $\mathbf{w}^T\mathbf{u}$ due to the difference in dimensionality (more importantly the richness of representation) of $\mathbf{u}$ and $\boldsymbol{\varphi}(\mathbf{u})$. So finding $\boldsymbol{\omega}$ through stochastic gradient descent may prove as an effective way of nonlinear filtering as LMS does for linear problems. Denote $\boldsymbol{\varphi}(i) = \boldsymbol{\varphi}(\mathbf{u}(i))$ for simplicity. Using the LMS algorithm on the new example sequence $\{\boldsymbol{\varphi}(i), d(i)\}$ yields

$$\begin{aligned} &\boldsymbol{\omega}(0) = 0 \\ &e(i) = d(i) - \boldsymbol{\omega}(i-1)^T\boldsymbol{\varphi}(i) \\ &\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta e(i)\boldsymbol{\varphi}(i) \end{aligned} \tag{2–13}$$

where $\boldsymbol{\omega}(i)$ denotes the estimate (at iteration i) of the weight vector in $\mathbb{F}$. We can see the direct correspondence between (2–1) and (2–13).

However, the dimensionality of $\boldsymbol{\varphi}$ is very high (infinity in the case of the Gaussian kernel) and $\boldsymbol{\varphi}$ is only implicitly known (it is the kernel's eigenfunctions), so we need an alternative way of carrying out the computation. The repeated application of the

weight-update equation (2–13) through iterations yields

$$\begin{aligned}\boldsymbol{\omega}(i) &= \boldsymbol{\omega}(i-1) + \eta e(i)\boldsymbol{\varphi}(i) \\ &= [\boldsymbol{\omega}(i-2) + \eta e(i-1)\boldsymbol{\varphi}(i-1)] + \eta e(i)\boldsymbol{\varphi}(i) \\ &= \boldsymbol{\omega}(i-2) + \eta[e(i-1)\boldsymbol{\varphi}(i-1) + e(i)\boldsymbol{\varphi}(i)] \\ &\dots \\ &= \boldsymbol{\omega}(0) + \eta \sum_{j=1}^{i} e(j)\boldsymbol{\varphi}(j) \\ &= \eta \sum_{j=1}^{i} e(j)\boldsymbol{\varphi}(j) \quad (\text{assuming } \boldsymbol{\omega}(0) = 0)\end{aligned} \tag{2–14}$$

that is, after i-step training, the weight estimate is expressed as a linear combination of all the previous and present (transformed) inputs, weighted by the prediction errors (and scaled by η). More importantly, the output of the system to a new input $\mathbf{u}'$ can be solely expressed in terms of inner products between transformed inputs

$$\begin{aligned}\boldsymbol{\omega}(i)^T\boldsymbol{\varphi}(\mathbf{u}') &= [\eta \sum_{j=1}^{i} e(j)\boldsymbol{\varphi}(\mathbf{u}(j))^T]\boldsymbol{\varphi}(\mathbf{u}') \\ &= \eta \sum_{j=1}^{i} e(j)[\boldsymbol{\varphi}(\mathbf{u}(j))^T\boldsymbol{\varphi}(\mathbf{u}')]\end{aligned} \tag{2–15}$$

Now by the kernel trick (1–28) we can efficiently compute the filter output in the input space by kernel evaluations

$$\boldsymbol{\omega}(i)^T\boldsymbol{\varphi}(\mathbf{u}') = \eta \sum_{j=1}^{i} e(j)\kappa(\mathbf{u}(j), \mathbf{u}') \tag{2–16}$$

It is important to stop here and compare this equation with the weight update of LMS (2–1). The new algorithm is computed without using the weights. Instead we have the sum of all past errors multiplied by the kernel evaluations on the previously received data, which is equivalent to the weights as can be seen in (2–14). Therefore, having direct access to the weights enables the computation of the output with a single inner product which is a huge time saving, but the two procedures are actually equivalent.

If f_i is denoted as the estimate of the input-output nonlinear mapping at time i, we have the following sequential learning rule for the new algorithm:

$$\begin{aligned} f_{i-1} &= \eta \sum_{j=1}^{i-1} e(j)\kappa(\mathbf{u}(j), \cdot) \\ f_{i-1}(\mathbf{u}(i)) &= \eta \sum_{j=1}^{i-1} e(j)\kappa(\mathbf{u}(j), \mathbf{u}(i)) \\ e(i) &= d(i) - f_{i-1}(\mathbf{u}(i)) \\ f_i &= f_{i-1} + \eta e(i)\kappa(\mathbf{u}(i), \cdot) \end{aligned} \tag{2–17}$$

We call the new algorithm *Kernel Least Mean Square (KLMS)*. It is the LMS in RKHS and filtering is done by kernel evaluation. KLMS allocates a new kernel unit for the new training data with input $\mathbf{u}(i)$ as the center and $\eta e(i)$ as the coefficient. The coefficients and the centers are stored in memory during training. The algorithm is summarized in Algorithm 2 and illustrated in Figure 2-1. $\mathbf{a}(i)$ is the coefficient vector at iteration i, $\mathbf{a}_j(i)$ its jth component and $\mathcal{C}(i)$ the corresponding set of centers. At iteration i, given a test input point $\mathbf{u}_*$, the output of the system is

$$f(\mathbf{u}_*) = \eta \sum_{j=1}^{i} e(j)\kappa(\mathbf{u}(j), \mathbf{u}_*). \tag{2–18}$$

The KLMS topology reminds us of a radial-basis function (RBF) network[1], with three major differences: first, the output weights are essentially the scaled prediction errors at each sample; second, this is a *growing network* where each new unit is placed over each new input; third, κ is not limited to be a radial-basis function and can be any Mercer kernel.

KLMS is a rather simple algorithm, which requires $O(i)$ operations per filter evaluation and weight update, but we need to pay attention to several aspects that are still unspecified. The first is how to select the kernel κ, the second is how to select the step-size parameter η, and finally how to cope with the growing memory/computation requirement for online operation.

Algorithm 2 The Kernel Least Mean Square Algorithm

Initialization

choose step-size parameter η and kernel κ

$\mathbf{a}_1(1) = \eta d(1)$, $\mathcal{C}(1) = \{\mathbf{u}(1)\}$, $f_1 = \mathbf{a}_1(1)\kappa(\mathbf{u}(1), \cdot)$

Computation

while $\{\mathbf{u}(i), d(i)\}$ available **do**

%compute the output

$f_{i-1}(\mathbf{u}(i)) = \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa(\mathbf{u}(i), \mathbf{u}(j))$

%compute the error

$e(i) = d(i) - f_{i-1}(\mathbf{u}(i))$

%store the new center

$\mathcal{C}(i) = \{\mathcal{C}(i-1), \mathbf{u}(i)\}$

%compute and store the coefficient

$\mathbf{a}_i(i) = \eta e(i)$

end while

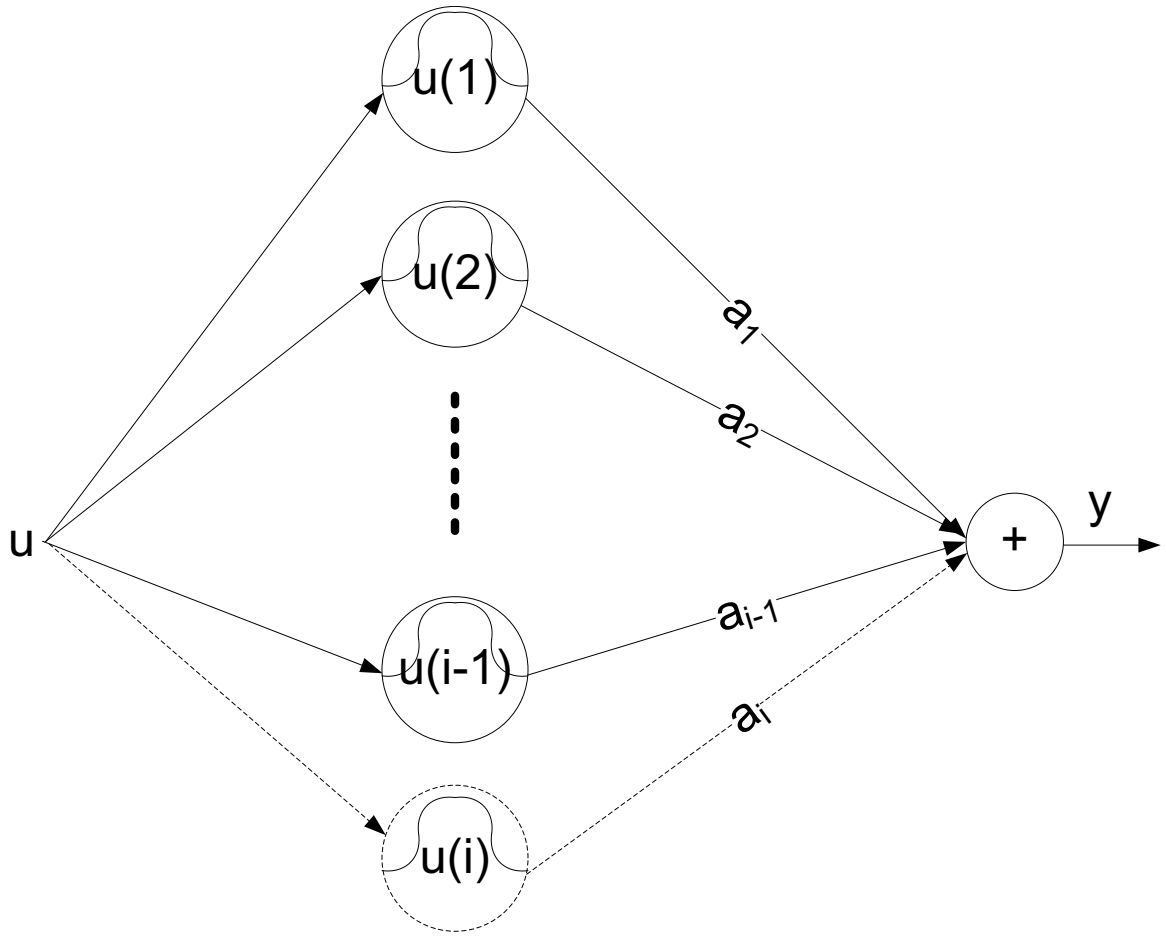

Figure 2-1. Network topology of KLMS at iteration i

2.3 Kernel and Parameter Selection

The necessity of specifying the kernel and its parameter applies to all kernel methods and it is reminiscent of nonparametric regression, where the weight function and its smoothing parameter must be chosen. The kernel is a crucial ingredient of any kernel method in the sense that it defines the *similarity* between data points. An exhaustive treatment on this topic is out of the scope of the book[2]. In the following, we provide a brief and engineering-oriented discussion.

First and foremost, we need to pick a kernel. In the literature of nonparametric regression, it is known that any bell-shaped weight function (Gaussian function, tricube function, etc) leads to equivalent asymptotic accuracy. However, weight functions are not necessarily reproducing kernels and vice versa. For example, the polynomial kernel (1–25) is not bell-shaped and cannot be considered as a weight function. The RKHS approach examines more closely the eigenfunctions of the kernel and its richness for approximation. It is known that the Gaussian kernel (among many others such as the Laplacian) creates a reproducing kernel Hilbert space with universal approximating capability while the polynomial kernel of finite order does not. The approximating capability of the polynomial kernel with order p is limited to any polynomial function with its degree less than or equal to p. Unless it is clear from the problem domain that the target function is a polynomial function or can be well approximated by a polynomial function, the Gaussian kernel is usually a default choice. The Gaussian kernel has the *universal approximating capability*, is numerically stable, and usually gives reasonable results.

The *kernel bandwidth* (also known as kernel size, smoothing parameter) in the Gaussian kernel is an important parameter to be specified. In Chapter 1, we define the Gaussian kernel as

$$\kappa(\mathbf{u}, \mathbf{u}') = \exp(-a||\mathbf{u} - \mathbf{u}'||^2) \tag{2–19}$$

which is sometimes defined as

$$\kappa(\mathbf{u}, \mathbf{u}') = \exp(-\frac{||\mathbf{u} - \mathbf{u}'||^2}{2h^2}) \tag{2–20}$$

where h is the kernel bandwidth. If the Gaussian kernel is defined as in (2–19), the kernel bandwidth is $h = 1/\sqrt{2a}$. And a is simply called the kernel parameter. Available methods to select suitable kernel bandwidth include cross-validation, nearest neighbors, penalizing functions and plug-in methods [Härdle, 1992]. From the viewpoint of functional analysis, the kernel size helps define the inner product, i.e. the metric of similarity in RKHS. Similarity is the basis of inference. Therefore, the same input data can be mapped to vastly different functionals depending upon the kernel bandwidth selected. And very different filter outputs will be created if the kernel bandwidth is varied on the same data with the same kernel. If the kernel size is too large, all the data would look similar in the RKHS (with inner products all close to 1) and the system reduces to linear regression. If the kernel size is too small, all the data would look distinct (with inner products all close to 0) and the system is unable to do inference on unseen samples that fall between the training points. Since the kernel size is a free parameter and we are interested in an adaptive framework, potentially it may be adapted during operation as any of the other parameters. The resource allocating network is such an example and other relevant work can be found in the literature of locally adaptive kernel regression estimation [Herrmann, 1997]. In nonparametric regression, the kernel size is usually framed as the compromise between mean and variance of the estimator, which is very appropriate to help us find experimental procedures to estimate its optimal value from the data. Our experience tells that *cross-validation* on a small subset of data is usually adequate to select an appropriate kernel bandwidth and it is very straightforward. We provide a brief introduction on cross-validation below, and more details can be found in [Wahba, 1990].

If data are abundant and a validation set is affordable, the cross-validation cost function is defined as

$$\mathrm{CV}(h) = N_{\mathrm{CV}}^{-1} \sum_{j=1}^{N_{\mathrm{CV}}} [y_j - f_h(\mathbf{x}_j)]^2 \tag{2–21}$$

where h is the parameter we need to choose, $\{(\mathbf{x}_j, y_j)\}_{j=1}^{N_{\mathrm{CV}}}$ is the validation set and f_h is the estimated function by using the training data and h. We are interested in the minimum of this curve across a range of h values. If training data are scarce, *k-fold cross-validation* can be used. First, the training data are randomly split into k disjoint, equally sized subsets. Then each subset is picked as a validation set and the training is done on the union of the remaining $k-1$ subsets. After this process is repeated by k times by using different subset as validation set, we get k systems with k cross-validation cost, denoted by $\mathrm{CV}_1(h)$, $\mathrm{CV}_2(h)$,...,$\mathrm{CV}_k(h)$. Therefore the overall cost function of the k-fold cross-validation is

$$\mathrm{kCV}(h) = k^{-1} \sum_{j=1}^{k} \mathrm{CV}_j(h) \tag{2–22}$$

An extreme case of k-fold cross-validation is *leave-one-out cross-validation* where k equals the number of training data. The cost function of the leave-one-out cross-validation can be simply expressed as

$$\mathrm{LOOCV}(h) = N^{-1} \sum_{j=1}^{N} [y_j - f_{h,j}(\mathbf{x}_j)]^2 \tag{2–23}$$

where $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ is the training set and $f_{h,j}$ is the estimated function by using the training data excluding only the jth pair $\{(\mathbf{x}_j, y_j)\}$.

Searching for the best value by cross-validation is quite simple but can be tedious. It would be nice if we have a rough guess to start with. If $\mathbf{x}$ is one-dimensional, then Silverman's rule is often helpful:

$$h_s = 1.06 \min\{\sigma, R/1.34\} N^{-1/5} \tag{2–24}$$

where σ is the standard deviation of $\mathbf{x}$ and R is the interquartile of $\mathbf{x}$. The range $[h_s/10, 10h_s]$ is a good start for cross-validation. If $\mathbf{x}$ is multi-dimensional, it gets more complicated. In many nonlinear signal processing applications, $\mathbf{x}$ is constructed by time-embedding a one-dimensional time series. Then we can roughly estimate that the optimal parameter is somewhere in the interval

$$[1.06 \min\{\sigma, R/1.34\} N^{-1/5}, 1.06 \min\{\sigma, R/1.34\} N^{-1/(5L)}]$$

where σ is the standard deviation of the time series, R is the interquartile of the time series and L is the time-embedding dimension. This is understandable since higher dimensionality requires far more data to cover the data space. We have to emphasize that choosing the parameter requires experience and experiments.

2.4 Step-Size Parameter

After choosing the kernel and its free parameter, the next thing is to find a suitable step-size parameter. Since KLMS is the LMS algorithm in RKHS, the role of the step-size parameter remains in principle the same and the results from the adaptive filtering literature can be utilized. In particular, the step-size parameter is the compromise between convergence time and misadjustment (i.e. increasing the step-size parameter decreases convergence time but increases misadjustment). Moreover, the step-size parameter is upper bounded by the reciprocal of the largest eigenvalue of the transformed data autocorrelation matrix. Denoting the transformed data matrix $\mathbf{\Phi} = [\boldsymbol{\varphi}(1), \boldsymbol{\varphi}(2), ..., \boldsymbol{\varphi}(N)]$, $\mathbf{R}_{\boldsymbol{\varphi}}$ its autocorrelation matrix, and $\mathbf{G}_{\boldsymbol{\varphi}}$ its Gram matrix, we have

$$\begin{aligned} \mathbf{R}_{\boldsymbol{\varphi}} &= \frac{1}{N} \mathbf{\Phi} \mathbf{\Phi}^T \\ \mathbf{G}_{\boldsymbol{\varphi}} &= \mathbf{\Phi}^T \mathbf{\Phi} \end{aligned} \tag{2–25}$$

$\mathbf{G}_{\boldsymbol{\varphi}}$ is an $N \times N$ matrix with $\kappa(\mathbf{u}(i), \mathbf{u}(j))$ as its (i, j)-th component.

The step-size parameter is required to satisfy the following condition for the algorithm to stay stable [Haykin, 2002]

$$\eta < \frac{1}{\varsigma_{max}} \tag{2–26}$$

where ς_{max} is the largest eigenvalue of $\mathbf{R}_{\boldsymbol{\varphi}}$. The dimensionality of $\mathbf{R}_{\boldsymbol{\varphi}}$ could be very high and it is usually unfeasible to compute it directly. Fortunately, its eigenvalues can be computed from $\mathbf{G}_{\boldsymbol{\varphi}}$ [Golub and Loan, 1996]. More specifically, if $\mathbf{R}_{\boldsymbol{\varphi}}$ has r non-zero eigenvalues $\{\varsigma_j\}_{j=1}^{r}$, then $\mathbf{G}_{\boldsymbol{\varphi}}$ also has r non-zero eigenvalues, which are $\{N\varsigma_j\}_{j=1}^{r}$. Because $\mathbf{R}_{\boldsymbol{\varphi}}$ and $\mathbf{G}_{\boldsymbol{\varphi}}$ are both positive semi-definite, all the non-zero eigenvalues are positive. It is also known that the trace of a matrix equals the summation of all its eigenvalues. Using these facts, we have

$$\varsigma_{max} < \mathrm{tr}[\mathbf{R}_{\boldsymbol{\varphi}}] = \mathrm{tr}[\mathbf{G}_{\boldsymbol{\varphi}}]/N.$$

Therefore, a conservative upper bound for the step-size parameter is

$$\eta < \frac{N}{\mathrm{tr}[\mathbf{G}_{\boldsymbol{\varphi}}]} = \frac{N}{\sum_{j=1}^{N} \kappa(\mathbf{u}(j), \mathbf{u}(j))} \tag{2–27}$$

For shift-invariant kernels, i.e., $\kappa(\mathbf{u}(j), \mathbf{u}(j)) = g_0$, the upper bound becomes $1/g_0$, which is data-independent. We find this upper bound is quite handy in practice and use it as a default value.

Other properties of the LMS algorithm can also be easily used for KLMS. For example, the misadjustment of KLMS can be estimated as

$$\mathcal{M} = \frac{\eta}{2}\mathrm{tr}[\mathbf{R}_{\boldsymbol{\varphi}}] = \frac{\eta}{2N}\mathrm{tr}[\mathbf{G}_{\boldsymbol{\varphi}}] \tag{2–28}$$

In the case of shift-invariant kernels, the misadjustment of KLMS equals $\eta g_0/2$ which is also data-independent and is simply proportional to the step-size parameter.

2.5 Novelty Criterion

In a stationary environment, the learning system will eventually converge after processing sufficient examples and stop training afterwards. As we see in the formulation

of KLMS, the size of the network increases linearly with the number of training data, which poses a challenge for applying KLMS in nonstationary signal processing. A fundamental question is if it is necessary to memorize all the past inputs. By removing redundant data, it is possible to keep a minimal set of centers that covers the area where inputs will likely appear (Imagine that a kernel is a sphere in the input space ($\mathbb{R}^L$) with the kernel bandwidth as the radius.). On the other hand, a sparse model (a network with as few kernels as possible) is desirable because it reduces the complexity in terms of computation and memory, and it usually gives better generalization ability (Occam's Razor). There are many approaches to sparsification of kernel-based solutions but most of them are off-line methods. We focus the discussion here on *online sparsification* or *sequential sparsification*. A simple way to check if the newly arrived datum is informative enough is the *novelty criterion* (NC) proposed by Platt [1991]. Richard et al. [2009] also studied a similar method called *coherence criterion* with many mathematical properties. Engel et al. [2004] introduced another way to tackle this problem with the idea of *approximate linear dependency* test (ALD), which is close to the work of Csato and Opper [2002]. This has also been explored specifically for KLMS in [Pokharel et al., 2009]. Sequential sparsification is also being studied in computational learning theory, such as the kernel perceptron with a fixed budget [Dekel et al., 2006]. We will propose yet another criterion to address this issue in Chapter 6 and unify NC and ALD in a rigorous information theoretic framework. In this chapter, we focus on Platt's novelty criterion.

Online sparsification is usually obtained by *construction* in a sense that it starts from an empty set and gradually adds samples into a center set called *dictionary* according to some criterion. Suppose the present dictionary is $\mathcal{C}(i) = \{\mathbf{c}_j\}_{j=1}^{m_i}$ where $\mathbf{c}_j$ is the jth center and m_i is the cardinality. When a new data pair $\{\mathbf{u}(i+1), d(i+1)\}$ is presented, a decision is made immediately whether $\mathbf{u}(i+1)$ should be added into the dictionary as a new center. In novelty criterion, it first calculates the distance of $\mathbf{u}(i+1)$ to the present dictionary $\text{dis}_1 = \min_{\mathbf{c}_j \in \mathcal{C}(i)} ||\mathbf{u}(i+1) - \mathbf{c}_j||$. If it is smaller than some preset threshold, say δ_1, $\mathbf{u}(i+1)$

will not be added into the dictionary. Otherwise, the algorithm computes the prediction error $e(i+1)$. Only if the prediction error is larger than another preset threshold, say δ_2, $\mathbf{u}(i+1)$ will be accepted as a new center. Here are some heuristics on how to set the parameters for NC. Initially, the kernel filter is designed without the novelty criterion such that we can focus on step-size parameter and kernel size selection. After picking the kernel size and having an estimate of steady-state mean square error (MSE), the second step becomes straightforward. A reasonable δ_1 is around one-tenth of the kernel bandwidth $\sqrt{1/2a}$. Increasing δ_1 will decrease the network size but the performance may degrade. A reasonable default value for δ_2 is the square root of the steady-state MSE. Increasing δ_2 will decrease the network size but the performance may degrade. Cross-validation also can be used to select appropriate thresholds.

If the input domain $\mathbb{U}$ is a compact set, with the aid of the novelty criterion, the cardinality of the dictionary is always finite and upper bounded. This statement is not hard to prove using the finite covering theorem of the compact set and the fact that elements in the dictionary are δ-separable. Here is a brief outline of the proof:

Suppose spheres with diameter δ are used to cover $\mathbb{U}$ and the optimal covering number is N_c. Then, because any two centers in the dictionary cannot be in the same sphere, the total number of the centers will be no greater than N_c regardless of the distribution and temporal structure of $\mathbf{u}$. Of course, this is a worst-case upper bound. In the case of finite training data, the network size will be finite anyway. This is true in applications like channel equalization, where the training sequence is part of each transmission frame. In a stationary environment, the network converges quickly and the threshold on prediction errors plays its part to constrain the network size. We will validate this claim in the simulation section. In a non-stationary environment, there are two scenarios. In the first scenario, the input domain does not change and only the input-output mapping changes. After the network grows to a point that the input domain is sufficiently covered, simple LMS can be used to just modify the coefficients to track

the nonstationarity. In the second scenario, input domain changes as well. In this case, pruning methods should be used to constrain the network size[3]. Another alternative approach is to solve the problem in the primal space directly by using the low-rank approximation methods[4]. It should be pointed out that the scalability issue is at the core of the kernel methods and all the kernel methods need to deal with it in one way or the other. Indeed, the sequential nature of KLMS enables active learning on huge data sets.

2.6 Self-Regularization Property of KLMS

The KLMS algorithm is derived in a high-dimensional feature space, using stochastic gradient to solve a least-squares problem. If we study any other kernel machine algorithms, we are alerted for the central role of regularization to obtain solutions that generalize appropriately. Therefore, it is not surprising that all the attempts to derive kernel adaptive filters mentioned in Chapter 1, utilized a regularized cost function. The surprising fact is that we were able to prove mathematically that KLMS does not need explicit regularization since it is well posed in the sense of Hadamard [Liu et al., 2008]. These results are summarized below.

2.6.1 Solution Norm Bound

From the viewpoints of the regularization and optimization theories [Hoerl and Kennard, 1970], the concepts of regularization, stability and solution norm constraint are tightly related. The significance of an upper bound for the solution norm is also studied by Poggio and Smale [2003].

Assume the training data $\{\mathbf{u}(i), d(i)\}_{i=1}^{N}$ satisfy a multiple linear regression model in the RKHS:

$$d(i) = \boldsymbol{\varphi}(i)^T \boldsymbol{\omega}^o + v(i) \tag{2–29}$$

where $\boldsymbol{\omega}^o$ is the underlying model and $v(i)$ is the modeling uncertainty. Then by the H^∞ robustness theorem [Haykin, 2002]: for any unknown vector $\boldsymbol{\omega}^o$ and finite energy noise

sequence $v(i)$ without further statistical assumptions, the following inequality holds

$$\frac{\sum_{j=1}^{i}|\hat{s}(j)-s(j)|^2}{\eta^{-1}||\boldsymbol{\omega}^o||^2+\sum_{j=1}^{i-1}|v(j)|^2}<1 \tag{2–30}$$

if and only if the matrices $\{\eta^{-1}I-\boldsymbol{\varphi}(i)\boldsymbol{\varphi}(i)^T\}$ are positive-definite for all $i\leq N$. In the inequality, $s(j)=(\boldsymbol{\omega}^o)^T\boldsymbol{\varphi}(j)$ and $\hat{s}(j)=\boldsymbol{\omega}(j-1)^T\boldsymbol{\varphi}(j)$, where $\boldsymbol{\omega}(j-1)$ is calculated by the KLMS recursion (2–13). This result is used to prove the following theorem.

Theorem 2.1. *Under the H^∞ stability condition, the prediction error satisfies the following inequality:*

$$||\mathbf{e}||^2<\eta^{-1}||\boldsymbol{\omega}^o||^2+2||\mathbf{v}||^2 \tag{2–31}$$

where $\mathbf{e}=[e(1),...,e(N)]^T$ *and* $\mathbf{v}=[v(1),...,v(N)]^T$.

Proof. First we have

$$e(i)-v(i)=s(i)-\hat{s}(i)$$

Substituting it into (2–30), we have

$$\frac{\sum_{j=1}^{i}|e(j)-v(j)|^2}{\eta^{-1}||\boldsymbol{\omega}^o||^2+\sum_{j=1}^{i-1}|v(j)|^2}<1$$

or equivalently,

$$\sum_{j=1}^{i}|e(j)-v(j)|^2<\eta^{-1}||\boldsymbol{\omega}^o||^2+\sum_{j=1}^{i-1}|v(j)|^2$$

By the triangle inequality

$$\begin{aligned}\sum_{j=1}^{i}|e(j)|^2&\leq\sum_{j=1}^{i}|e(j)-v(j)|^2+\sum_{j=1}^{i}|v(j)|^2\\&<\eta^{-1}||\boldsymbol{\omega}^o||^2+\sum_{j=1}^{i-1}|v(j)|^2+\sum_{j=1}^{i}|v(j)|^2\end{aligned} \tag{2–32}$$

which is valid for all $i\leq N$. In terms of vector norm,

$$||\mathbf{e}||^2<\eta^{-1}||\boldsymbol{\omega}^o||^2+2||\mathbf{v}||^2 \tag{2–33}$$

□

Theorem 2.2. *Under the H^∞ stability condition, $\boldsymbol{\omega}(N)$ is upper-bounded:*

$$||\boldsymbol{\omega}(N)|| < \sqrt{N\varsigma_1\eta(||\boldsymbol{\omega}^o||^2 + 2\eta||\mathbf{v}||^2)} \tag{2–34}$$

where ς_1 is the largest eigenvalue of $\mathbf{R}_{\boldsymbol{\varphi}}$.

Proof.

$$\begin{aligned}
||\boldsymbol{\omega}(N)||^2 &= ||\eta \sum_{i=1}^{N} e(i)\boldsymbol{\varphi}(i)||^2 \\
&= \eta^2 \mathbf{e}^T \mathbf{G}_{\boldsymbol{\varphi}} \mathbf{e} \\
&= \eta^2 N \mathbf{e}^T \mathbf{Q} \text{diag}\{\varsigma_1, \varsigma_2, ..., \varsigma_N\} \mathbf{Q}^T \mathbf{e} \\
&\leq \eta^2 N \mathbf{e}^T \mathbf{Q} \text{diag}\{\varsigma_1, \varsigma_1, ..., \varsigma_1\} \mathbf{Q}^T \mathbf{e} \\
&= \eta^2 N \varsigma_1 ||\mathbf{Q}^T \mathbf{e}||^2 \\
&= \eta^2 N \varsigma_1 ||\mathbf{e}||^2
\end{aligned}$$

where

$$\mathbf{G}_{\boldsymbol{\varphi}} = \mathbf{Q} \text{diag}\{N\varsigma_1, N\varsigma_2, ..., N\varsigma_N\} \mathbf{Q}^T$$

is the standard eigenvalue decomposition. $\mathbf{Q}$ is an orthogonal matrix. Then by Theorem 2.1, we have the result directly. □

This result effectively shows that the norm of the KLMS solution is constrained. It also directly implies the compactness of the hypothesis space and thus ensures algorithmic stability.

2.6.2 Singular Value Analysis

Although the result in Theorem 2.2 is conclusive, several useful insights have been neglected. Meanwhile, a singular value analysis is able to clearly show that the self-regularization property of KLMS is due to its different convergence speeds along different eigen-directions.

Let the singular value decomposition (SVD) of $\mathbf{\Phi}$ be

$$\mathbf{\Phi} = \mathbf{P} \begin{bmatrix} \mathbf{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{Q}^T \tag{2–35}$$

where $\mathbf{P}$, $\mathbf{Q}$ are orthogonal matrices and $\mathbf{S} = \text{diag}(s_1, ..., s_r)$ with s_i the singular values and r the rank of $\mathbf{\Phi}$. It is assumed that $s_1 \geq ... \geq s_r > 0$ without loss of generality. Then, we have

$$\mathbf{R}_{\boldsymbol{\varphi}} = \mathbf{P} \begin{bmatrix} \mathbf{S}^2/N & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{P}^T \tag{2–36}$$

$$\mathbf{G}_{\boldsymbol{\varphi}} = \mathbf{Q} \begin{bmatrix} \mathbf{S}^2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{Q}^T \tag{2–37}$$

It is clear that $\varsigma_j = s_j^2/N$.

The well-known pseudo-inverse solution to estimate $\boldsymbol{\omega}^o$ in (2–29) obtained by minimizing

$$J(\boldsymbol{\omega}) = ||\mathbf{d} - \mathbf{\Phi}^T \boldsymbol{\omega}||^2 \tag{2–38}$$

is

$$\boldsymbol{\omega}_{PI} = \mathbf{P}\text{diag}(s_1^{-1}, ..., s_r^{-1}, 0, ..., 0)\mathbf{Q}^T\mathbf{d} \tag{2–39}$$

The least-squares solution (even with the pseudo-inverse, just think of a very small s_r) can be ill-posed due to the nature of the problem, small data size, or severe noise. The Tikhonov regularization [Tikhonov and Arsenin, 1977] is widely used to address this issue. A regularization term is introduced in the least squares cost function which penalizes the solution norm:

$$J(\boldsymbol{\omega}) = ||\mathbf{d} - \mathbf{\Phi}^T \boldsymbol{\omega}||^2 + \lambda ||\boldsymbol{\omega}||^2 \tag{2–40}$$

Solving this minimization problem yields the Tikhonov regularization solution

$$\boldsymbol{\omega}_{TR} = \mathbf{P}\text{diag}(\frac{s_1}{s_1^2 + \lambda}, ..., \frac{s_r}{s_r^2 + \lambda}, 0, ..., 0)\mathbf{Q}^T\mathbf{d} \tag{2–41}$$

Comparing (2–41) with (2–39), we see that the Tikhonov regularization modifies the diagonal terms through the following regularization function (reg-function):

$$H_{TR}(x) = \frac{x^2}{x^2 + \lambda} \tag{2–42}$$

If s_r is very small, the pseudo-inverse solution becomes problematic as the solution approaches infinity. However, for the Tikhonov regularization, $H_{TR}(s_r)s_r^{-1} \rightarrow 0$ if s_r is small and $H_{TR}(s_r)s_r^{-1} \rightarrow s_r^{-1}$ if s_r is large. In this sense, the Tikhonov regularization *smoothly filters out* the minor components that correspond to small singular values (relative to λ). Attenuating the minor components is important to get a smaller norm solution or in other words, a more stable solution. With this understanding, the so-called truncated pseudo-inverse regularization [Golub and Loan, 1996] is nothing but using the following hard cut-off reg-function:

$$H_{PCA}(x) = \begin{cases} 1 & if\ x > t \\ 0 & if\ x \leq t \end{cases} \tag{2–43}$$

where t is the cut-off threshold. If $s_k > t \geq s_{k+1}$ (usually $k \ll r$), the solution becomes

$$\boldsymbol{\omega}_{PCA} = \mathbf{P}\text{diag}(s_1^{-1}, ..., s_k^{-1}, 0, ..., 0)\mathbf{Q}^T\mathbf{d} \tag{2–44}$$

This method is equivalent to applying principal components analysis technique (PCA) to the data and using the first k principal components to represent the original data. Under reasonable signal-noise-ratio, the small singular value components are purely associated with the noise. Discarding these spurious features can effectively prevent over-learning.

With the above discussion, we are ready to show why KLMS possesses a self-regularization property. First define the natural modes of the weight error in terms of the eigen-vectors of $\mathbf{R}_{\boldsymbol{\varphi}}$:

$$\boldsymbol{\omega}(n) - \boldsymbol{\omega}^o = \sum_{j=1}^{M} \varepsilon_j(n)\mathbf{P}_j$$

where $\mathbf{P}_j$ is the jth column of $\mathbf{P}$, M is the dimensionality of $\mathbf{R}_{\boldsymbol{\varphi}}$, $\varepsilon_j(n)$ denotes the distance between $\boldsymbol{\omega}(n)$ and $\boldsymbol{\omega}^o$ in the jth eigen-vector direction. It has been shown that [Haykin, 2002]

$$\begin{aligned}\mathbf{E}[\varepsilon_j(n)] &= (1-\eta\varsigma_j)^n\varepsilon_j(0)\\ \mathbf{E}[|\varepsilon_j(n)|^2] &= \frac{\eta J_{min}}{2-\eta\varsigma_j} + (1-\eta\varsigma_j)^{2n}(|\varepsilon_j(0)|^2 - \frac{\eta J_{min}}{2-\eta\varsigma_j})\end{aligned} \tag{2–45}$$

where J_{min} is the irreducible error power. Therefore,

$$\mathbf{E}[\boldsymbol{\omega}(n)] = \boldsymbol{\omega}^o + \sum_{j=1}^{M}(1-\eta\varsigma_j)^n\varepsilon_j(0)\mathbf{P}_j \tag{2–46}$$

Furthermore with $\boldsymbol{\omega}^o = \sum_{j=1}^{M}\boldsymbol{\omega}_j^o\mathbf{P}_j$, $\boldsymbol{\omega}(0) = 0$ and $\varepsilon_j(0) = -\boldsymbol{\omega}_j^o$, we have

$$\begin{aligned}\mathbf{E}[\boldsymbol{\omega}(n)] &= \sum_{j=1}^{M}\boldsymbol{\omega}_j^o\mathbf{P}_j - \sum_{j=1}^{M}(1-\eta\varsigma_j)^n\boldsymbol{\omega}_j^o\mathbf{P}_j\\ &= \sum_{j=1}^{M}[1-(1-\eta\varsigma_j)^n]\boldsymbol{\omega}_j^o\mathbf{P}_j\end{aligned} \tag{2–47}$$

It is clear that the norm of the expected weight is upper bounded by

$$\begin{aligned}||\mathbf{E}[\boldsymbol{\omega}(n)]||^2 &= \sum_{j=1}^{M}[1-(1-\eta\varsigma_j)^n]^2(\boldsymbol{\omega}_j^o)^2\\ &= \sum_{j=1}^{M}[1-(1-\eta\varsigma_j)^n]^2(\boldsymbol{\omega}_j^o)^2\\ &\leq \sum_{j=1}^{M}(\boldsymbol{\omega}_j^o)^2 = ||\boldsymbol{\omega}^o||^2\end{aligned} \tag{2–48}$$

assuming $\eta \leq 1/\varsigma_j$. In the worst case, by replacing the optimal weight with the pseudo inverse solution, we have

$$\begin{aligned}\mathbf{E}[\boldsymbol{\omega}(n)] &= \mathbf{P}\text{diag}([1-(1-\eta\varsigma_1)^n]s_1^{-1},...,[1-(1-\eta\varsigma_r)^n]s_r^{-1},0,...,0)\mathbf{Q}^T\mathbf{d}\\ &= \mathbf{P}\text{diag}([1-(1-\eta s_1^2/N)^n]s_1^{-1},...,[1-(1-\eta s_r^2/N)^n]s_r^{-1},0,...,0)\mathbf{Q}^T\mathbf{d}\end{aligned} \tag{2–49}$$

which means the reg-function for KLMS (in the mean sense) stopped at iteration N is

$$H_{KLMS}(x) = 1 - (1 - \eta x^2/N)^N \tag{2–50}$$

And the following theorem tells why KLMS takes care of the small singular values.

Theorem 2.3. $\lim_{x\to 0} H_{KLMS}(x)\, x^{-1} = 0$

Proof.

$$\begin{aligned} H_{KLMS}(x)\, x^{-1} &= \frac{1}{x}[1 - (1 - \eta x^2/N)][1 + (1 - \eta x^2/N) + ... + (1 - \eta x^2/N)^{N-1}] \\ &= \eta x/N[1 + (1 - \eta x^2/N) + ... + (1 - \eta x^2/N)^{N-1}] \end{aligned}$$

Therefore, it is a polynomial in x and the conclusion follows directly. □

A comparison of three regularization methods is illustrated in Figure 2-2. In the reg-function of Tikhonov regularization, the regularization parameter is chosen as 1. For the reg-function of PCA, $t = 0.5$. For the reg-function of KLMS, $\eta = 0.1$ and $N = 500$. Furthermore in Figures 2-3 and 2-4, we show the effect of the step-size parameter and data size on the regularization function of KLMS. The figures show clearly that the step-size parameter affects the regularization significantly while the training data size does not as long as it is sufficiently large. This fact is not surprising if we recall the basic mathematical formula

$$\lim_{n\to\infty}(1 + \frac{1}{n})^n = e.$$

The conclusion from this discussion is that the step-size parameter in KLMS plays a similar role as the regularization parameter in explicitly regularized cost functions. Therefore, there is no need for explicit regularization which simplifies the algorithm implementation tremendously:

The step-size parameter in KLMS is not only a compromise between adaptation speed and misadjustment, it also controls the generalization ability of the algorithm. Increasing the step-size parameter leads to a danger of overfitting, while a smaller step size helps generalization.

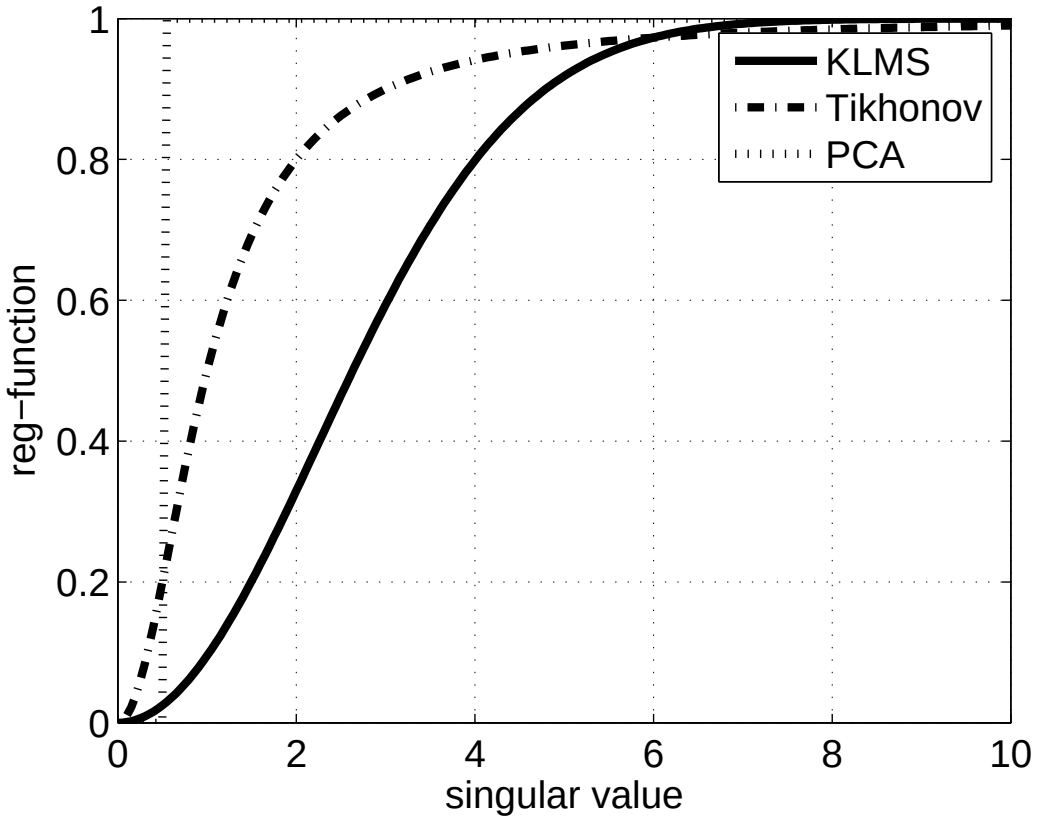

Figure 2-2. Comparison of three regularization approaches: KLMS, Tikhonov regularization and PCA

2.6.3 A Unit Lower Triangular Linear System

Another interesting observation can be made about KLMS is that it can be formulated as solving a unit lower triangular linear system.

Theorem 2.4. *The KLMS prediction errors* $e(1)$, $e(2)$,..., $e(i)$ *are linearly related to the desired samples* $d(1)$, $d(2)$,..., $d(i)$ *through a unit lower triangular matrix.*

Proof. By (2–16),

$$e(j) = d(j) - \eta \sum_{k=1}^{j-1} e(k)\kappa(\mathbf{u}(k), \mathbf{u}(i))$$

so

$$d(j) = e(j) + \eta \sum_{k=1}^{j-1} e(k)\kappa(\mathbf{u}(k), \mathbf{u}(i))$$

for $j = 1, ..., i$. Writing them into matrix form yields

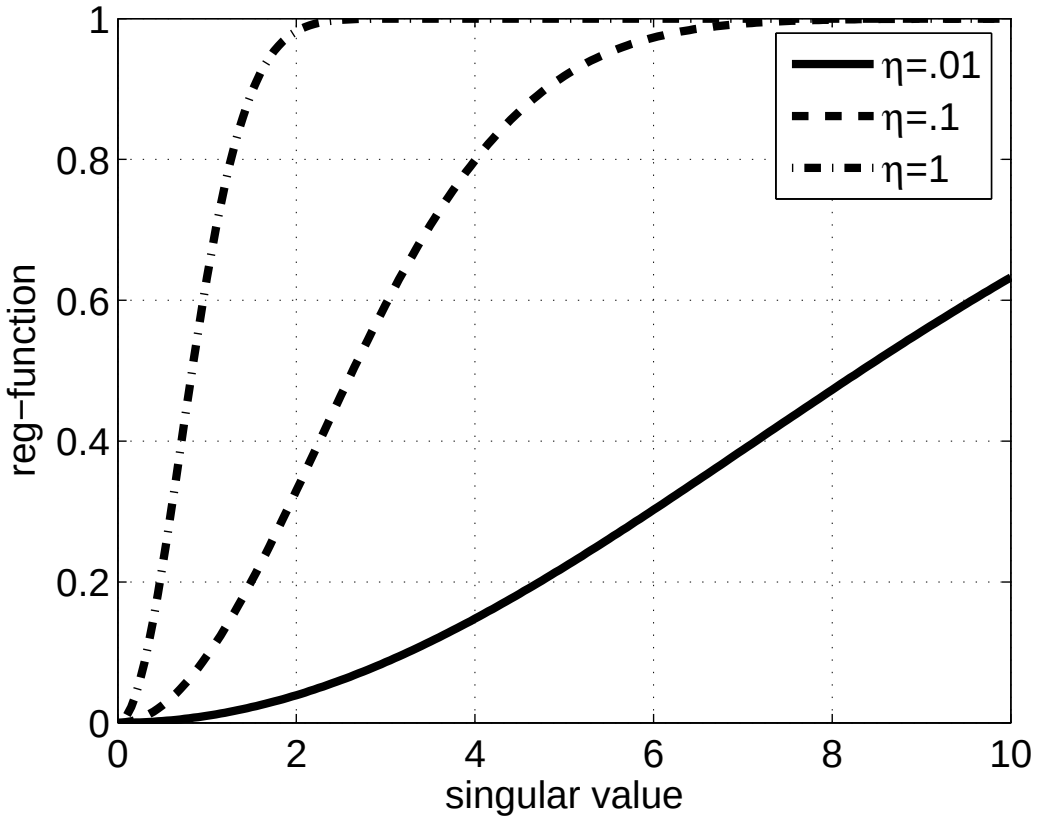

Figure 2-3. Effect of step-size parameter on the reg-function of KLMS ($N = 500$)

$$\begin{pmatrix} 1 & 0 & 0 & ... & 0 \\ \eta\kappa_{1,2} & 1 & 0 & ... & 0 \\ ... & ... & ... & ... & ... \\ \eta\kappa_{1,i} & \eta\kappa_{2,i} & \eta\kappa_{3,i} & ... & 1 \end{pmatrix}_{i\times i} \begin{pmatrix} e(1) \\ e(2) \\ ... \\ e(i) \end{pmatrix}_{i\times 1} = \begin{pmatrix} d(1) \\ d(2) \\ ... \\ d(i) \end{pmatrix}_{i\times 1} \tag{2–51}$$

where $\kappa_{i,j} = \kappa(\mathbf{u}(i), \mathbf{u}(j))$ for simplicity. This completes the proof. □

This result is very interesting. It tells us that instead of solving a large-scale dense linear system, we can find an "approximate" solution by solving a unit lower triangular linear system. Numerically, we know that inverting a unit lower triangular matrix is quite stable. When the step-size parameter is very small, the matrix is close to the identity matrix and its stability is guaranteed.

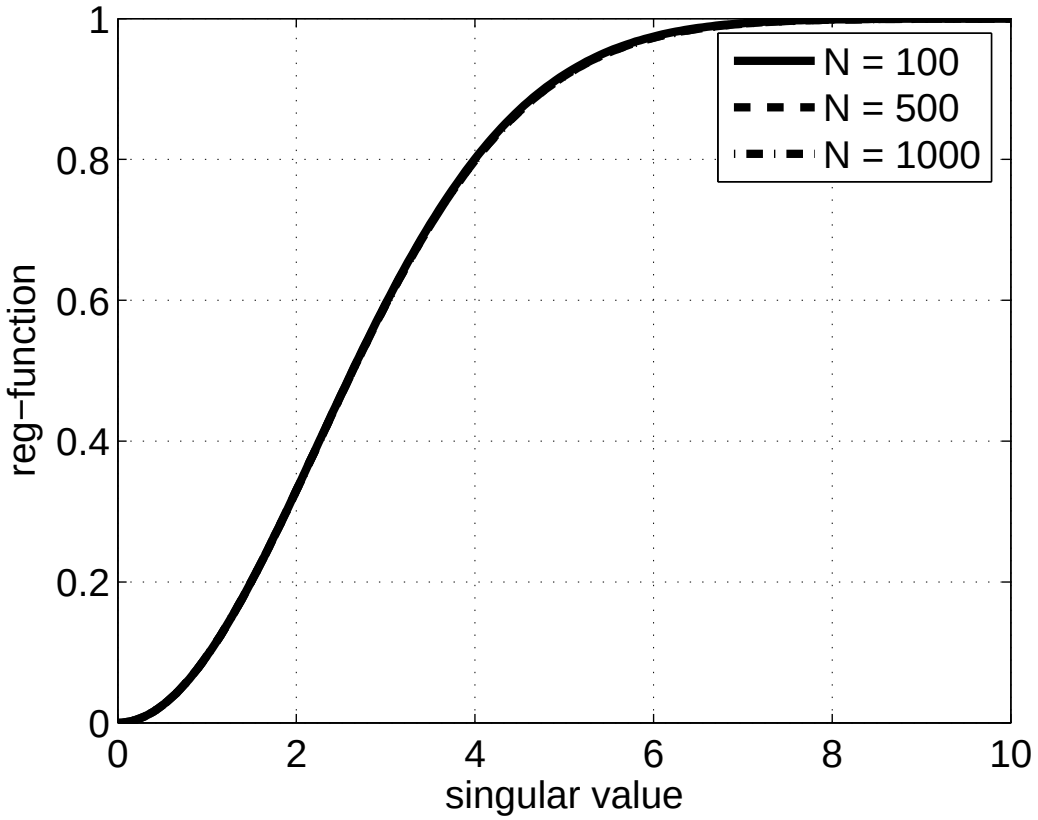

Figure 2-4. Effect of training data size on the reg-function of KLMS ($\eta = 0.1$). Three lines overlap which means the data size does not affect the shape of the reg-function of KLMS.

2.7 Leaky Kernel Least Mean Square Algorithm

A similar algorithm called NORMA was derived in [Kivinen et al., 2004], but from a vastly different viewpoint. The authors of the paper just cited differentiated the following regularized functional directly to get the stochastic gradient in the function space

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

with λ as the regularization parameter.

While the derivation involves advanced mathematics, the results are actually equivalent to the following update rule

$$f_i = (1 - \eta\lambda) f_{i-1} + \eta e(i) \kappa(\mathbf{u}(i), \cdot) \tag{2–52}$$

Comparing (2–52) with KLMS (2–17), it has a scaling factor $(1 - \eta\lambda)$ on the previous estimate and it imposes a forgetting mechanism so that the training data in the far past are scaled down exponentially. Therefore, by neglecting the units with very small coefficients, the number of actual active units is finite.

The regularization introduces a bias in the solution as is well known in leaky LMS [Sayed, 2003]. Pokharel et al. [2007] reported that even a very small regularization parameter degrades its performance comparing with KLMS.

2.8 Normalized Kernel Least Mean Square Algorithm

The *normalized least-mean-square algorithm* (NLMS) usually exhibits better performance than LMS in many practical applications. The weight update equation for NLMS is [Haykin, 2002]

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \frac{\eta}{\varepsilon + ||\mathbf{u}(i)||^2} e(i)\mathbf{u}(i)$$

where $\varepsilon + ||\mathbf{u}(i)||^2$ is the normalizing term and ε is a small positive number introduced to prevent divide-by-zero exception. The normalized LMS algorithm is summarized in Algorithm 3.

Algorithm 3 The Normalized Least Mean Square Algorithm

Initialization
$\mathbf{w}(0) = 0$, choose η, ε
Computation
while $\{\mathbf{u}(i), d(i)\}$ available **do**
  $e(i) = d(i) - \mathbf{w}^T(i-1)\mathbf{u}(i)$
  $\mathbf{w}(i) = \mathbf{w}(i-1) + \frac{\eta}{\varepsilon + ||\mathbf{u}(i)||^2} e(i)\mathbf{u}(i)$
end while

It is straightforward to derive the *normalized kernel least mean square algorithm* based on the above discussion. The weight update equation for normalized KLMS is

$$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \frac{\eta}{\varepsilon + ||\boldsymbol{\varphi}(i)||^2} e(i)\boldsymbol{\varphi}(i)$$

And by using the definition of the norm in the feature space, we have

$$||\boldsymbol{\varphi}(i)||^2 =< \boldsymbol{\varphi}(i), \boldsymbol{\varphi}(i) >= \kappa(\mathbf{u}(i), \mathbf{u}(i))$$

If the kernel is shift-invariant, i.e., $\kappa(\mathbf{u}(j), \mathbf{u}(j)) = g_0$, KLMS is automatically normalized.

2.9 Kernel ADALINE

Kernel ADALINE [T.-T.Frieb and Harrison, 1999] is a gradient descent method solving an *unregularized* least squares cost in RKHS. Suppose the number of training data is N and we are solving the following unregularized least squares cost

$$\min_{\boldsymbol{\omega}} J(\boldsymbol{\omega}) = ||\mathbf{d} - \boldsymbol{\Phi}^T\boldsymbol{\omega}||^2 \tag{2–53}$$

where

$$\boldsymbol{\Phi} = [\boldsymbol{\varphi}(1), ..., \boldsymbol{\varphi}(N)]$$
$$\mathbf{d} = [d(1), d(2), ..., d(N)]^T$$

The gradient of the cost function (2–53) is

$$\nabla J(\boldsymbol{\omega}) = -2\boldsymbol{\Phi}(\mathbf{d} - \boldsymbol{\Phi}^T\boldsymbol{\omega}) \tag{2–54}$$

Therefore, the gradient descent method is

$$\begin{aligned} \boldsymbol{\omega}(i) &= \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(\mathbf{d} - \boldsymbol{\Phi}^T\boldsymbol{\omega}(i-1))/N \\ &= \boldsymbol{\omega}(i-1) + \frac{\eta}{N}\sum_{j=1}^{N}[\boldsymbol{\varphi}(j)(d(j) - \boldsymbol{\varphi}(j)^T\boldsymbol{\omega}(i-1))] \end{aligned} \tag{2–55}$$

where $\boldsymbol{\omega}(i)$ denotes the estimate of the weight at iteration i. η is the step-size parameter. Comparing with (2–13), it is clear that KLMS is a stochastic gradient descent method whereas kernel ADALINE is a batch-mode gradient descent method.

With initial value $\boldsymbol{\omega}(0) = 0$, the weight estimate by (2–55) is a linear combination of the transformed data at any iteration, i.e.,

$$\boldsymbol{\omega}(i) = \boldsymbol{\Phi}\mathbf{a}(i) = \sum_{j=1}^{N} \mathbf{a}_j(i)\varphi(j), \quad \forall\, i \tag{2–56}$$

Notice that this result cannot be derived from the representer theorem because we do not have the explicit norm constraint in (2–53). Instead, we can use mathematical induction to prove the claim. Since $\boldsymbol{\omega}(0) = 0$, the claim is true for $i = 0$. Suppose (2–56) is true for $i - 1$. Therefore

$$\begin{aligned}
\mathbf{e}(i) &= \mathbf{d} - \boldsymbol{\Phi}^T\boldsymbol{\omega}(i-1) \\
&= \mathbf{d} - (\boldsymbol{\Phi}^T\boldsymbol{\Phi})\,\mathbf{a}(i-1) \\
&= \mathbf{d} - \mathbf{G}\,\mathbf{a}(i-1)
\end{aligned}$$

Then, by (2–55), we have

$$\begin{aligned}
\boldsymbol{\omega}(i) &= \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(\mathbf{d} - \boldsymbol{\Phi}^T\boldsymbol{\omega}(i-1))/N \\
&= \boldsymbol{\Phi}\mathbf{a}(i-1) + \eta\boldsymbol{\Phi}\mathbf{e}(i)/N \\
&= \boldsymbol{\Phi}(\mathbf{a}(i-1) + \eta\mathbf{e}(i)/N)
\end{aligned}$$

i.e.,

$$\mathbf{a}(i) = \mathbf{a}(i-1) + \eta\mathbf{e}(i)/N \tag{2–57}$$

This result is crucial in kernel methods, since $\boldsymbol{\omega}$ is in a high-dimensional space and we usually do not have access to it. By writing $\boldsymbol{\omega}$ as a linear combination of the training data, we actually solve a problem with dimensionality N. Furthermore, we can show that the gradient descent iteration of kernel ADALINE provides an inherent regularization similar to KLMS.

First, rewrite (2–55) as

$$\begin{aligned}
&\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(\mathbf{d} - \boldsymbol{\Phi}^T\boldsymbol{\omega}(i-1))/N \\
&= (\mathbf{I} - \eta\boldsymbol{\Phi}\boldsymbol{\Phi}^T/N)\boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}\mathbf{d}/N \\
&= (\mathbf{I} - \eta\mathbf{P}\begin{bmatrix} \mathbf{S}^2/N & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}\mathbf{P}^T)\boldsymbol{\omega}(i-1) + \eta\mathbf{P}\begin{bmatrix} \mathbf{S}/N & 0 \\ 0 & 0 \end{bmatrix}\mathbf{Q}^T\mathbf{d} \\
&= \mathbf{P}[(\mathbf{I} - \eta\begin{bmatrix} \mathbf{S}^2/N & 0 \\ 0 & 0 \end{bmatrix})(\mathbf{P}^T\boldsymbol{\omega}(i-1)) + \eta\begin{bmatrix} \mathbf{S}/N & 0 \\ 0 & 0 \end{bmatrix}\mathbf{Q}^T\mathbf{d}]
\end{aligned} \tag{2–58}$$

Here we use the result of (2–35). Denote $\mathbf{b}(i) = \mathbf{P}^T\boldsymbol{\omega}(i)$, which amounts to decomposing the weight vector along the column vectors of matrix P as

$$\boldsymbol{\omega}(i) = \sum_{j=1}^{M} \mathbf{b}_j(i)\mathbf{P}_j = \mathbf{P}\mathbf{b}(i)$$

where M is assumed the dimensionality of RKHS. Therefore, by (2–58), we have

$$\mathbf{b}(i) = (\mathbf{I} - \eta\begin{bmatrix} \mathbf{S}^2/N & 0 \\ 0 & 0 \end{bmatrix})(\mathbf{b}(i-1)) + \eta\begin{bmatrix} \mathbf{S}/N & 0 \\ 0 & 0 \end{bmatrix}\mathbf{Q}^T\mathbf{d} \tag{2–59}$$

or equivalently for each component

$$\mathbf{b}_j(i) = (1 - \eta s_j^2/N)\mathbf{b}_j(i-1) + \eta s_j\mathbf{Q}_j^T\mathbf{d}/N \tag{2–60}$$

for $1 \leq j \leq M$.

Observe that if $s_j = 0$, then

$$\mathbf{b}_j(i) = \mathbf{b}_j(i-1) = ... = \mathbf{b}_j(0)$$

If $s_j \neq 0$, we repeatedly use (2–60) for $i = 1, 2, ...$ and obtain

$$\begin{aligned}
&\mathbf{b}_j(i) = (1 - \eta s_j^2/N)^i\,\mathbf{b}_j(0) + (\eta s_j\mathbf{Q}_j^T\mathbf{d}/N)(\sum_{m=0}^{i-1}(1 - \eta s_j^2/N)^m) \\
&= (1 - \eta s_j^2/N)^i\,\mathbf{b}_j(0) + [1 - (1 - \eta s_j^2/N)^i](\mathbf{Q}_j^T\mathbf{d})/s_j
\end{aligned} \tag{2–61}$$

Notice that s_j^2/N is the eigenvalue of the correlation matrix which is asymptotically independent of N. The interesting observation is if proper early-stopping is used in the training, then the solution norm of the kernel ADALINE is upper bounded. For example, we start from $\boldsymbol{\omega}(0) = 0$ and the training stops after n steps. Therefore,

$$\mathbf{b}_j(n) = [1 - (1 - \eta s_j^2/N)^n](\mathbf{Q}_j^T \mathbf{d})/s_j \tag{2–62}$$

This equation shows that along different eigen-directions, the algorithm converges at vastly different speeds. If s_j is very small, $(1 - \eta s_j^2/N)$ is very close to 1, which leads to a very slow convergence. On the other hand, for large s_j, $(1 - \eta s_j^2/N)$ is close to 0 and the convergence is very fast.

Furthermore, for $\boldsymbol{\omega}(n) = \mathbf{P}\mathbf{b}(n)$, we have

$$\boldsymbol{\omega}_{KA,n} = \mathbf{P}\text{diag}([1 - (1 - \eta s_1^2/N)^n]s_1^{-1}, ..., [1 - (1 - \eta s_r^2/N)^n]s_r^{-1}, 0, ..., 0)\mathbf{Q}^T\mathbf{d} \tag{2–63}$$

It means the reg-function for the kernel ADALINE stopped at iteration n is

$$H_{KA,n}(x) = 1 - (1 - \eta x^2/N)^n$$

which is quite similar to (2–50) for KLMS except the exponent. In the following theorem, we explicitly establish an upper bound for the solution norm $||\boldsymbol{\omega}_{KA,n}||$.

Lemma 2.5. *Assume* $|1 - \eta x^2/N| < 1$ *and* $x \geq 0$.

$$|\frac{1 - (1 - \eta x^2/N)^n}{x}| \leq \sqrt{\frac{2\eta}{N}}n$$

Proof. Let $z = \sqrt{\frac{\eta}{N}}x$ and $H(z) = \frac{1-(1-z^2)^n}{z}$.

$$\begin{aligned}
|H(z)| &= \frac{1}{z}[1 - (1 - z^2)]|[1 + (1 - z^2) + ... + (1 - z^2)^{n-1}]| \\
&= z|[1 + (1 - z^2) + ... + (1 - z^2)^{n-1}]| \\
&\leq z[1 + |(1 - z^2)| + ... + |(1 - z^2)^{n-1}|] \\
&\leq zn
\end{aligned}$$

for all z. Substituting $z = \sqrt{\frac{\eta}{N}}x$, we have

$$|\frac{1-(1-\eta x^2/N)^n}{x}| \leq \frac{\eta n}{N}x$$

Using the fact that $0 \leq x \leq \sqrt{2N/\eta}$, we have

$$|\frac{1-(1-\eta x^2/N)^n}{x}| \leq \sqrt{\frac{2\eta}{N}}n$$

□

Theorem 2.6. *Assume* $|1-\eta s_i^2/N| < 1$, $\forall i$.

$$||\boldsymbol{\omega}_{KA,n}|| \leq \sqrt{\frac{2\eta}{N}}n||\mathbf{d}||$$

Proof.

$$\begin{aligned}
&||\boldsymbol{\omega}_{KA,n}|| = ||\mathbf{P}\text{diag}([1-(1-\eta s_1^2/N)^n]s_1^{-1}, ..., \\
&[1-(1-\eta s_r^2/N)^n]s_r^{-1}, 0, ..., 0)\mathbf{Q}^T\mathbf{d}|| \\
&\leq ||\text{diag}([1-(1-\eta s_1^2/N)^n]s_1^{-1}, ..., \\
&[1-(1-\eta s_r^2/N)^n]s_r^{-1}, 0, ..., 0)||\ ||\mathbf{d}|| \\
&\leq \sqrt{\frac{2\eta}{N}}n||\mathbf{d}|| \ \text{ (using Lemma 2.5)}
\end{aligned}$$

where $\mathbf{P}$, $\mathbf{Q}$ are orthogonal matrices. □

The bound just derived reveals a great deal of insight into the adaptation. We note that small n (the number of iterations) or small η gives a smaller bound, indicating more regularization. On the other hand, small N (the size of training data) makes the bound larger, indicating less regularization[5].

2.10 Resource Allocating Networks

A resource-allocating network (RAN) described by Platt [1991] is probably the earliest attempt in this research direction. Although RAN is fundamentally different from kernel adaptive filters, its learning procedure bears some resemblance to KLMS. Also

many of our ideas are directly influenced by this pioneer algorithm such as the novelty criterion.

RAN is a growing radial-basis function network. It stores the centers, the widthes of the centers and the linear coefficients in the format of $\{\mathbf{c}_j, w_j, a_j\}$ for the jth unit. The calculation of the output for an input pattern $\mathbf{u}$ is given by

$$\begin{aligned} x_j &= \exp(-||\mathbf{u} - \mathbf{c}_j||^2/w_j^2) \\ y &= \sum_j a_j x_j + \gamma \end{aligned}$$

where γ is a bias term.

The learning strategy is as follows: The network starts with a blank slate. When $\{\mathbf{u}, d\}$ is identified as a pattern that is not currently well presented by the network, the network allocates a new unit that memorizes the pattern. Let the index of this new unit be n. The center of the unit is set to the novel input,

$$\mathbf{c}_n = \mathbf{u}.$$

The linear coefficient on the second layer is set to the difference between the output of the network and the novel output,

$$a_n = d - y.$$

The width of the new unit is proportional to the distance from the nearest stored center to the novel input,

$$w_n = k||\mathbf{u} - \mathbf{c}_{nearest}||.$$

where k is an overlap factor. As k grows larger, the responses of the units overlap more and more.

RAN uses a two-part novelty condition. An input-output pair $\{\mathbf{u}, d\}$ is considered novel if the input is far away from existing centers,

$$||\mathbf{u} - \mathbf{c}_{nearest}|| > \delta(t),$$

and if the difference between the desired output and the output of the network is large

$$||d - y(\mathbf{u})|| > \delta_2.$$

Errors larger than δ_2 are immediately corrected by the allocation of a new unit, while errors smaller than δ_2 are gradually repaired using gradient descent. The distance $\delta(t)$ is the scale of resolution that the network is fitting at the tth input presentation. The learning session starts with $\delta(t) = \delta_{max}$, which is the largest length scale of interest, typically the size of the entire input space of non-zero probability density. The distance $\delta(t)$ shrinks until it reaches δ_{min}, which is the smallest length scale of interest. The following function is used to determine $\delta(t)$:

$$\delta(t) = \max(\delta_{max} exp(-t/\tau), \delta_{min}),$$

where τ is a decay constant.

When a new unit is not allocated, the LMS algorithm is used to decrease the error:

$$\begin{aligned} \Delta a_j &= \eta(d - y)x_j, \\ \Delta\gamma &= \eta(d - y), \\ \Delta\mathbf{c}_j &= \frac{2\eta}{w_j}(\mathbf{u} - \mathbf{c}_j)x_j[(d - y)a_j] \end{aligned}$$

It is shown that RAN is able to learn quickly, accurately and to form a compact representation. However, we have to point out that RAN is built upon intuition and heuristics; it is not a convex optimization problem; its convergence is hard to prove and not guaranteed. Unlike RAN, KLMS is not restricted to the Gaussian kernel and uses a step-size parameter to gradually correct the error. On the whole, KLMS is conceptually and practically simpler.

2.11 Computer Experiments

2.11.1 KLMS Applied to Mackey-Glass Time Series Prediction

The first example is the short-term prediction of the Mackey-Glass (MG) chaotic time series[6]. It is generated from the following time-delay ordinary differential equation

$$\frac{dx(t)}{dt} = -bx(t) + \frac{ax(t-\tau)}{1+x(t-\tau)^{10}} \tag{2–64}$$

with $b = 0.1$, $a = 0.2$, and $\tau = 30$. The time series is discretized at a sampling period of 6 seconds. A segment of 5000 points of the time series is generated using (2–74) and stored in the mat file MK30.mat.

The first question is how to select the best filter order. Inspired by chaos theory, a principled approach to select the minimal filter-order that preserves the shape of the trajectories (after the transients die down) is called the Takens embedding theorem [Takens, 1981]. According to this theorem, the optimal embedding for this system is around 7. In this example, we choose the time embedding as 10, i.e. $\mathbf{u}(i) = [x(i-10), x(i-9), ..., x(i-1)]^T$ (the 10 most recent values in the past) are used as the input to predict the present one $x(i)$ which is the desired response in this example. The code for the experiment can be found at `http://www.cnel.ufl.edu/~weifeng/publication.htm`. The readers are encouraged to play with all the parameters.

PART 1: A segment of 500 samples is used as the training data and another 100 as the test data. The data are corrupted by additive Gaussian noise with zero mean and 0.04 standard deviation. The purpose of the experiment is to compare the performance of a linear combiner trained with LMS and KLMS. The step-size parameter for LMS is 0.2. For KLMS, the Gaussian kernel (1–24) with $a = 1$ is chosen and the step-size parameter is also 0.2. Figure 2-5 is a typical plot of the learning curves. At each iteration, the Mean Square Error (MSE) is computed on the test set using the filter resulting from in the training set. As expected, KLMS converges to a smaller value of MSE due to its nonlinear nature.

Surprisingly, the rate of decay of both learning curves is basically the same, which suggests that the eigenvalue spread in the RKHS is similar to that of the input space.

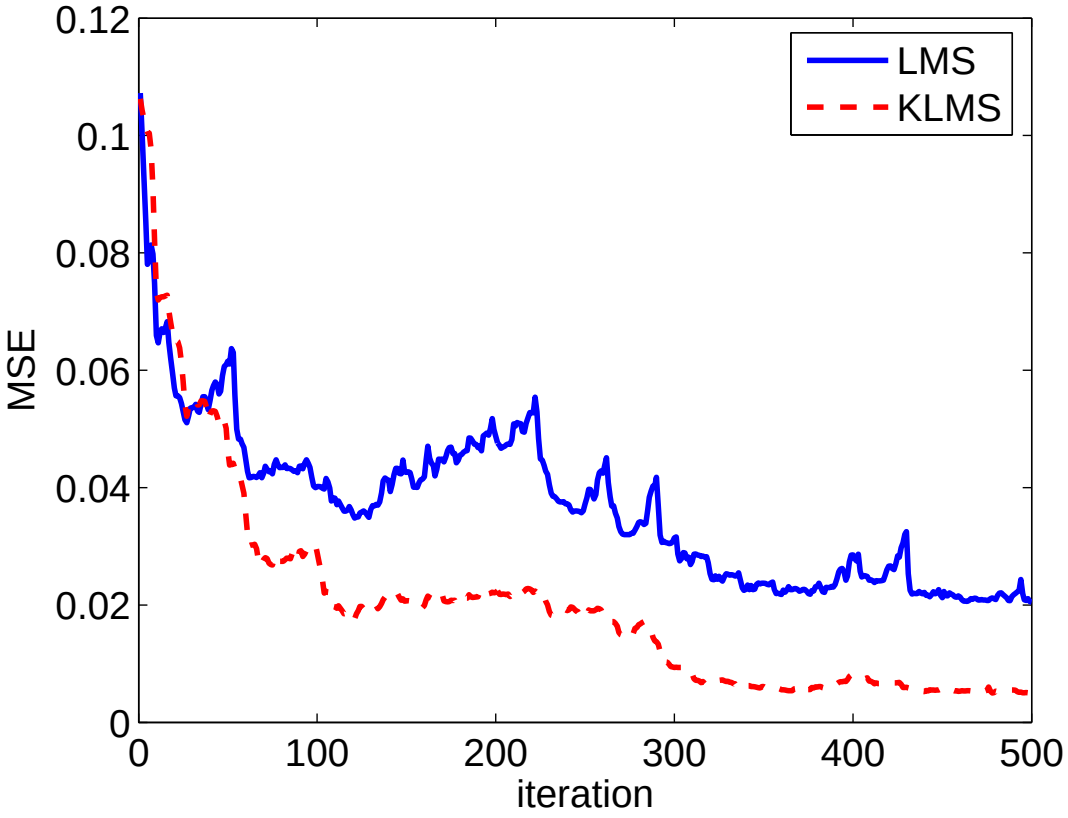

Figure 2-5. Learning curves of LMS and KLMS in Mackey-Glass time series prediction

PART 2: This is a more comprehensive comparison among LMS, KLMS and a regularization network (RN), which serves as a batch-mode baseline. RN is a classical nonlinear modeling tool using a radial-basis function network topology specified by the kernel utilized [Poggio and Girosi, 1990]. The Gaussian kernel with $a = 1$ is chosen for both RN and KLMS. In RN, every input point is used as the center and the training is done in batch mode. One hundred Monte Carlo simulations are run with different realizations of noise. The results are summarized in Table 2-1.

All the results in these tables are in the form of "average $\pm$ standard deviation". As we can observe in Table 2-1, the performance of KLMS is much better than the linear LMS, which is to be expected (Mackey-Glass time series is a nonlinear system)

Table 2-1. Performance comparison of KLMS with different step sizes and RN with different regularization parameters in Mackey-Glass time series prediction

Algorithm	Training MSE	Testing MSE
Linear LMS	0.021 ± 0.002	0.026 ± 0.007
KLMS ($\eta = 0.1$)	0.0074 ± 0.0003	0.0069 ± 0.0008
KLMS ($\eta = 0.2$)	0.0054 ± 0.0004	0.0056 ± 0.0008
KLMS ($\eta = 0.6$)	0.0062 ± 0.0012	0.0058 ± 0.0017
RN ($\lambda = 0$)	0 ± 0	0.012 ± 0.004
RN ($\lambda = 1$)	0.0038 ± 0.0002	0.0039 ± 0.0008
RN ($\lambda = 10$)	0.011 ± 0.0001	0.010 ± 0.0003

Table 2-2. Complexity comparison of LMS, KLMS and RN at iteration i

Algorithm	Computation	Memory
LMS	$O(L)$	$O(L)$
KLMS	$O(i)$	$O(i)$
RN	$O(i^3)$	$O(i^2)$

and is comparable to RN with the best regularization. This is indeed surprising since RN can be viewed as a batch mode kernel regression method versus KLMS which is a straight stochastic gradient approach implemented in RKHS. It is interesting to compare the design and performance of KLMS with different step sizes and RN with different regularization parameters since each controls the stability of the obtained solution. First of all, when the regularization parameter is zero, RN performs poorly on the test set (worse than the linear solution), which indicates that the solution is poorly regularized. RN is capable of outperforming KLMS with the proper regularization parameter ($\lambda = 1$), but the difference is small and at the expense of a more complex solution as well as with a careful selection of the regularization parameter.

Table 2-2 summarizes the computational complexity of the three algorithms. KLMS effectively reduces the computational complexity and memory storage when compared with RN.

PART 3: We compute the solution norms to support our theory that the norm of the KLMS solution is well-bounded. As we see in Tables 2-1 and 2-3, increasing the

step-size parameter in KLMS increases the norm of the solution but fails to increase the performance because of the gradient noise in the estimation (misadjustment).

Table 2-3. Solution norms of KLMS with different step sizes and RN with different regularization parameters in Mackey-Glass time series prediction

Algorithm	Solution norm
KLMS ($\eta = 0.1$)	0.84 ± 0.02
KLMS ($\eta = 0.2$)	1.14 ± 0.02
KLMS ($\eta = 0.6$)	1.73 ± 0.06
RN ($\lambda = 0$)	3375 ± 639
RN ($\lambda = 1$)	1.47 ± 0.03
RN ($\lambda = 10$)	0.55 ± 0.01

PART 4: Different noise variances σ^2 are used in the data to further validate KLMS's applicability. As we see in Tables 2-4 and 2-5, KLMS performs consistently on the training and test sets with different noise levels and degrades gracefully with increasing noise variance. It is observed that at severe noise level ($\sigma = .5$), all methods fall apart since the noise component will no longer correspond to the smallest singular value as required by Tikhonov regularization. With small noise, the regularization network outperforms KLMS since the misadjustment becomes the bottleneck. This is a good illustration of the difficulty KLMS may face to balance among convergence, misadjustment and regularization. But remember KLMS is a much simpler, online algorithm and the performance gap compared with RN is the price to be paid. Throughout this set of simulations, the kernel used in KLMS and RN is the Gaussian kernel with $a = 1$. The learning step is 0.1 for both the linear LMS and KLMS. The regularization parameter of RN is set at the best value ($\lambda = 1$).

Table 2-4. Performance comparison of LMS, KLMS and RN with different noise levels in Mackey-Glass time series prediction (training MSE)

Algorithm	Linear LMS	KLMS ($\eta = 0.1$)	RN ($\lambda = 1$)
$\sigma = .005$	$0.017 \pm 5e-5$	$0.0050 \pm 2e-5$	$0.0014 \pm 1e-5$
$\sigma = .02$	0.018 ± 0.0002	0.0055 ± 0.0001	$0.0020 \pm 6e-5$
$\sigma = .04$	0.021 ± 0.002	0.0074 ± 0.0003	0.0038 ± 0.0002
$\sigma = .1$	0.033 ± 0.001	0.019 ± 0.001	0.010 ± 0.0005
$\sigma = .5$	0.326 ± 0.015	0.252 ± 0.010	0.097 ± 0.003

Table 2-5. Performance comparison of LMS, KLMS and RN with different noise levels in Mackey-Glass time series prediction (testing MSE)

Algorithm	Linear LMS	KLMS ($\eta = 0.1$)	RN ($\lambda = 1$)
$\sigma = .005$	0.018 ± 0.0002	0.0041 ± 0.0001	$0.0012 \pm 6e-5$
$\sigma = .02$	0.018 ± 0.0007	0.0046 ± 0.0004	0.0016 ± 0.0002
$\sigma = .04$	0.026 ± 0.007	0.0069 ± 0.0008	0.0039 ± 0.0008
$\sigma = .1$	0.031 ± 0.005	0.018 ± 0.003	0.017 ± 0.003
$\sigma = .5$	0.363 ± 0.057	0.332 ± 0.052	0.331 ± 0.052

PART 5: Any kernel method, including KLMS, needs to choose a suitable kernel and its bandwidth. The effect of different kernels and different kernel parameters on KLMS is demonstrated. In the case of the Gaussian kernel (1–24), we choose 3 kernel parameters: 10, 2, and 0.2. The learning rate is set at 0.1 for both the linear LMS and KLMS and the regularization parameter of RN is 1 throughout the simulation. The results are summarized in Table 2-6. As expected, too small or too large kernel sizes hurt performance for both KLMS and RN. In this problem, a kernel size around 1 gives the best performance on the test set.

Table 2-6. Effect of kernel size of Gaussian kernel on performance of KLMS and RN in Mackey-Glass time series prediction

Algorithm	Training MSE	Testing MSE
Linear LMS	0.022 ± 0.001	0.022 ± 0.001
KLMS ($a = 10$)	0.0085 ± 0.0005	0.0078 ± 0.0010
KLMS ($a = 2$)	0.0061 ± 0.0003	0.0056 ± 0.0014
KLMS ($a = .2$)	0.017 ± 0.0007	0.016 ± 0.0010
RN ($a = 10$)	0.0040 ± 0.0002	0.0068 ± 0.0009
RN ($a = 2$)	0.0043 ± 0.0002	0.0047 ± 0.0006
RN ($a = .2$)	0.0098 ± 0.0003	0.0092 ± 0.0005

PART 6: In the case of the polynomial kernel (1–25), the order is set to 2, 5, and 8. The learning rate is chosen accordingly in KLMS as listed in Table 2-7 (recall the relation between the learning rate and the trace of the Gram matrix). It is observed that the performance deteriorates substantially when p is too large (> 8) for KLMS. This is also validated by the misadjustment formula (2–28).

Table 2-7. Effect of order of polynomial kernel on performance of KLMS and RN in Mackey-Glass time series prediction

Algorithm	Training MSE	Testing MSE
KLMS ($p = 2, \eta = 0.1$)	0.010 ± 0.001	0.009 ± 0.002
KLMS ($p = 5, \eta = 0.01$)	0.0099 ± 0.0006	0.0099 ± 0.0007
KLMS ($p = 8, \eta = .0006$)	0.027 ± 0.009	0.025 ± 0.009
RN ($p = 2, \lambda = 1$)	0.0064 ± 0.0005	0.0066 ± 0.0008
RN ($p = 5, \lambda = 1$)	0.0034 ± 0.0003	0.0059 ± 0.0007
RN ($p = 8, \lambda = 1$)	0.0014 ± 0.0001	0.0078 ± 0.0004

PART 7: It is noted in the theoretical analysis that the training data size will not affect the regularization of KLMS. To illustrate this behavior, we choose different training data sizes to see how KLMS performs. The noise variance is set at 0.05 and the numbers of training data are 1000, 2000 and 4000, respectively. Other parameters are the same as in the first set of simulations. As presented in Table 2-8, KLMS performs very consistently on the training and test sets with increasing number of training data.

Table 2-8. Performance comparison of LMS and KLMS with different training data sizes

Algorithm	Training MSE	Testing MSE
LMS ($N = 1000$)	0.020 ± 0.0004	0.019 ± 0.0015
LMS ($N = 2000$)	0.019 ± 0.0004	0.018 ± 0.0009
LMS ($N = 4000$)	0.019 ± 0.0003	0.020 ± 0.0016
KLMS ($N = 1000$)	0.0060 ± 0.0002	0.0062 ± 0.0009
KLMS ($N = 2000$)	0.0058 ± 0.0002	0.0053 ± 0.0010
KLMS ($N = 4000$)	0.0054 ± 0.0001	0.0058 ± 0.0007

PART 8: In this simulation, we examine the effect of the regularization parameter on the performance of NORMA (leaky KLMS). Twenty regularization parameters are chosen within [0, 0.1]. For each regularization parameter, fifty Monte Carlo simulations are performed with different realizations of noise ($\sigma = 0.01$). The final average MSE on the testing set is plotted in Figure 2-6 along with its standard deviation. As we see, the explicit regularization has a detrimental effect in this example.

PART 9: We next test how the novelty criterion affects the performance. A segment of 1000 samples is used as the training data and another 200 as the test data. All the data are corrupted by Gaussian noise with zero mean and 0.0001 variance. The typical learning

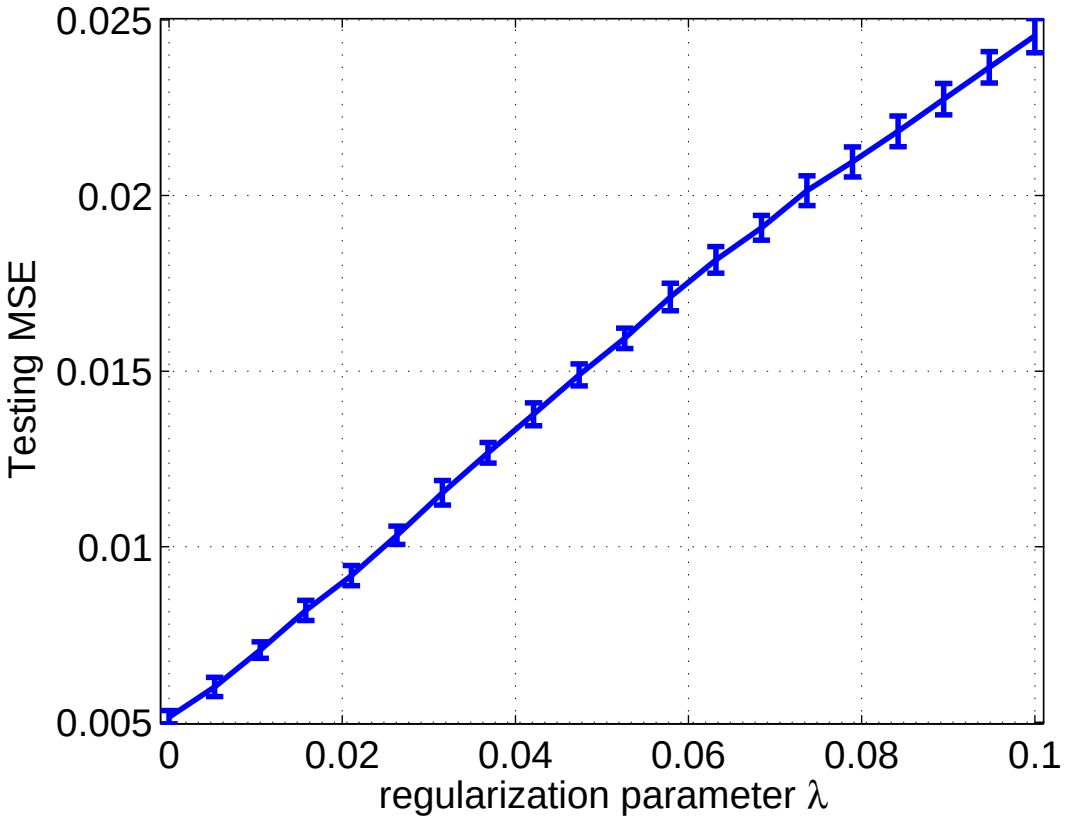

Figure 2-6. Performance of NORMA with explicit regularization in Mackey-Glass time series prediction

curves are shown in Figure 2-7 with the thresholds in the novelty criterion $\delta_1 = 0.1$ and $\delta_2 = 0.05$. The step-size parameter used is 0.1. With the previous results, we know that the optimal kernel bandwidth is around 1 which means that δ_1 is about 0.07 $(0.1/\sqrt{2a})$. Also the testing MSE is around 0.006 which means δ_2 is about 0.08 $(\sqrt{0.006})$. Different thresholds are tested and the results are summarized in Table 2-9. The MSE is calculated from the last 100 points of the learning curves. It is seen that the complexity can be reduced dramatically with the novelty criterion preserving the prediction accuracy. Of course with δ_1 and δ_2 too large, the performance degrades.

PART 10: We further examine how the novelty criterion affects the growth pattern of KLMS. A segment of 4500 samples is used as the training data and another 200 as the test data. All the data are corrupted by Gaussian noise with zero mean and 0.0001 variance.

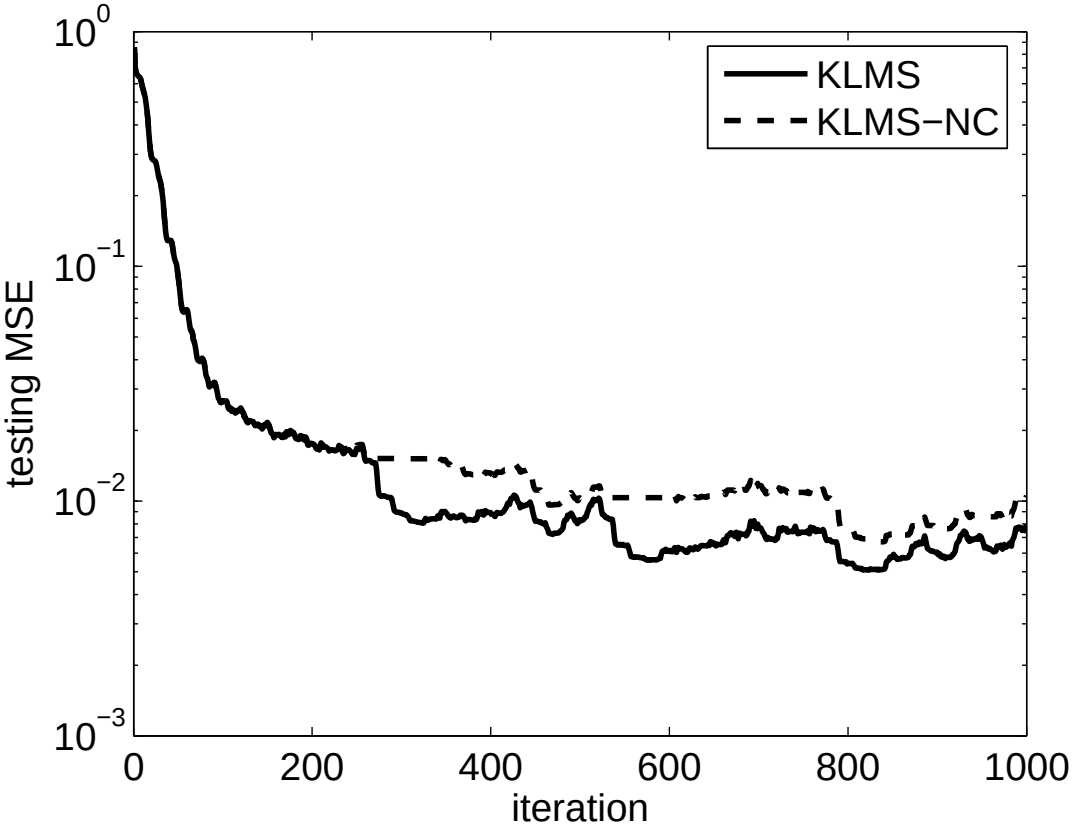

Figure 2-7. Learning curves of KLMS with and without novelty criterion in Mackey-Glass time series prediction

Table 2-9. Performance of KLMS with novelty criterion in Mackey-Glass time series prediction

Algorithm	Parameters	Testing MSE	Dictionary size
KLMS		0.0062 ± 0.00048	1000
KLMS-NC	$\delta_1 = 0.05$, $\delta_2 = 0.02$	0.0065 ± 0.00051	754
KLMS-NC	$\delta_1 = 0.05$, $\delta_2 = 0.05$	0.0066 ± 0.00048	528
KLMS-NC	$\delta_1 = 0.05$, $\delta_2 = 0.1$	0.0072 ± 0.00033	286
KLMS-NC	$\delta_1 = 0.1$, $\delta_2 = 0.05$	0.0078 ± 0.00055	490
KLMS-NC	$\delta_1 = 0.2$, $\delta_2 = 0.05$	0.0134 ± 0.00041	284

The thresholds in the novelty criterion are set as $\delta_1 = 0.05$ and $\delta_2 = 0.1$. All other settings are the same as in **PART 9**. The learning curve is plotted in Figure 2-8. The growth curve in Figure 2-9 shows the network size at each iteration. Only 571 inputs out of 4500 (13%) are eventually selected into the dictionary. The growth rate curve shows the average growth rate in a fixed-width window (window length is 100 in Figure 2-10). It is seen that

the network growth is effectively contained with the novelty criterion. The growth rate drops dramatically from around 0.8 to 0.05. This is perhaps the worst-case scenario since the Mackey-Glass time series is chaotic and never repeats itself.

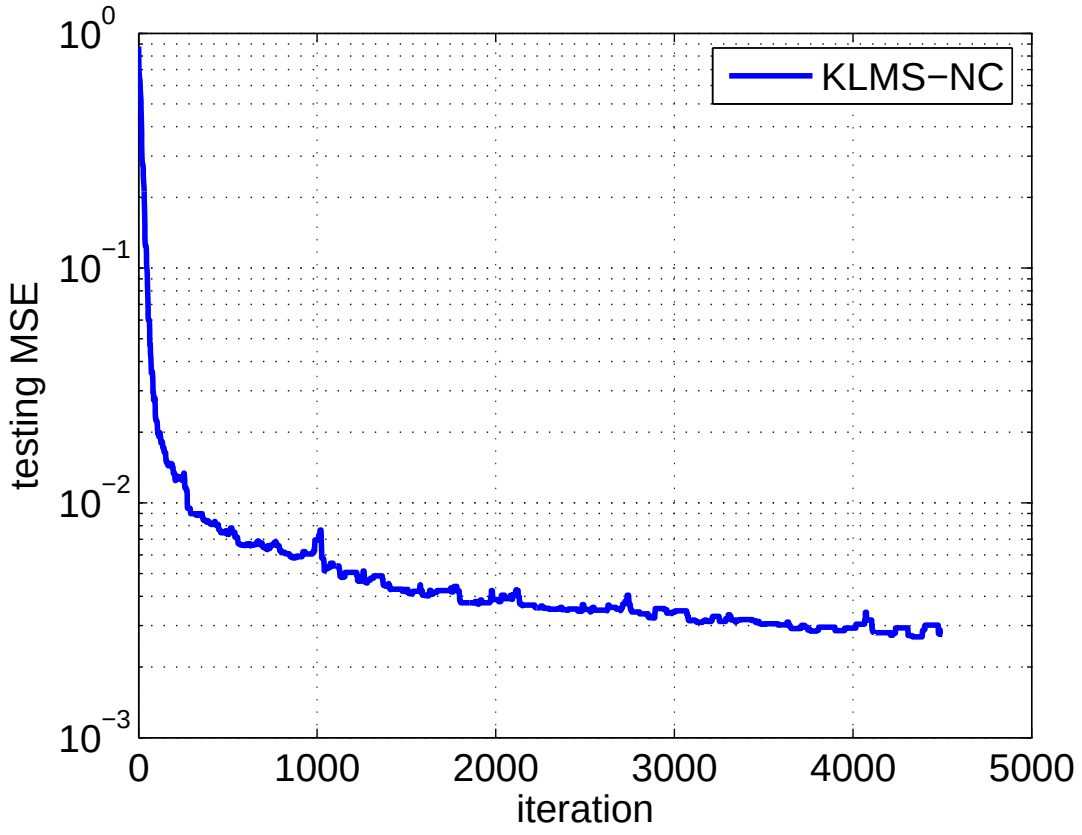

Figure 2-8. Learning curve of KLMS with novelty criterion in Mackey-Glass time series prediction

2.11.2 KLMS Applied to Nonlinear Channel Equalization

The LMS algorithm is widely used in channel equalization and we tested KLMS on a nonlinear channel equalization problem[7]. The nonlinear channel model consists of a serial connection of a linear filter and a memoryless nonlinearity (See Figure 3-7). This kind of model has been used to model digital satellite communication channels and digital magnetic recording channels.

The problem setting is as follows: A binary signal $\{s(1), s(2), ..., s(N)\}$ is fed into a nonlinear channel. At the receiver end of the channel, the signal is further corrupted by

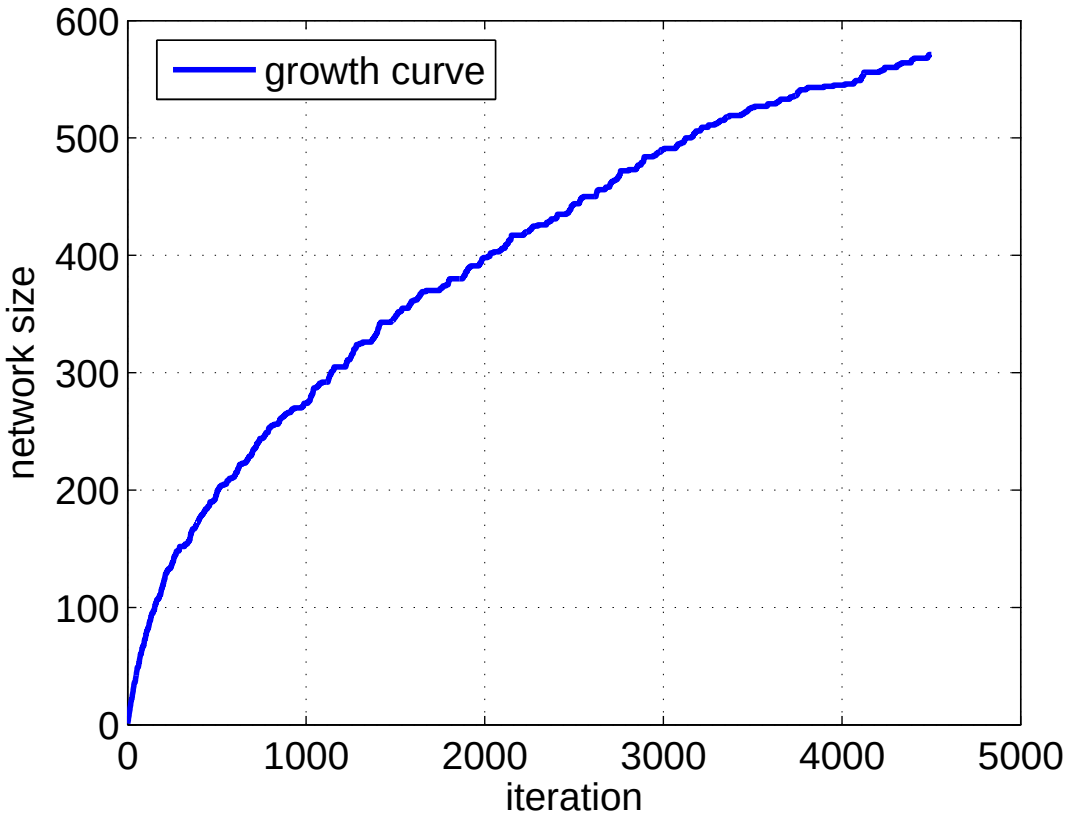

Figure 2-9. Growth curve of KLMS with novelty criterion in Mackey-Glass time series prediction

additive white Gaussian noise and is then observed as $\{r(1), r(2), ..., r(N)\}$. The aim of channel equalization is to construct an "inverse" filter that reproduces the original signal with as low an error rate as possible. It is easy to formulate it as a regression problem, with examples $\{([r(i), r(i+1), ..., r(i+l)], s(i-D))\}$, where l is the time embedding length, and D is the equalization time lag. $l = 5$ and $D = 2$ are used in this experiment. The nonlinear channel model is defined by $x(i) = s(i) + 0.5s(i-1)$, $r(i) = x(i) - 0.9x(i)^2 + n(i)$, where $n(i)$ is the white Gaussian noise with a variance of σ^2.

PART 1: The performance of LMS, KLMS and RN (the latter as a batch-mode baseline) are compared. The filters are trained with 1000 data and fixed afterwards. Testing is performed on a 5000-sample random test sequence. The Gaussian kernel with $a = 0.1$ and

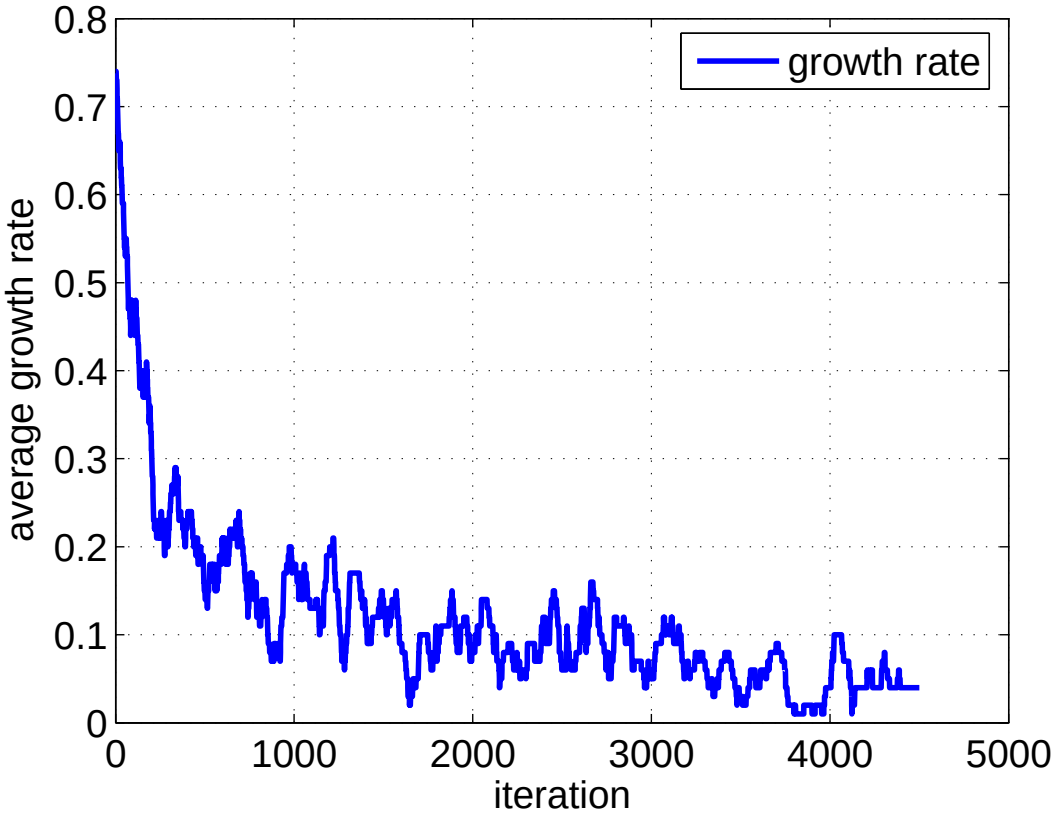

Figure 2-10. Average growth rate curve of KLMS with novelty criterion in Mackey-Glass time series prediction

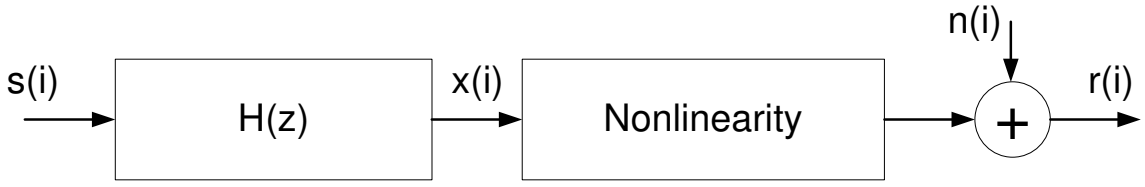

Figure 2-11. Basic structure of a nonlinear channel

step-size parameter $\eta = 0.2$ are used in KLMS for best results. The step-size parameter of LMS is set at 0.01. Figure 2-12 is a typical plot of the learning curves.

PART 2: The three algorithms, namely LMS, KLMS and RN, are tested on this problem with different noise levels. The results are presented in Table 2-10; each entry consists of the average and the standard deviation for 100 Monte Carlo independent tests. The results show that, RN outperforms KLMS in terms of bit error rate (BER) but not by

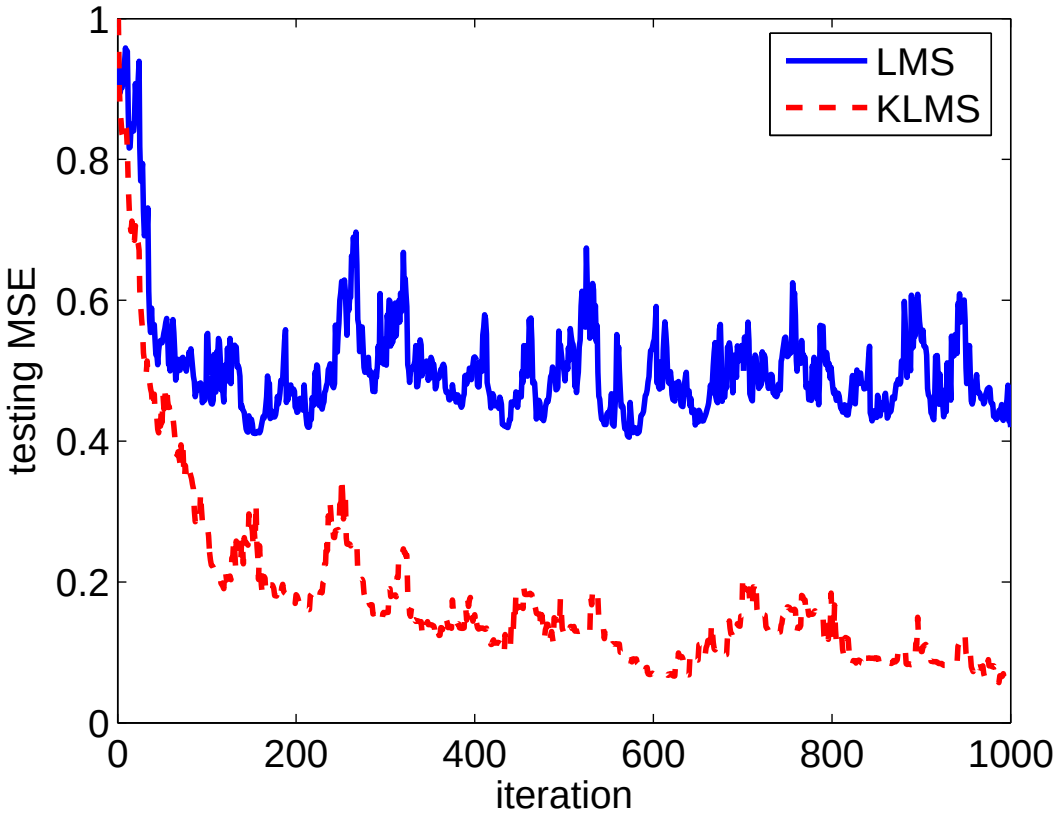

Figure 2-12. Learning curves of LMS and KLMS in nonlinear channel equalization ($\sigma = 0.4$)

much, which is surprising since one is a batch method and the other is online. They both outperform the conventional LMS substantially as can be expected because the channel is nonlinear.

Table 2-10. Performance comparison of LMS, KLMS and RN in nonlinear channel equalization

Algorithm	Linear LMS ($\eta = .005$)	KLMS ($\eta = 0.1$)	RN ($\lambda = 1$)
BER ($\sigma = .1$)	0.162 ± 0.014	0.020 ± 0.012	0.008 ± 0.001
BER ($\sigma = .4$)	0.177 ± 0.012	0.058 ± 0.008	0.046 ± 0.003
BER ($\sigma = .8$)	0.218 ± 0.012	0.130 ± 0.010	0.118 ± 0.004

2.12 Conclusion

The KLMS algorithm is a stochastic gradient methodology to solve least squares problems in RKHS. Since the update equation can be written in terms of inner product,

KLMS can be efficiently computed in the input space. The good approximation ability of KLMS stems from the fact that the transformed data include possibly infinite different features of the original data. In the framework of stochastic projection, the space spanned by $\{\boldsymbol{\varphi}(i)\}$ is so large that the projection error of the desired signal $d(i)$ could be very small [Parzen, 1959], as is well known from Cover's theorem [Haykin, 2009]. This capability includes modeling of nonlinear systems, which is the main reason why KLMS can achieve good performance in the Mackey-Glass system prediction and nonlinear channel equalization.

As demonstrated by the experiments, KLMS has general applicability due to its simplicity; in particular, it does not need to work with large Gram matrices as most of the kernel algorithms because it utilizes the data on the basis of one sample at a time. KLMS may be very useful in problems like nonlinear channel equalization, nonlinear system identification, nonlinear active noise control, where online filters are a necessity. Almost all the literature for LMS can be used to analyze KLMS; especially, its convergence and stability are well understood. Also in the framework of RKHS, any Mercer kernel can be used in KLMS instead of restricting the architecture to the Gaussian kernel as in RAN.

KLMS is a simple and effective nonlinear filter design. It has the universal approximation capability in stationary environments. Its convergence property and regularization property are mainly controlled by a simple parameter (step-size parameter). Practical approaches are available to select the kernel, to choose the step-size parameter, and to contain the network growth. Issues that require future investigation include pruning methods to further reduce network size and adaptive mechanisms for kernel size to better capture local data structure.

Notes and References

1. **Radial-Basis Function Networks.** Radial-basis function networks are motivated to find a surface in a multidimensional space that provides a best fit to the training data, with the criterion for "best fit" being measured in some statistical sense. They were first introduced in

the solution of the real multivariate interpolation problem. The early work on this subject is surveyed in [Powell, 1985]. A survey of their use in the field of neural networks can be found in [Light, 1992].

In a strict sense, the interpolation problem may be stated:

> Given a set of N different points $\{\mathbf{x}_i \in \mathbb{R}^L\}_{i=1}^N$ and a corresponding set of N real numbers $\{d_i \in \mathbb{R}\}_{i=1}^N$, find a function $f : \mathbb{R}^L \to \mathbb{R}$ that satisfies the interpolation condition:
>
> $$f(\mathbf{x}_i) = d_i, \quad i = 1, 2, ..., N \tag{2–65}$$

For strict interpolation as specified above, the interpolating surface is constrained to pass through all the training data points, which may be undesirable when the observed data are noisy. The radial-basis functions (RBF) technique chooses a function f which is a linear combination of a set of basis functions:

$$f(\mathbf{x}) = \sum_{i=1}^{N} a_i g(||\mathbf{x} - \mathbf{x}_i||) \tag{2–66}$$

where $g(||\mathbf{x} - \mathbf{x}_i||)$ is a set of N arbitrary functions, known as *radial-basis functions*, and $||\cdot||$ denotes a norm between $\mathbf{x} - \mathbf{x}_i$, which is usually Euclidean. Notice that the centers of the radial-basis functions are the regressors from the observed data. Using the condition of interpolation (2–65) in (2–66), we have

$$\begin{pmatrix} g_{11} & g_{12} & ... & g_{1N} \\ g_{21} & g_{22} & ... & g_{2N} \\ ... & ... & ... & ... \\ g_{N1} & g_{N2} & ... & g_{NN} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ ... \\ a_N \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ ... \\ d_N \end{pmatrix} \tag{2–67}$$

where

$$g_{ij} = g(||\mathbf{x}_i - \mathbf{x}_j||), \quad i = 1, ..., N; \; j = 1, ..., N$$

If the function form of g is fixed and known, we can solve this linear system for the unknown coefficients $\{a_i\}_{i=1}^N$. Let $\mathbf{G}$ denote an $N \times N$ matrix with elements g_{ij} at the (i,j)th entry and

$$\mathbf{d} = [d_1, d_2, ..., d_N]^T$$

$$\mathbf{a} = [a_1, a_2, ..., a_N]^T$$

$\mathbf{G}$ is called the *interpolation matrix*, $\mathbf{d}$ the *desired response vector* and $\mathbf{a}$ the *linear weight vector*. We may rewrite (2–67) in the compact form

$$\mathbf{Ga} = \mathbf{d} \tag{2–68}$$

By the matrix theory, we know that there exists a unique solution of $\mathbf{a}$, if and only if $\mathbf{G}$ is invertible. An important theorem proved by Micchelli [1986] states:

> Let $\{\mathbf{x}_i\}_{i=1}^N$ be a set of *distinct* points in $\mathbb{R}^L$ and g an arbitrary nonlinear function. Then the N-by-N interpolation matrix $\mathbf{G}$, whose ijth element is $g_{ij} = g(||\mathbf{x}_i - \mathbf{x}_j||)$, is *nonsingular*.

Therefore, as long as the inputs are distinct, the inverse of $\mathbf{G}$ exists and the linear weight vector can be simply solved by

$$\mathbf{a} = \mathbf{G}^{-1}\mathbf{d} \tag{2–69}$$

There is a large class of radial-basis functions that is covered by Micchelli's theorem. Commonly used types of radial basis functions include

1. Gaussian:
$$g(||\mathbf{x}_i - \mathbf{x}_j||) = \exp(-a||\mathbf{x}_i - \mathbf{x}_j||^2) \text{ for some } a > 0 \tag{2–70}$$

2. Multiquadrics:
$$g(||\mathbf{x}_i - \mathbf{x}_j||) = \sqrt{||\mathbf{x}_i - \mathbf{x}_j||^2 + c^2} \text{ for some } c > 0 \tag{2–71}$$

3. Inverse multiquadric:
$$g(||\mathbf{x}_i - \mathbf{x}_j||) = \frac{1}{\sqrt{||\mathbf{x}_i - \mathbf{x}_j||^2 + c^2}} \text{ for some } c > 0 \tag{2–72}$$

4. Thin plate spline:

$$g(||\mathbf{x}_i - \mathbf{x}_j||) = ||\mathbf{x}_i - \mathbf{x}_j||^2 \ln(||\mathbf{x}_i - \mathbf{x}_j||) \tag{2–73}$$

Clearly, RBF networks, in the form of (2–66), have the same *shallow* structure as KLMS. The linear coefficients are solved by inverting the interpolation matrix. The complexity in the training phase follows a cubic rule $O(N^3)$. Another important thing is that the interpolation matrix is not guaranteed to be positive-definite. For example, an interpolation matrix made from the multiquadrics of (2–71) has $N - 1$ negative eigenvalues and only one positive eigenvalue [Micchelli, 1986].

2. **Kernel Selection.** For a thorough treatment on reproducing kernels, see [Schölkopf and Smola, 2002, Shawe-Taylor and Cristianini, 2004, Rasmussen and Williams, 2006].

All the kernel methods need to choose the kernel type and parameters. The most popular method so far is by cross-validation [Racine, 1993, Cawley and Talbot, 2003, An et al., 2007]. The nearest neighbor method is also used in the resource-allocating networks which allow the adaptation of the kernel size during the learning. With a close relation to Gaussian process theory, maximum marginal likelihood [Rasmussen and Williams, 2006] is also applicable, which we will discuss later. Besides, the general kernel selection problem has been studied as a convex optimization through parameterization of the kernel function family [Micchelli and Pontil, 2005, Argyriou et al., 2005, Chapelle et al., 2002].

3. **Pruning in Kernel Methods.** Simple pruning strategies include pruning the oldest unit in the dictionary [Van Vaerenbergh et al., 2006], pruning randomly [Cavallanti et al., 2007], pruning the unit with the least coefficient [Dekel et al., 2006] and pruning the unit with the smallest outputs on recent inputs.

4. **Low Rank Approximation.** Since the kernel space is a high dimensional space (can be infinite in case of Gaussian kernel), to solve the adaptive filtering problem directly in the primal space need low-rank approximation methods such as Nyström method [Williams and Seeger, 2001], incomplete Cholesky factorization [Fine and Scheinberg, 2001] and kernel principal component analysis [Schölkopf et al., 1998].

5. **Gradient Descent is Regularization.** Gradient descent method is known for its regularization property in the literature as illustrated by the deterministic analysis of early stopping in inverse problems (see [Raudys and Cibas, 1996, Engl et al., 2000, Hagiwara and Kuno, 2000, Yao et al., 2007]).

6. **Mackey-Glass Equation.** The Mackey-Glass equation is a nonlinear time-delay differential equation

$$\frac{dx(t)}{dt} = -bx(t) + \frac{ax(t-\tau)}{1+x(t-\tau)^n} \tag{2–74}$$

where a, b, n and τ are real numbers. This equation displays characteristics of periodic and chaotic dynamics. Mackey and Glass [1977] first used it to model physiological control systems such as electrolytes, oxygen, glucose, and blood cells in the blood, blood pressure to the brain and various organs. Equation (2–74) represents a typical feedback system. In real feedback systems, there is typically a time lag τ between the sensing of the value of a variable under control, and the mounting of an appropriate response, which requires the dependency of x on the time delayed value of $x(t-\tau)$. For example, following a loss of blood cells, it can take many days before new blood cells can be produced through the activation, differentiation, and proliferation of the appropriate blood stem cells. Then, Farmer [1982] recognized that increasing the value of τ in (2–74) increases the dimension of the attractor in this chaotic system. This observation, and the simplicity of the equation, has led to the evolution of this equation into one of the standard models used to test algorithms for nonlinear modeling capability [Farmer and Sidorowich, 1987, Crowder, 1990, Platt, 1991, Martinetz et al., 1993, Mukherjee et al., 1997, Müller et al., 1997]. For more details, please refer to Glass and Mackey [1988], Beuter et al. [2003].

7. **Adaptive Channel Equalization.** In 1965, Lucky [1965] made a major breakthrough in the equalization problem by proposing a *zero-forcing algorithm* for automatically adjusting the tap weights of a transversal equalizer. Gersho [1969] and Proakis and Miller [1969] independently reformulated the adaptive equalization problem using a mean-square-error criterion. In 1972, using the LMS algorithm, Ungerboeck presented a detailed mathematical analysis of the convergence properties of an adaptive transversal equalizer. In 1974, Godard used Kalman filter theory to derive a powerful algorithm for adjusting the tap weights of a transversal equalizer.

It has been shown by Sayed [2003] that the optimal equalizer for a linear channel is actually nonlinear. Also from the viewpoint of communication theory, any physical channel exhibits nonlinear characteristics to some extent [Proakis, 2000]. Most notable examples include digital satellite communication channels [Benedetto and Biglieri, 1983, Kechriotis et al., 1994] and digital magnetic recording channels [Sands and Cioffi, 1993]. Theodoridis et al. [1992] presented a review of the use of clustering techniques for the channel equalization problem. The application of a radial-basis function network to digital communications channel equalization was examined in [Chen et al., 1993a,b]. It is shown that the radial-basis function network can be employed to implement the optimal Bayesian symbol-decision equalizer. Cha and Kassam [1995] investigated the use of a complex-value radial-basis function network. Kechriotis et al. [1994] introduced an adaptive recurrent neural network (RNN) based equalizer which is very suitable for high-speed channel equalization. RNN equalizers have comparable performance with traditional linear filter based equalizers when the channel interferences are relatively mild; however, they outperform the linear counterparts by several orders of magnitude when either the channel's transfer function has spectral nulls or severe nonlinear distortion is present. In addition, the small-size RNN equalizers are reported to outperform multilayer perceptron equalizers in many cases. Adali et al. [1997] have shown that the single and multilayer perceptron models can be used to implement the so-called maximum partial likelihood estimation which are very useful for dependent observations and sequential processing. More recently, support vector machines have been used to solve the equalization problem [Sebald and Bucklew, 2000]. Erdogmus et al. [2001] studied the use of multilayer perceptron equipped with information-theoretic cost functions to compensate nonlinear effects caused by practical transmitter power amplifiers.

CHAPTER 3
KERNEL AFFINE PROJECTION ALGORITHMS

This chapter extends the kernel least mean square algorithm to the class of algorithms that fall under Goodwin's online learning model, creating a rich, flexible and cohesive taxonomy of online algorithms in RKHS. The center piece of Goodwin's family is the affine projection algorithms (APA), which inherit the simplicity and online nature of LMS while reducing the gradient noise by using multiple samples, therefore boosting LMS performance. APA appear as intermediate complexity algorithms between the LMS and the recursive least squares (RLS).

As can be expected the affine projection algorithms can be extended to RKHS using the basic methodology outlined for KLMS, and give rise to the kernel affine projection algorithms (KAPA) family [Liu and Príncipe, 2008b]. Besides the number of samples, the other two degrees of freedom in the taxonomy are: (1) the regularization in the cost function for better generalization and (2) the Newton updates which avoid the slowness of gradient descent produced by the eigenvalue spread of the input correlation matrix. Of course the performance and computational complexity of all these versions are different, but they provide a full range of options to users trying to meet trade-offs between data rates (or database sizes) and hardware constraints.

More interestingly, KAPA provides a unifying model for several existing neural network techniques, including kernel least mean square algorithms, sliding-window kernel recursive least squares algorithm and regularization networks (Figure 1-5). Therefore, many insights can be gained into the basic relations among them and the trade-off between computation complexity and performance. We will start with a review of affine projection algorithms, focusing on its subtle variations due to different optimization techniques. Then the matrix inversion lemma is used to derive equivalent representations which are more suitable for kernel extensions. Finally, the kernel affine projection algorithms follow naturally.

3.1 Affine Projection Algorithms

Let d be a zero-mean scalar-valued random variable and let $\mathbf{u}$ be a zero-mean $L \times 1$ random variable with a positive-definite covariance matrix $\mathbf{R}_{\mathbf{u}} = \mathbf{E}[\mathbf{u}\mathbf{u}^T]$. The cross-covariance vector of d and $\mathbf{u}$ is denoted by $\mathbf{r}_{d\mathbf{u}} = \mathbf{E}[d\mathbf{u}]$. The weight vector $\mathbf{w}$ that solves

$$\min_{\mathbf{w}} J(\mathbf{w}) = \mathbf{E}|d - \mathbf{w}^T\mathbf{u}|^2 \tag{3–1}$$

is given by $\mathbf{w}^o = \mathbf{R}_{\mathbf{u}}^{-1}\mathbf{r}_{d\mathbf{u}}$ (the Wiener solution) [Haykin, 2002].

Several methods to approximate $\mathbf{w}^o$ iteratively exist. For example, the gradient descent method:

$$\mathbf{w}(0) = \text{initial guess}; \quad \mathbf{w}(i) = \mathbf{w}(i-1) + \eta[\mathbf{r}_{d\mathbf{u}} - \mathbf{R}_{\mathbf{u}}\mathbf{w}(i-1)] \tag{3–2}$$

or the smoothed Newton's recursion to increase convergence speed:

$$\mathbf{w}(0) = \text{initial guess}; \quad \mathbf{w}(i) = \mathbf{w}(i-1) + \eta(\mathbf{R}_{\mathbf{u}} + \varepsilon\mathbf{I})^{-1}[\mathbf{r}_{d\mathbf{u}} - \mathbf{R}_{\mathbf{u}}\mathbf{w}(i-1)] \tag{3–3}$$

where ε is a small positive smoothing factor to prevent divide-by-zero and η is the step-size parameter specified by the designer.

Stochastic-gradient algorithms replace the covariance matrix and the cross-covariance vector at each iteration by local data approximations. There are several ways for obtaining such approximations, the trade-off being computation complexity, convergence performance, and steady-state behavior. Assume that we have access to observations of the random variables d and $\mathbf{u}$ over time

$$\{d(1), d(2), \dots\} \quad \text{and} \quad \{\mathbf{u}(1), \mathbf{u}(2), \dots\}$$

The least-mean-square (LMS) algorithm simply uses the instantaneous values to approximate $\hat{\mathbf{R}}_{\mathbf{u}} = \mathbf{u}(i)\mathbf{u}(i)^T$ and $\hat{\mathbf{r}}_{d\mathbf{u}} = d(i)\mathbf{u}(i)$. The corresponding steepest-descent

recursion (3–2) and Newton's recursion (3–3) become

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \eta \mathbf{u}(i)[d(i) - \mathbf{u}(i)^T \mathbf{w}(i-1)] \tag{3–4}$$

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \eta \mathbf{u}(i)[\mathbf{u}(i)^T \mathbf{u}(i) + \varepsilon \mathbf{I}]^{-1}[d(i) - \mathbf{u}(i)^T \mathbf{w}(i-1)] \tag{3–5}$$

The affine projection algorithm however employs better approximations. Specifically, $\mathbf{R}_{\mathbf{u}}$ and $\mathbf{r}_{d\mathbf{u}}$ are replaced by the approximations from the K most recent inputs and observations. Denoting

$$\mathbf{U}(i) = [\mathbf{u}(i-K+1), ..., \mathbf{u}(i)]_{L \times K} \quad \text{and} \quad \mathbf{d}(i) = [d(i-K+1), \ldots, d(i)]^T$$

we have

$$\begin{aligned} \hat{\mathbf{R}}_{\mathbf{u}} &= \frac{1}{K} \mathbf{U}(i) \mathbf{U}(i)^T \\ \hat{\mathbf{r}}_{d\mathbf{u}} &= \frac{1}{K} \mathbf{U}(i) \mathbf{d}(i) \end{aligned} \tag{3–6}$$

Therefore (3–2) and (3–3) become

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \eta \mathbf{U}(i)[\mathbf{d}(i) - \mathbf{U}(i)^T \mathbf{w}(i-1)] \tag{3–7}$$

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \eta [\mathbf{U}(i) \mathbf{U}(i)^T + \varepsilon \mathbf{I}]^{-1} \mathbf{U}(i)[\mathbf{d}(i) - \mathbf{U}(i)^T \mathbf{w}(i-1)] \tag{3–8}$$

Notice that

$$[\mathbf{U}(i) \mathbf{U}(i)^T + \varepsilon \mathbf{I}]^{-1} \mathbf{U}(i) = \mathbf{U}(i)[\mathbf{U}(i)^T \mathbf{U}(i) + \varepsilon \mathbf{I}]^{-1}.$$

This equation can be established by the matrix inversion lemma

$$(\mathbf{A} + \mathbf{BCD})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1} \mathbf{B} (\mathbf{C}^{-1} + \mathbf{D} \mathbf{A}^{-1} \mathbf{B})^{-1} \mathbf{D} \mathbf{A}^{-1} \tag{3–9}$$

with the identifications

$$\varepsilon \mathbf{I} \to \mathbf{A}, \ \mathbf{U}(i) \to \mathbf{B}, \ \mathbf{I} \to \mathbf{C}, \ \mathbf{U}(i)^T \to \mathbf{D}$$

Therefore, equation (3–8) is equivalent to

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \eta \mathbf{U}(i)[\mathbf{U}(i)^T\mathbf{U}(i) + \varepsilon\mathbf{I}]^{-1}[\mathbf{d}(i) - \mathbf{U}(i)^T\mathbf{w}(i-1)] \tag{3–10}$$

It is noted that this equivalence deals with the matrix $[\mathbf{U}(i)^T\mathbf{U}(i) + \varepsilon\mathbf{I}]$ instead of $[\mathbf{U}(i)\mathbf{U}(i)^T + \varepsilon\mathbf{I}]$ and it plays a very important role in the derivation of kernel extensions. We call recursion (3–7) APA-1 and recursion (3–10) APA-2. In the classical adaptive filtering literature, the name, affine projection algorithm, is exclusively used for the recursion (3–10) while we use affine projection algorithms to refer to a family of similar algorithms[1].

In some circumstances, a regularized cost function is needed instead of (3–1). The regularized LS problem is

$$\min_{\mathbf{w}} \mathbf{E}|d - \mathbf{w}^T\mathbf{u}|^2 + \lambda||\mathbf{w}||^2 \tag{3–11}$$

where λ is the regularization parameter (do not confuse with the smoothing factor ε in Newton's recursion, which is introduced mainly to ensure numerical stability and is not directly related to the norm constraint implemented by λ). The gradient method for this new cost function becomes

$$\begin{aligned} \mathbf{w}(i) &= \mathbf{w}(i-1) + \eta[\mathbf{r}_{d\mathbf{u}} - (\lambda\mathbf{I} + \mathbf{R}_\mathbf{u})\mathbf{w}(i-1)] \\ &= (1-\eta\lambda)\mathbf{w}(i-1) + \eta[\mathbf{r}_{d\mathbf{u}} - \mathbf{R}_\mathbf{u}\mathbf{w}(i-1)] \end{aligned} \tag{3–12}$$

and the Newton's recursion with $\varepsilon = 0$ is

$$\begin{aligned} \mathbf{w}(i) &= \mathbf{w}(i-1) + \eta(\lambda\mathbf{I} + \mathbf{R}_\mathbf{u})^{-1}[\mathbf{r}_{d\mathbf{u}} - (\lambda\mathbf{I} + \mathbf{R}_\mathbf{u})\mathbf{w}(i-1)] \\ &= (1-\eta)\mathbf{w}(i-1) + \eta(\lambda\mathbf{I} + \mathbf{R}_\mathbf{u})^{-1}\mathbf{r}_{d\mathbf{u}} \end{aligned} \tag{3–13}$$

If the approximations (3–6) are used in (3–12) and (3–13), we have

$$\mathbf{w}(i) = (1-\eta\lambda)\mathbf{w}(i-1) + \eta\mathbf{U}(i)[\mathbf{d}(i) - \mathbf{U}(i)^T\mathbf{w}(i-1)] \tag{3–14}$$

and

$$\mathbf{w}(i) = (1-\eta)\mathbf{w}(i-1) + \eta[\lambda\mathbf{I} + \mathbf{U}(i)\mathbf{U}(i)^T]^{-1}\mathbf{U}(i)\mathbf{d}(i) \tag{3–15}$$

The latter, by the matrix inversion lemma is equivalent to

$$\mathbf{w}(i) = (1-\eta)\mathbf{w}(i-1) + \eta\mathbf{U}(i)[\lambda\mathbf{I} + \mathbf{U}(i)^T\mathbf{U}(i)]^{-1}\mathbf{d}(i) \tag{3–16}$$

For simplicity, recursions (3–14) and (3–16) are named here APA-3 and APA-4 respectively[2].

3.2 Kernel Affine Projection Algorithms

Following the KLMS approach, the Mercer theorem is utilized to transform the data $\mathbf{u}(i)$ into the feature space $\mathbb{F}$ as $\boldsymbol{\varphi}(\mathbf{u}(i))$ (denoted as $\boldsymbol{\varphi}(i)$). The affine projection algorithms are formulated on the example sequence $\{d(1), d(2), \dots\}$ and $\{\boldsymbol{\varphi}(1), \boldsymbol{\varphi}(2), \dots\}$ to estimate the weight vector $\boldsymbol{\omega}$ that solves

$$\min_{\boldsymbol{\omega}} E|d - \boldsymbol{\omega}^T\boldsymbol{\varphi}(\mathbf{u})|^2 \tag{3–17}$$

By straightforward manipulation, the stochastic gradient descent (3–7) becomes

$$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)[\mathbf{d}(i) - \boldsymbol{\Phi}(i)^T\boldsymbol{\omega}(i-1)] \tag{3–18}$$

and stochastic Newton's method (3–10) becomes

$$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)[\boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i) + \varepsilon\mathbf{I}]^{-1}[\mathbf{d}(i) - \boldsymbol{\Phi}(i)^T\boldsymbol{\omega}(i-1)] \tag{3–19}$$

where $\boldsymbol{\Phi}(i) = [\boldsymbol{\varphi}(i-K+1), ..., \boldsymbol{\varphi}(i)]$.

Likewise, if the regularized cost function is specified, (3–14) for the stochastic gradient descent becomes

$$\boldsymbol{\omega}(i) = (1-\lambda\eta)\boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)[\mathbf{d}(i) - \boldsymbol{\Phi}(i)^T\boldsymbol{\omega}(i-1)] \tag{3–20}$$

and the corresponding Newton's method (3–16) becomes

$$\boldsymbol{\omega}(i) = (1-\eta)\boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)[\boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i) + \lambda\mathbf{I}]^{-1}\mathbf{d}(i) \tag{3–21}$$

For simplicity, we refer to the recursions (3–18), (3–19), (3–20), and (3–21) as KAPA-1, KAPA-2, KAPA-3, and KAPA-4 respectively. Each will be treated independently in the sequel.

3.2.1 KAPA-1 (Simple KAPA)

Recursion (3–18) uses the stochastic gradient descent and is the simplest among all. It is hence also named simple KAPA here. The same methodology for KLMS is used to rewrite (3–18) as a sum of errors multiplied by the transformed inputs. If we set the initial guess $\boldsymbol{\omega}(0) = 0$, the iteration of (3–18) will be

$$\begin{aligned}
&\boldsymbol{\omega}(0) = 0,\\
&\boldsymbol{\omega}(1) = \eta d(1)\boldsymbol{\varphi}(1) = \mathbf{a}_1(1)\boldsymbol{\varphi}(1),\\
&...\\
&\boldsymbol{\omega}(i-1) = \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\boldsymbol{\varphi}(j),\\
&\boldsymbol{\Phi}(i)^T\boldsymbol{\omega}(i-1) = [\sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{i-K+1,j}, , ..., \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{i-1,j}, \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{i,j}]^T,\\
&\mathbf{e}(i) = \mathbf{d}(i) - \boldsymbol{\Phi}(i)^T\boldsymbol{\omega}(i-1),\\
&\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)\mathbf{e}(i) = \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\boldsymbol{\varphi}(j) + \sum_{j=1}^{K} \eta\mathbf{e}_j(i)\boldsymbol{\varphi}(i-j+K).
\end{aligned} \tag{3–22}$$

where $\kappa_{i,j} = \kappa(\mathbf{u}(i), \mathbf{u}(j))$ for simplicity.

Note that during the iteration, the weight vector in the feature space assumes the following expansion

$$\boldsymbol{\omega}(i) = \sum_{j=1}^{i} \mathbf{a}_j(i)\boldsymbol{\varphi}(j) \quad \forall\, i > 0 \tag{3–23}$$

i.e. the weight at time i is a linear combination of the previous transformed input. This result may seem simply a restatement of the representer theorem in [Schölkopf et al., 2001]. However, it should be emphasized that this result does not rely on any explicit minimal norm constraint as required for the representer theorem. As we discussed in

Chapter 2, the gradient search has an inherent regularization mechanism which guarantees the solution is in the data subspace under appropriate initialization. In general, the initialization $\boldsymbol{\omega}(0)$ can alternatively translate whatever a priori information is available as long as it can be expressed as a linear combination of transformed data in order to utilize the kernel trick, but the solution loses its minimum norm property. By (3–23), the weight vector updating is accomplished through the expansion coefficients

$$\mathbf{a}_k(i) = \begin{cases} \eta(d(i) - \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{i,j}), & k = i \\ \mathbf{a}_k(i-1) + \eta(d(k) - \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{k,j}), & i-K+1 \le k \le i-1 \\ \mathbf{a}_k(i-1), & 1 \le k < i-K+1 \end{cases} \tag{3–24}$$

Let us introduce a simplified notation $e(i;k) = \mathbf{e}_{K+k-i}(i) = d(k) - \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{k,j}$ indicating the prediction error on data $\{\mathbf{u}(k), d(k)\}$ using $\boldsymbol{\omega}(i-1)$. The interpretation of (3–24) becomes straightforward: allocate a new unit with coefficient $\eta e(i;i)$ and update the coefficients for the other $K-1$ most recent units by $\eta e(i;k)$ for $i-K+1 \le k \le i-1$.

If we denote f_i as the estimate of the input-output mapping at time i, we have the following sequential learning rule for KAPA-1:

$$f_i = f_{i-1} + \eta \sum_{j=i-K+1}^{i} e(i;j)\kappa(\mathbf{u}(j), \cdot) \tag{3–25}$$

The coefficients $\mathbf{a}(i)$ and the centers $\mathcal{C}(i)$ should be stored in the computer during training. The updates needed for KAPA-1 at time i is

$$\begin{aligned} \mathbf{a}_i(i) &= \eta e(i;i) \\ \mathbf{a}_j(i) &= \mathbf{a}_j(i-1) + \eta e(i;j), \quad j = i-K+1, ..., i-1 \\ \mathbf{a}_j(i) &= \mathbf{a}_j(i-1), \quad j = 1, ..., i-K \\ \mathcal{C}(i) &= \{\mathcal{C}(i-1), \mathbf{u}(i)\} \end{aligned} \tag{3–26}$$

The pseudocode for KAPA-1 is listed in Algorithm 4.

Algorithm 4 The Kernel Affine Projection Algorithm–Type 1 (KAPA-1)

Initialization
step-size parameter η
$\mathbf{a}_1(1) = \eta d(1)$
Computation
while $\{\mathbf{u}(i), d(i)\}$ available **do**
  %allocate a new unit
  $\mathbf{a}_i(i-1) = 0$
  for $k = \max(1, i-K+1)$ to i **do**
    %evaluate outputs of the current network
    $y(i;k) = \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{k,j}$
    %computer errors
    $e(i;k) = d(k) - y(i;k)$
    %update the $\min(i, K)$ most recent units
    $\mathbf{a}_k(i) = \mathbf{a}_k(i-1) + \eta e(i;k)$
  end for
  if $i > K$ **then**
    %keep the remaining
    for $k = 1$ to $i-K$ **do**
      $\mathbf{a}_k(i) = \mathbf{a}_k(i-1)$
    end for
  end if
end while

At iteration i, given a test point input $\mathbf{u}_*$, the system output is computed as

$$f(\mathbf{u}_*) = \sum_{j=1}^{i} \mathbf{a}_j(i)\kappa(\mathbf{u}(j), \mathbf{u}_*).$$

3.2.2 KAPA-2 (Normalized KAPA)

Similarly, the smoothed Newton's recursion (3–19) can be factorized into the following steps

$$\begin{aligned}
\boldsymbol{\omega}(i-1) &= \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\boldsymbol{\varphi}(j), \\
\mathbf{e}(i) &= \mathbf{d}(i) - \boldsymbol{\Phi}(i)^T\boldsymbol{\omega}(i-1), \\
\mathbf{G}(i) &= \boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i), \\
\boldsymbol{\omega}(i) &= \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)[\mathbf{G}(i) + \varepsilon\mathbf{I}]^{-1}\mathbf{e}(i).
\end{aligned} \tag{3–27}$$

In practice, we do not have access to the transformed weight $\boldsymbol{\omega}$ or any transformed data, so the update has to be evaluated through the expansion coefficient $\mathbf{a}$, just like in KAPA-1. The whole recursion is similar to (3–24) except that the error is normalized by a $K \times K$ matrix $[\mathbf{G}(i) + \varepsilon \mathbf{I}]^{-1}$.

3.2.3 KAPA-3 (Leaky KAPA)

When the cost function (3–17) is ill-posed in the conventional empirical risk minimization (ERM) sense [Girosi et al., 1995], the common practice is to constrain the solution norm:

$$\min_{\boldsymbol{\omega}} \mathbf{E}|d - \boldsymbol{\omega}^T \boldsymbol{\varphi}(\mathbf{u})|^2 + \lambda ||\boldsymbol{\omega}||^2 \tag{3–28}$$

As we have already shown in (3–20), the leaky KAPA is

$$\boldsymbol{\omega}(i) = (1 - \lambda\eta)\boldsymbol{\omega}(i-1) + \eta \boldsymbol{\Phi}(i)[\mathbf{d}(i) - \boldsymbol{\Phi}(i)^T \boldsymbol{\omega}(i-1)] \tag{3–29}$$

Again, the iteration will be based on the expansion coefficient $\mathbf{a}$, which is similar to (3–24).

$$\mathbf{a}_k(i) = \begin{cases} \eta(d(i) - \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{i,j}), & k = i \\ (1 - \lambda\eta)\mathbf{a}_k(i-1) + \eta(d(k) - \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{k,j}), & i - K + 1 \le k \le i - 1 \\ (1 - \lambda\eta)\mathbf{a}_k(i-1), & 1 \le k < i - K + 1 \end{cases} \tag{3–30}$$

The only difference with respect to KAPA-1 is that KAPA-3 has a scaling factor $(1 - \lambda\eta)$ multiplying the previous weight, which is less than 1, and it imposes a forgetting mechanism so that the training data in the far past are scaled down exponentially. Furthermore since the network size is growing over training, a transformed data can be pruned from the expansion easily if its coefficient is smaller than some pre-specified threshold.

3.2.4 KAPA-4 (Leaky KAPA with Newton's recursion)

As before, KAPA-4 (3–21) reduces to

$$\mathbf{a}_k(i) = \begin{cases} \eta\tilde{d}(i), & k = i \\ (1-\eta)\mathbf{a}_k(i-1) + \eta\tilde{d}(k), & i-K+1 \le k \le i-1 \\ (1-\eta)\mathbf{a}_k(i-1), & 1 \le k < i-K+1 \end{cases} \tag{3–31}$$

where $\tilde{\mathbf{d}}(i) = (\mathbf{G}(i) + \lambda\mathbf{I})^{-1}\mathbf{d}(i)$.

Among these four algorithms, the first three require the error information to update the network which is computationally expensive, however KAPA-4 does not. Therefore the different update rule in KAPA-4 has a huge significance in terms of computation since it only needs a $K \times K$ matrix inversion, which by using the sliding-window trick only requires $O(K^2)$ operations [Van Vaerenbergh et al., 2006]. We summarize the four KAPA update equations in Table 3-1 for ease of comparison.

Table 3-1. Comparison of KAPA update rules

Algorithm	Update equation
KAPA-1	$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)[\mathbf{d}(i) - \boldsymbol{\Phi}(i)^T\boldsymbol{\omega}(i-1)]$
KAPA-2	$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)[\boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i) + \varepsilon\mathbf{I}]^{-1}[\mathbf{d}(i) - \boldsymbol{\Phi}(i)^T\boldsymbol{\omega}(i-1)]$
KAPA-3	$\boldsymbol{\omega}(i) = (1-\lambda\eta)\boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)[\mathbf{d}(i) - \boldsymbol{\Phi}(i)^T\boldsymbol{\omega}(i-1)]$
KAPA-4	$\boldsymbol{\omega}(i) = (1-\eta)\boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)[\boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i) + \lambda\mathbf{I}]^{-1}\mathbf{d}(i)$

3.3 Error Reusing

As we see in KAPA-1, KAPA-2 and KAPA-3, the most time-consuming part of the computation is to calculate the prediction errors. For example, suppose $\boldsymbol{\omega}(i-1) = \sum_{j=1}^{i-1}\mathbf{a}_j(i-1)\boldsymbol{\varphi}(j)$. We need to calculate

$$e(i;k) = d(k) - \boldsymbol{\omega}(i-1)^T\boldsymbol{\varphi}(k)$$

for $i-K+1 \le k \le i$ to compute $\boldsymbol{\omega}(i)$, which consists of $(i-1)K$ kernel evaluations. As i increases, this dominates the computation time. In this sense, the computation complexity of KAPA is K times of KLMS. However, after careful manipulation, we can shrink the complexity gap between KAPA and KLMS by reusing the errors.

Assume that all the K errors

$$e(i-1;k) = d(k) - \boldsymbol{\omega}(i-2)^T \boldsymbol{\varphi}(k)$$

for $i-K \le k \le i-1$ are stored from the previous iteration. At the present iteration, we have

$$\begin{aligned} e(i;k) &= d(k) - \boldsymbol{\varphi}(k)^T \boldsymbol{\omega}(i-1) \\ &= d(k) - \boldsymbol{\varphi}(k)^T [\boldsymbol{\omega}(i-2) + \eta \sum_{j=i-K}^{i-1} e(i-1;j)\boldsymbol{\varphi}(j)] \\ &= [d(k) - \boldsymbol{\varphi}(k)^T \boldsymbol{\omega}(i-2)] + \eta \sum_{j=i-K}^{i-1} e(i-1;j)\kappa_{j,k} \\ &= e(i-1;k) + \sum_{j=i-K}^{i-1} \eta e(i-1;j)\kappa_{j,k} \end{aligned} \tag{3–32}$$

Note that $e(i-1;k), k < i$ have all been previously computed. Therefore, the only term that is not available is $e(i-1;i)$ which requires $i-1$ times kernel evaluations. Overall the computation complexity of KAPA-1 is $O(i+K^2)$, which is only $O(K^2)$ more than KLMS.

3.4 Sliding Window Gram Matrix Inversion

In KAPA-2 and KAPA-4, another computation difficulty is to invert a $K \times K$ matrix, which normally requires $O(K^3)$ operations. However, in KAPA, the data matrix $\boldsymbol{\Phi}(i)$ has a sliding window structure, therefore a trick can be used to speed up the computation. The trick is based on the matrix inversion formula and was introduced in [Van Vaerenbergh et al., 2006]. We outline the basic calculation steps here. Suppose the sliding matrices share the same sub-matrix $\mathbf{D}$

$$\mathbf{G}(i-1) + \lambda \mathbf{I} = \begin{bmatrix} a & \mathbf{b}^T \\ \mathbf{b} & \mathbf{D} \end{bmatrix}, \quad \mathbf{G}(i) + \lambda \mathbf{I} = \begin{bmatrix} \mathbf{D} & \mathbf{h} \\ \mathbf{h}^T & g \end{bmatrix} \tag{3–33}$$

and we know from the previous iteration

$$(\mathbf{G}(i-1)+\lambda\mathbf{I})^{-1}=\begin{bmatrix} e & \mathbf{f}^T \\ \mathbf{f} & \mathbf{H} \end{bmatrix} \tag{3–34}$$

First, calculate the inverse of $\mathbf{D}$ as

$$\mathbf{D}^{-1}=\mathbf{H}-\mathbf{f}\mathbf{f}^T/e \tag{3–35}$$

Then, update the inverse of the new Gram matrix as

$$(\mathbf{G}(i)+\lambda\mathbf{I})^{-1}=\begin{bmatrix} \mathbf{D}^{-1}+(\mathbf{D}^{-1}\mathbf{h})(\mathbf{D}^{-1}\mathbf{h})^T s^{-1} & -(\mathbf{D}^{-1}\mathbf{h})s^{-1} \\ -(\mathbf{D}^{-1}\mathbf{h})^T s^{-1} & s^{-1} \end{bmatrix} \tag{3–36}$$

with $s=g-\mathbf{h}^T\mathbf{D}^{-1}\mathbf{h}$. The overall complexity is $O(K^2)$.

3.5 Taxonomy for Related Algorithms

KLMS Algorithm

If $K=1$, KAPA-1 reduces to the following kernel least-mean-square algorithm (KLMS)

$$\boldsymbol{\omega}(i)=\boldsymbol{\omega}(i-1)+\eta\boldsymbol{\varphi}(i)[d(i)-\boldsymbol{\varphi}(i)^T\boldsymbol{\omega}(i-1)]$$

It is seen that the KLMS allocates a new unit when a new training data comes in with the input $\mathbf{u}(i)$ as the center and the prediction error as the coefficient (scaled by the step-size parameter). In other words, once the unit is allocated, the coefficient is fixed. It mimics the resource-allocating step in the RAN algorithm whereas it neglects the adaptation step. In this sense, the KAPA algorithms that allocate a new unit for the present input and also adapt the other $K-1$ most recent allocated units, are closer to the original RAN.

Similarly the normalized KLMS algorithm is a special case of KAPA-2 with $K=1$:

$$\boldsymbol{\omega}(i)=\boldsymbol{\omega}(i-1)+\frac{\eta\boldsymbol{\varphi}(i)}{\varepsilon+\kappa(\mathbf{u}(i),\mathbf{u}(i))}[d(i)-\boldsymbol{\varphi}(i)^T\boldsymbol{\omega}(i-1)] \tag{3–37}$$

Notice that for translation invariant kernels, i.e., $\kappa(\mathbf{u}(i), \mathbf{u}(i)) = const$, KLMS is automatically normalized. Sometimes we use KLMS-1 and KLMS-2 to distinguish the two.

NORMA Algorithm

Similarly KAPA-3 (3–20) reduces to the NORMA algorithm introduced by Kivinen et al. [2004].

$$\boldsymbol{\omega}(i) = (1 - \eta\lambda)\boldsymbol{\omega}(i-1) + \eta\boldsymbol{\varphi}(i)[d(i) - \boldsymbol{\varphi}(i)^T\boldsymbol{\omega}(i-1)] \tag{3–38}$$

As we discussed in Chapter 2, penalizing explicitly the solution norm introduces a bias and significantly degenerates the overall performance, so in general we do not recommend the use of KAPA-3.

Kernel ADALINE

Assume that the size of the training data is finite N. If we set $K = N$, then the update rule of KAPA-1 becomes

$$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}[\mathbf{d} - \boldsymbol{\Phi}^T\boldsymbol{\omega}(i-1)]$$

where the full data matrices are

$$\boldsymbol{\Phi} = [\boldsymbol{\varphi}(1), ..., \boldsymbol{\varphi}(N)], \quad \mathbf{d} = [d(1), ..., d(N)]$$

It is easy to check that the weight vector also assumes the following expansion

$$\boldsymbol{\omega}(i) = \sum_{j=1}^{N} \mathbf{a}_j(i)\boldsymbol{\varphi}(j)$$

And the updating on the expansion coefficients is

$$\mathbf{a}_j(i) = \mathbf{a}_j(i-1) + \eta[d(j) - \boldsymbol{\varphi}(j)^T\boldsymbol{\omega}(i-1)]$$

This is nothing but the kernel ADALINE (KA) introduced by T.-T.Frieb and Harrison [1999]. Notice that the kernel ADALINE is not an online method.

Sliding Window Kernel Recursive Least Squares

In KAPA-4, if we set $\eta = 1$, we have

$$\boldsymbol{\omega}(i) = \boldsymbol{\Phi}(i)[\boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i) + \lambda\mathbf{I}]^{-1}\mathbf{d}(i) \tag{3–39}$$

which is the sliding-window kernel RLS (SW-KRLS) introduced by Van Vaerenbergh et al. [2006].

Regularization Networks

We assume there are only N training data and $K = N$. Equation (3–21) becomes directly

$$\boldsymbol{\omega}(i) = \boldsymbol{\Phi}[\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\mathbf{I}]^{-1}\mathbf{d} \tag{3–40}$$

which is the regularization network (RegNet) [Girosi et al., 1995].

We summarize all the related algorithms in Table 3-2 for convenience.

Table 3-2. List of algorithms related to KAPA

Algorithm	Update equation	Relation to KAPA
KLMS	$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\varphi}(i)[d(i) - \boldsymbol{\varphi}(i)^T\boldsymbol{\omega}(i-1)]$	KAPA-1, $K = 1$
NKLMS	$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \frac{\eta\boldsymbol{\varphi}(i)}{(\varepsilon+\kappa_{i,i})}[d(i) - \boldsymbol{\varphi}(i)^T\boldsymbol{\omega}(i-1)]$	KAPA-2, $K = 1$
NORMA	$\boldsymbol{\omega}(i) = (1-\eta\lambda)\boldsymbol{\omega}(i-1) + \eta\boldsymbol{\varphi}(i)[d(i) - \boldsymbol{\varphi}(i)^T\boldsymbol{\omega}(i-1)]$	KAPA-3, $K = 1$
KA	$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}[\mathbf{d} - \boldsymbol{\Phi}^T\boldsymbol{\omega}(i-1)]$	KAPA-1, $K = N$
SW-KRLS	$\boldsymbol{\omega}(i) = \boldsymbol{\Phi}(i)[\boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i) + \lambda\mathbf{I}]^{-1}\mathbf{d}(i)$	KAPA-4, $\eta = 1$
RegNet	$\boldsymbol{\omega}(i) = \boldsymbol{\Phi}[\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\mathbf{I}]^{-1}\mathbf{d}$	KAPA-4, $\eta = 1$, $K = N$

3.6 Computer Experiments

3.6.1 KAPA Applied to Mackey-Glass Time Series Prediction

This example is a further study on the short-term prediction of the Mackey-Glass (MG) chaotic time series discussed in Chapter 2. We set the time embedding as 7 here, i.e. $\mathbf{u}(i) = [x(i-7), x(i-6), ..., x(i-1)]^T$ are used as the input to predict the present one $x(i)$.

PART 1: A segment of 500 samples is used as the training data and another 100 points as the test data (in the testing phase, the filter is fixed). All the data are corrupted by Gaussian noise with zero mean and 0.001 variance.

We compare the prediction performance of KLMS, KAPA-1, KAPA-2, KRLS, and a linear combiner trained with LMS. KRLS will be fully discussed in the next chapter of the book and it is only presented here for comparison. The Gaussian kernel (1–24) with kernel parameter $a = 1$ is chosen for all the kernel-based algorithms. Figure 3-1 is a typical plot of the learning curves for the LMS, KLMS-1, KAPA-1, KAPA-2 ($K = 10$) and KRLS respectively. The last 100 points of the learning curves are used to compute the results listed in Table 3-3, where the parameters of each algorithm are also listed.

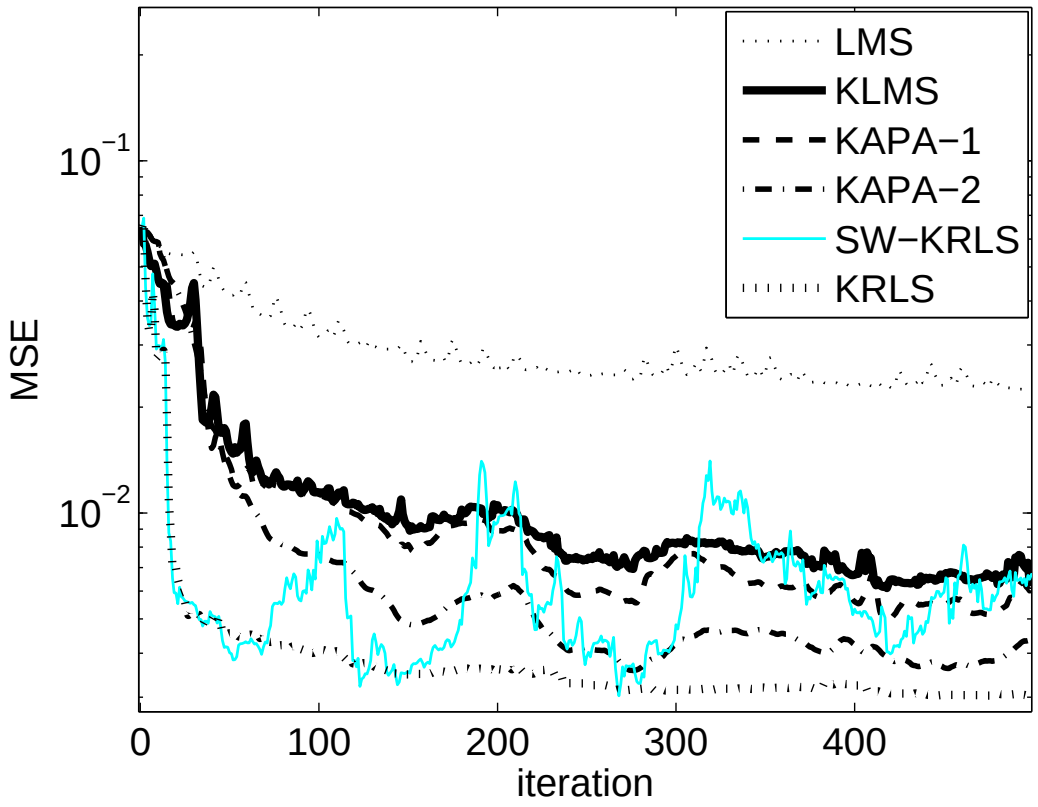

Figure 3-1. Learning curves of LMS, KLMS, KAPA-1 ($K = 10$), KAPA-2 ($K = 10$), SW-KRLS ($K = 50$) and KRLS in Mackey-Glass time series prediction

As we can see in Table 3-3, the performance of KAPA-2 is substantially better than KLMS. All the results in the tables are in the form of "average ± standard deviation". Table 3-4 summarizes the computational complexity of these algorithms. KLMS and KAPA effectively reduce the computational complexity and memory storage when

Table 3-3. Performance comparison of LMS, KLMS, KAPA, SW-KRLS and KRLS in Mackey-Glass time series prediction

Algorithm	Parameters	Test Mean Square Error
LMS	$\eta = 0.04$	0.0208 ± 0.0009
KLMS	$\eta = 0.02$	0.0052 ± 0.00022
SW-KRLS	$K = 50$, $\lambda = 0.1$	0.0052 ± 0.00026
KAPA-1	$\eta = 0.03$, $K = 10$	0.0048 ± 0.00023
KAPA-2	$\eta = 0.03$, $K = 10$, $\varepsilon = 0.1$	0.0040 ± 0.00028
KRLS	$\lambda = 0.1$	0.0027 ± 0.00009

Table 3-4. Complexity comparison of LMS, KLMS, KAPA, SW-KRLS and KRLS at iteration i

Algorithm	Computation	Memory
LMS	$O(L)$	$O(L)$
KLMS	$O(i)$	$O(i)$
SW-KRLS	$O(K^2)$	$O(K^2)$
KAPA-1	$O(i + K^2)$	$O(i + K)$
KAPA-2	$O(i + K^2)$	$O(i + K^2)$
KAPA-4	$O(K^2)$	$O(i + K^2)$
KRLS	$O(i^2)$	$O(i^2)$

compared with KRLS. KAPA-3 and SW-KRLS are also tested on this problem. It is observed that the performance of KAPA-3 is similar to KAPA-1 when the forgetting term is very close to 1 as expected and the results are severely biased when the forgetting term is reduced further. The performance of SW-KRLS is included in Figure 3-1 and in Table 3-3 with $K = 50$. It is observed that KAPA-4 (including SW-KRLS) does not perform well with small K (< 50).

PART 2: We test how the novelty criterion affects the performance of KAPA algorithms. A segment of 1500 samples is used as the training data and another 200 as the test data. All the data are corrupted by Gaussian noise with zero mean and 0.0001 variance. The thresholds in the novelty criterion are set as $\delta_1 = 0.1$ and $\delta_2 = 0.05$. The learning curves are shown in Figure 3-2 and the results are summarized in Table 3-5, which is calculated from the last 100 points of the learning curves. It is seen that the complexity can be reduced dramatically with the novelty criterion preserving the prediction accuracy.

Here we use "-NC" to indicate the corresponding algorithms equipped with the novelty criterion.

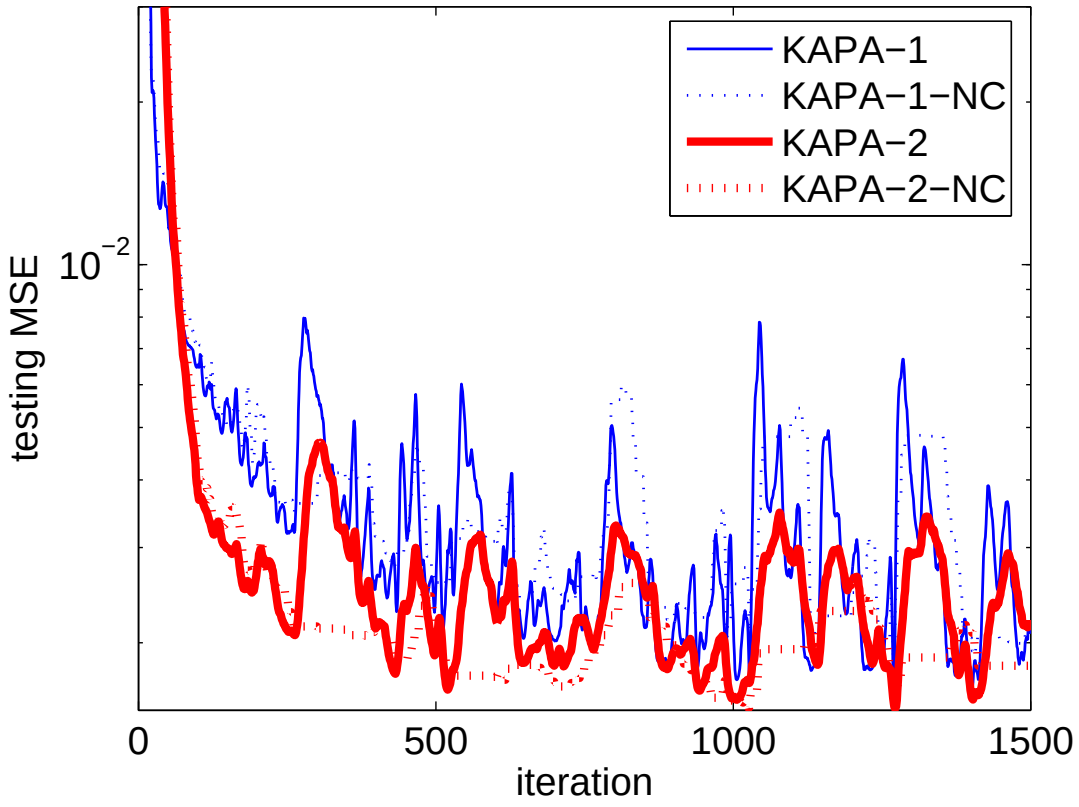

Figure 3-2. Learning curves of KAPA-1 ($K = 10$) and KAPA-2 ($K = 10$) with and without novelty criterion in Mackey-Glass time series prediction

Table 3-5. Performance of KAPA with novelty criterion in Mackey-Glass time series prediction

Algorithm	Parameters	Test Mean Square Error	Dictionary size
KAPA-1	$\eta = 0.05$	0.0026 ± 0.00069	1000
KAPA-1-NC	$\eta = 0.05$	0.0020 ± 0.00004	395
KAPA-2	$\eta = 0.05$, $\varepsilon = 0.1$	0.0022 ± 0.00041	1000
KAPA-2-NC	$\eta = 0.05$, $\varepsilon = 0.1$	0.0018 ± 0.00007	336

3.6.2 KAPA Applied to Noise Cancellation

Another important problem in signal processing is noise cancellation in which an unknown interference has to be removed based on some reference measurement[3]. The basic structure of a noise cancellation system is shown in Figure 3-3. The primary signal

is $s(i)$ and its noisy measurement $d(i)$ acts as the desired signal of the system. $n(i)$ is a white noise process which is unknown, and $u(i)$ is its reference measurement, i.e. a distorted version of the noise process through some distortion function, which is unknown in general. Here $u(i)$ is the input of the adaptive filter. The objective is to use $u(i)$ as the input to the filter and to obtain as the filter output an estimate of the noise source $n(i)$. Therefore, the noise can be subtracted from $d(i)$ to improve the signal-noise-ratio.

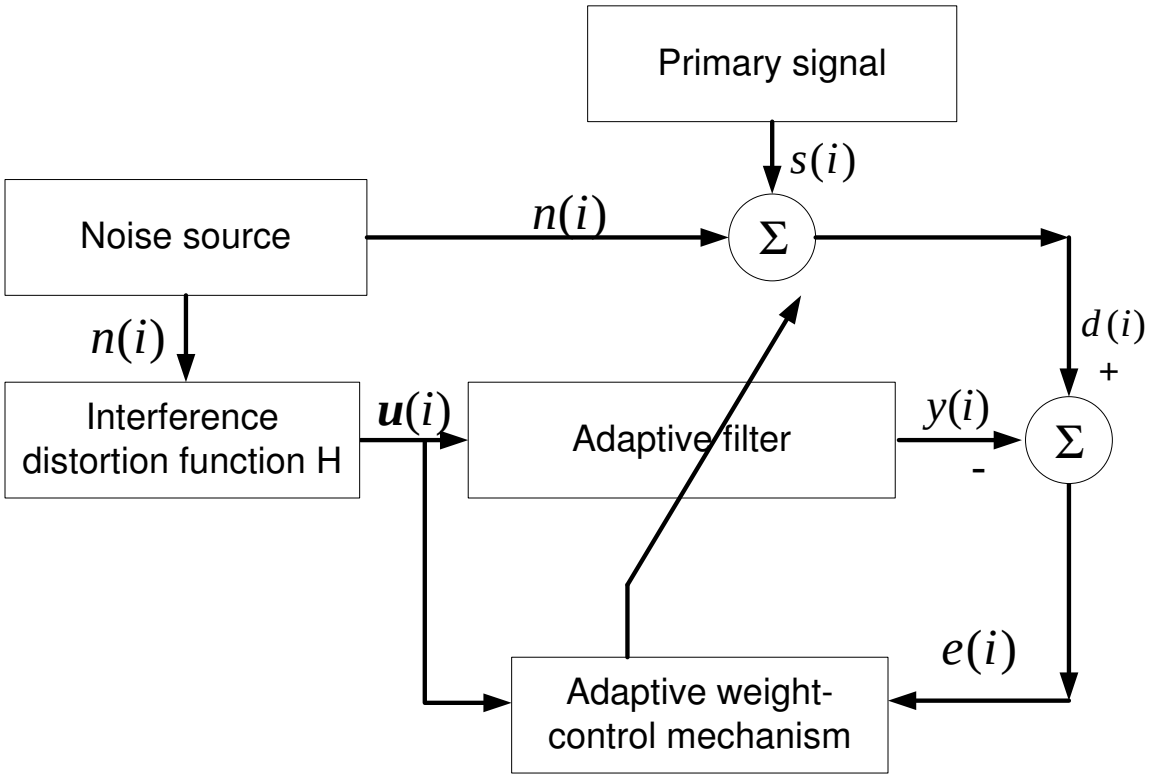

Figure 3-3. Basic structure of a noise cancellation system

PART 1: In this example, the noise source is assumed white, uniformly distributed between $[-0.5, 0.5]$. The interference distortion function is assumed to be

$$u(i) = n(i) - 0.2u(i-1) - u(i-1)n(i-1) + 0.1n(i-1) + 0.4u(i-2) \qquad (3\text{–}41)$$

As we see, the distortion function is nonlinear (multiplicative) and has infinite impulsive response, which on the other hand, means it is impossible to recover $n(i)$ from a

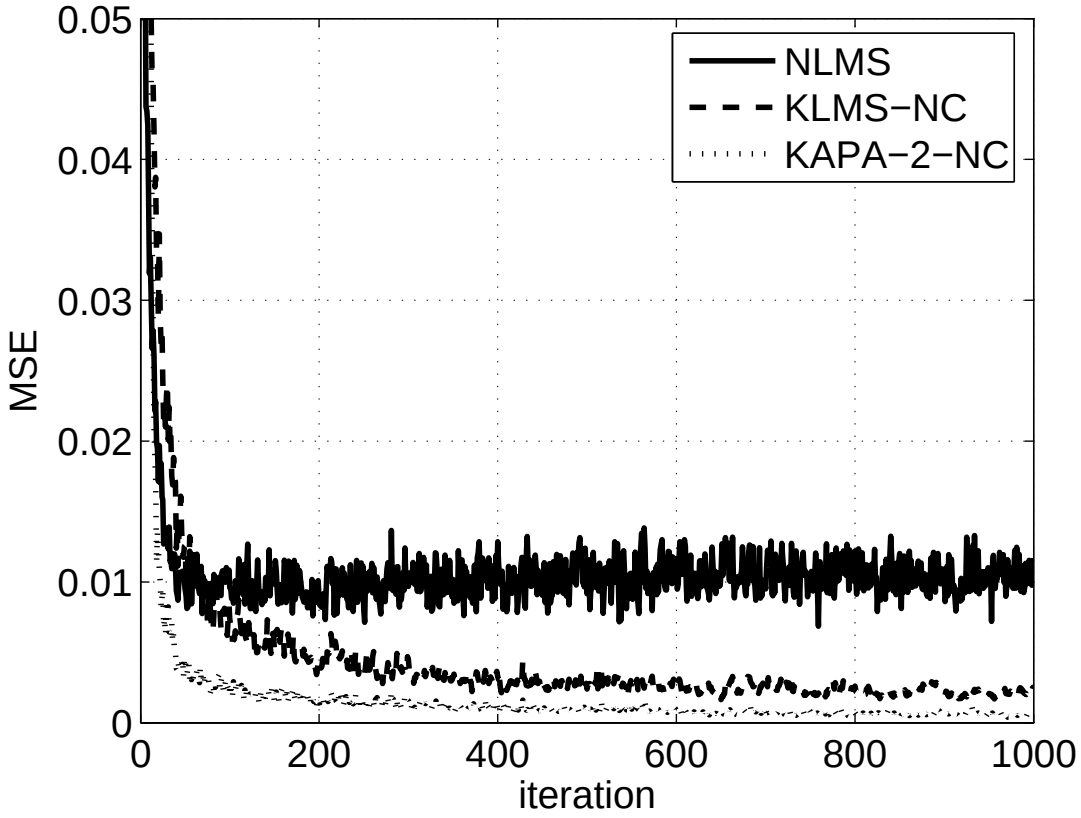

Figure 3-4. Ensemble learning curves of NLMS, KLMS-NC and KAPA-2-NC ($K = 10$) in noise cancellation

finite time delay embedding of $u(i)$. We rewrite the distortion function as

$$n(i) = u(i) + 0.2u(i-1) - 0.4u(i-2) + (u(i-1) - 0.1)n(i-1)$$

Therefore the present value of the noise source $n(i)$ not only depends on the reference noise measure $[u(i), u(i-1), u(i-2)]$, but it also depends on the previous value $n(i-1)$, which in turn depends on $[u(i-1), u(i-2), u(i-3)]$ and so on. It means we need a very long time embedding (infinitely long theoretically) in order to recover $n(i)$ accurately. However, the recursive nature of an adaptive system provides a feasible alternative, i.e. we feedback the output of the filter $\hat{n}(i-1)$, which is the estimate of $n(i-1)$ to estimate the present one, pretending $\hat{n}(i-1)$ is the true value of $n(i-1)$. Therefore, the input of the adaptive filter is of the form $[u(i), u(i-1), u(i-2), \hat{n}(i-1)]$. It can be seen

that the system is inherently recurrent. In the linear case, it belongs to the *output error methods* [Goodwin and Sin, 1984]. However, it will be non-trivial to generalize the results concerning convergence and stability to nonlinear cases and it serves as a line of future work.

We assume the primary signal $s(i) = 0$ during the training phase. And the system simply tries to reconstruct the noise source from the reference measurement. We use a linear filter trained with normalized LMS (NLMS), two nonlinear filters trained with KLMS-NC and KAPA-2-NC ($K = 10$) respectively. 1000 training samples are used and 200 Monte Carlo simulations are run to get the ensemble learning curves as shown in Figure 3-4. The step-size parameter and regularization parameter for NLMS is 0.2 and 0.005. The step-size parameter for KLMS-NC and KAPA-2-NC is 0.5 and 0.2 respectively. The Gaussian kernel is used for both KLMS-NC and KAPA-2-NC with kernel parameter $a = 1$. The tolerance parameters for KLMS-NC and KAPA-2-NC are $\delta_1 = 0.15$ and $\delta_2 = 0.01$. The noise reduction factor (NR), which is defined as $10\log_{10}\{E[n^2(i)]/E[n(i)-y(i)]^2\}$ is listed in Table 3-6 along with the corresponding network size (the final number of units). The performance improvement of KAPA-2-NC is obvious when compared with KLMS-NC.

Table 3-6. Performance comparison of NLMS, KLMS and KAPA-2 in noise cancellation

Algorithm	Network Size	NR(dB)
NLMS	N/A	9.09±0.45
KLMS-NC	407±14	15.58±0.48
KAPA-2-NC	370±14	21.99±0.80

PART 2: Next we use a more realistic noise source (instead of the white noise) which is a fMRI recording provided by Dr. Issa Panahi from University of Texas at Dallas. The mean of the fMRI noise is 0 and the standard deviation is 0.051. The typical waveform is shown in Figure 3-5. We compare KAPA-2-NC with NLMS. 200 Monte Carlo simulations are conducted using different segments of the recording. We average all the learning curves together to get the ensemble learning curves plotted in Figure 3-6.

The step-size parameter and regularization parameter for NLMS is 0.2 and 0.005. The step-size parameter for KAPA-2-NC is 0.2. The Gaussian kernel is used for KAPA-2-NC with kernel parameter $a = 1$. The tolerance parameters are $\delta_1 = 0$ and $\delta_2 = 0.001$. And the noise reduction factor (NR) is listed in Table 3-7 along with the corresponding network size (the final number of units). The performance improvement of KAPA-2-NC is quite significant when compared with NLMS.

Table 3-7. Performance comparison of NLMS and KAPA-2 using actual fMRI noise recording

Algorithm	Network Size	NR(dB)
NLMS	N/A	23.68±4.14
KAPA-2-NC	170±12	36.50±2.29

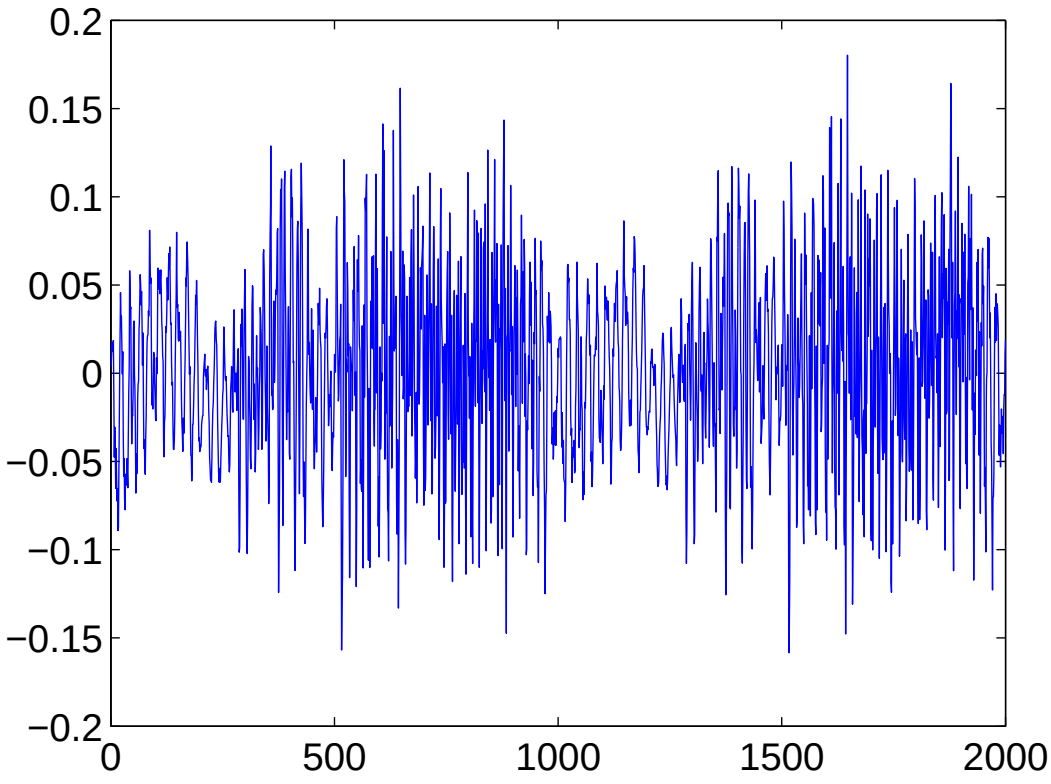

Figure 3-5. A typical segment of fMRI noise recording

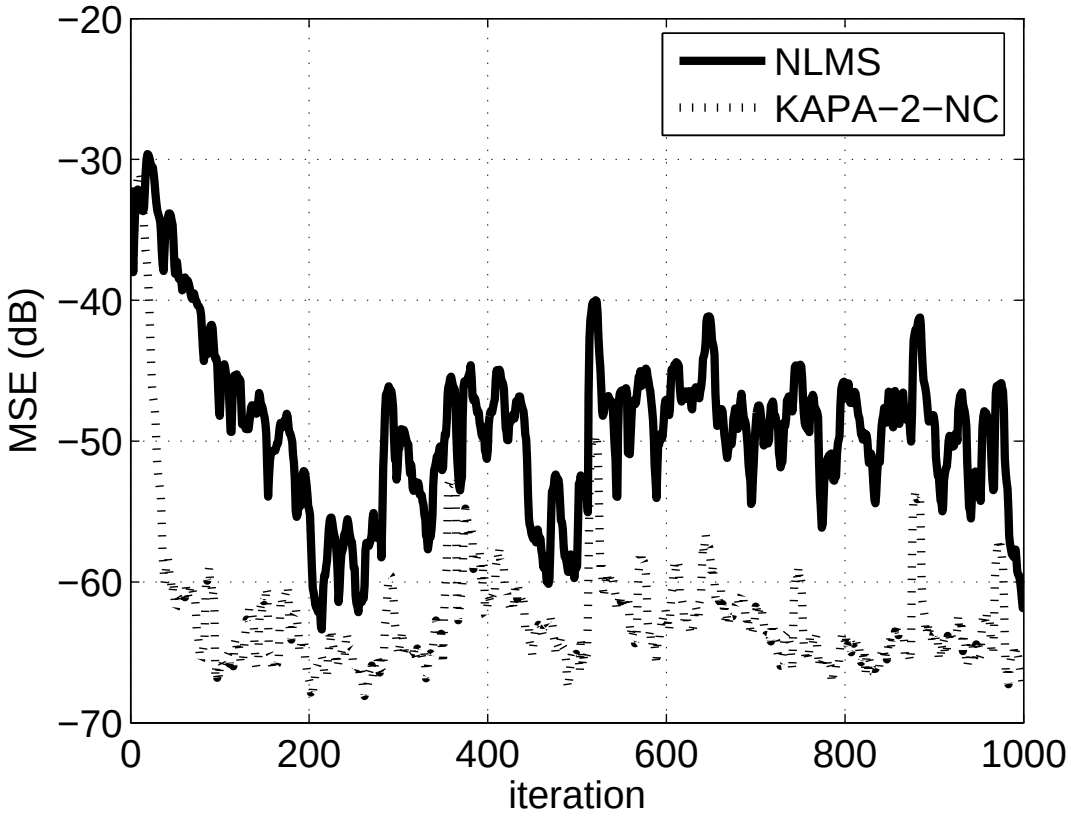

Figure 3-6. Ensemble learning curves of NLMS and KAPA-2-NC ($K = 10$) in fMRI noise cancellation

3.6.3 KAPA Applied to Nonlinear Channel Equalization

In this example, we reconsider the nonlinear channel equalization problem (See Figure 3-7). The problem setting is the same as in Chapter 2: A binary signal $\{s(1), s(2), ..., s(N)\}$ is fed into the nonlinear channel. At the receiver end of the channel, the signal is further corrupted by additive white Gaussian noise and is then observed as $\{r(1), r(2), ..., r(N)\}$. The aim of channel equalization is to construct an *inverse* filter that reproduces the original signal with as low an error rate as possible. It is easy to formulate it as a regression problem, with input-output examples $\{(r(i + D), r(i + D - 1), ..., r(i + D - l + 1)), s(i)\}$, where l is the time embedding length, and D is the equalization time lag. $l = 3$ and $D = 2$ in the equalizer.

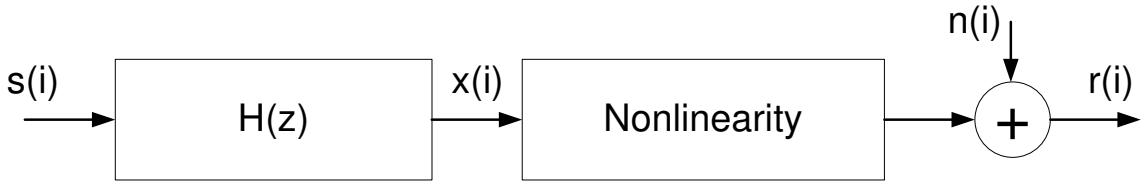

Figure 3-7. Basic structure of a nonlinear channel

PART 1: In this experiment, the nonlinear channel model is defined by $x(i) = s(i) + 0.5s(i-1)$, $r(i) = x(i) - 0.9x(i)^2 + n(i)$, where $n(i)$ is the white Gaussian noise with a variance of σ^2. We compare the performance of LMS, APA-1, KLMS-NC, KAPA-1-NC ($K = 10$), and KAPA-2-NC ($K = 10$). The Gaussian kernel with $a = 0.1$ is used in KLMS-NC, KAPA-1-NC and KAPA-2-NC. The noise variance is fixed here at $\sigma = 0.1$. The ensemble learning curves are plotted in Figure 3-8 with 50 Monte Carlo simulations. For each Monte Carlo simulation, the learning curves are calculated on a segment of 100 testing data. The MSE is calculated between the continuous output (before taking the hard decision) and the desired signal. The performance of LMS and APA-1 is similar and the two learning curves almost overlap. For KLMS-NC, KAPA-1-NC, and KAPA-2-NC, the novelty criterion is employed with $\delta_1 = 0.26$, $\delta_2 = 0.08$. The dynamic change of the network size is also plotted in Figure 3-9 over the training. It can be seen that at the beginning, the network sizes increase quickly but after convergence the network sizes increase slowly. And in fact, we can stop adding new centers after convergence by noticing that the MSE does not change after convergence.

PART 2: Next, different noise variances are set. To make the comparison fair, we tune the novelty criterion parameters to make the network size almost the same (around 100) in each scenario by cross-validation. For each setting, 20 Monte Carlo simulations are run with different training data and different testing data. The size of the training data is 1000 and the size of the testing data is 10^5. The filters are fixed during the testing phase. The results are presented in Figure 3-10. The normalized signal-noise-ratio (SNR) is defined as $10 \log_{10} \frac{1}{\sigma^2}$. It is clearly shown that KAPA-2-NC outperforms the KLMS-NC

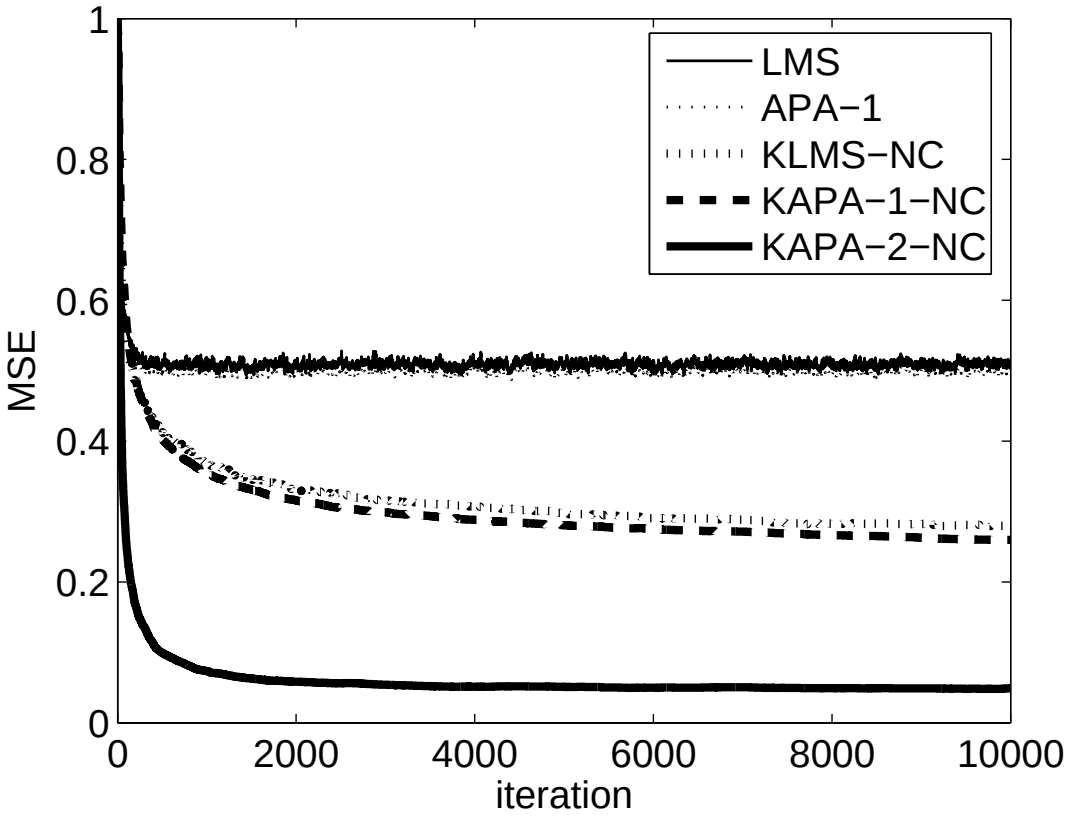

Figure 3-8. Ensemble learning curves of LMS, APA-1, KLMS-NC, KAPA-1-NC and KAPA-2-NC in nonlinear channel equalization ($\sigma = 0.1$)

substantially in terms of bit error rate (BER). The linear methods never really work in this simulation regardless of SNR. The improvement of KAPA-1-NC on KLMS-NC is marginal but it exhibits a smaller variance. The roughness in the curves is mostly due to the variance from the stochastic training.

PART 3: In the last simulation, we test the tracking ability of the proposed methods by introducing an abrupt channel change during training. The size of training data is 1500. For the first 500 data, the channel model is kept the same as before, but for the last 1000 data the nonlinearity of the channel is switched to $r(i) = -x(i) + 0.9x(i)^2 + n(i)$. The ensemble learning curves from 100 Monte Carlo simulations are plotted in Figure 3-11 and the dynamic change of the network size is plotted in Figure 3-12. It is seen that the KAPA-2-NC outperforms other methods with its fast tracking speed. KAPA-1-NC and

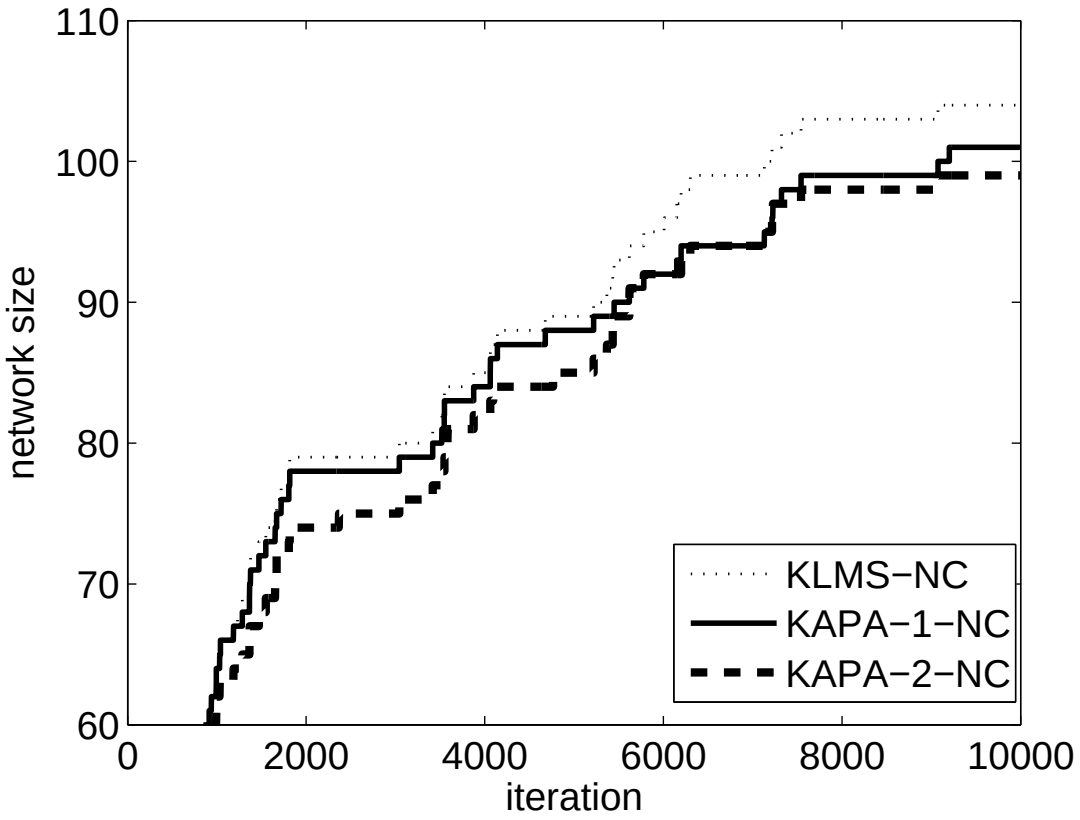

Figure 3-9. Network sizes of KLMS-NC, KAPA-1-NC and KAPA-2-NC over training in nonlinear channel equalization

KLMS-NC perform very similarly in this example. It is also noted that the network sizes increase right after the change to the channel model.

3.7 Conclusion

This chapter discusses the KAPA algorithm family which is a stochastic gradient methodology to solve least squares problems in RKHS. Since the KAPA update equations can be written as inner products, KAPA can be efficiently computed in the input space. Similar algorithms are discussed in [Richard et al., 2009, Slavakis and Theodoridis, 2008] from different perspectives[4].

Compared with the simplest online gradient decent algorithm in RKHS (KLMS), and perhaps the most complex (KRLS), the KAPA family provides a very flexible way of calculating a nonlinear filter online where the user can choose the performance/complexity

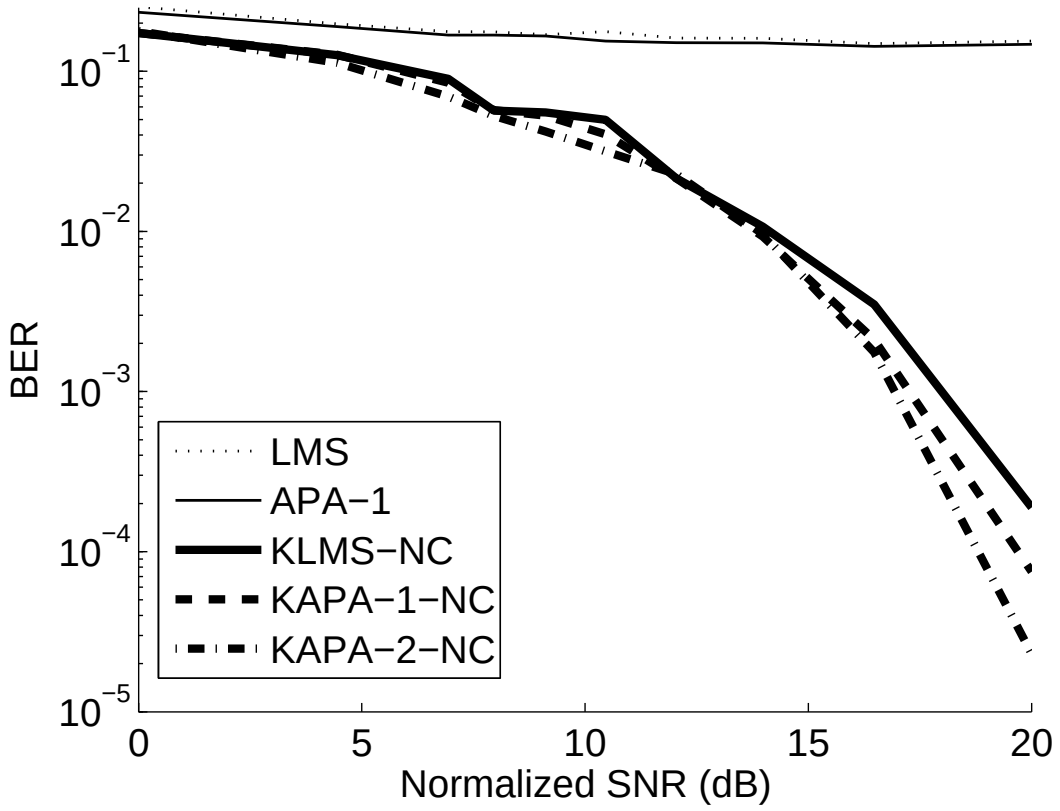

Figure 3-10. Performance comparison of LMS, APA-1, KLMS-NC, KAPA-1-NC and KAPA-2-NC with different SNR in nonlinear channel equalization

tradeoff at the point required by its application. Performance-wise KAPA is somewhere between KLMS and KRLS, which can be controlled by the window length K. The window length also controls the computational complexity. The relative performance was practically demonstrated in several important adaptive filtering applications, namely, time series prediction, nonlinear channel equalization and nonlinear noise cancellation.

Moreover, the KAPA family also provides a further theoretical understanding of RBF like neural networks, including the batch regularized network, and establishes relationships with a wealth of other algorithms available in the literature. Therefore, its role in building the taxonomy is also important and was well demonstrated in this chapter. We also illustrated the result of choosing the samples to keep in the filter using the novelty criterion. This simple criterion provides a large decrease in the number of samples

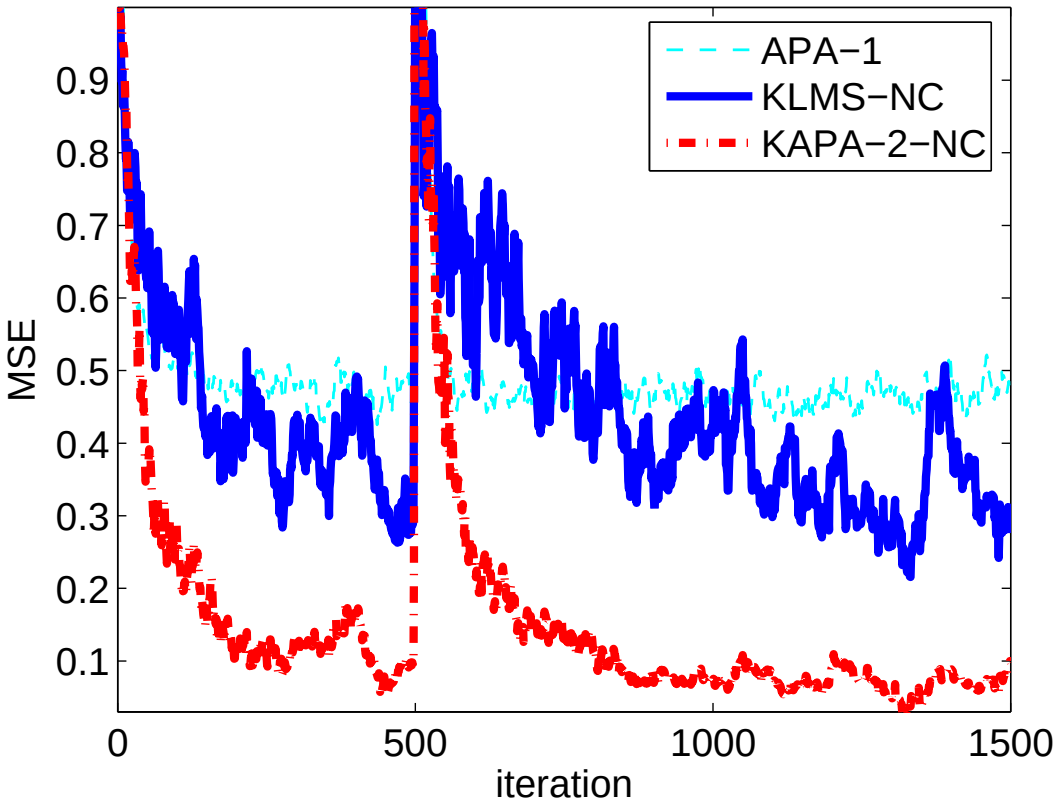

Figure 3-11. Ensemble learning curves of APA-1, KLMS-NC, and KAPA-2-NC with an abrupt change at iteration 500 in nonlinear channel equalization

with only a minor penalty in performance for the appropriate values of the thresholds, which are data dependent. This means that there is hope of decreasing even further the computational complexity with more sophisticated criteria.

Our emphasis in online adaptation is not accidental. Online algorithms are a necessity in many engineering applications (system identification of time varying systems, channel tracking and equalization, echo cancellation, etc). However we submit that online algorithms will also become increasingly more useful for batch machine learning algorithms applied to large databases. In fact, the database sizes will continue to grow exponentially. This poses real problems for the algorithm designer because of the higher than linear increase in memory and computational complexity of batch algorithms ($O(N^2)$ and even $O(N^3)$ in some kernel algorithms). Today we still can afford these algorithmic

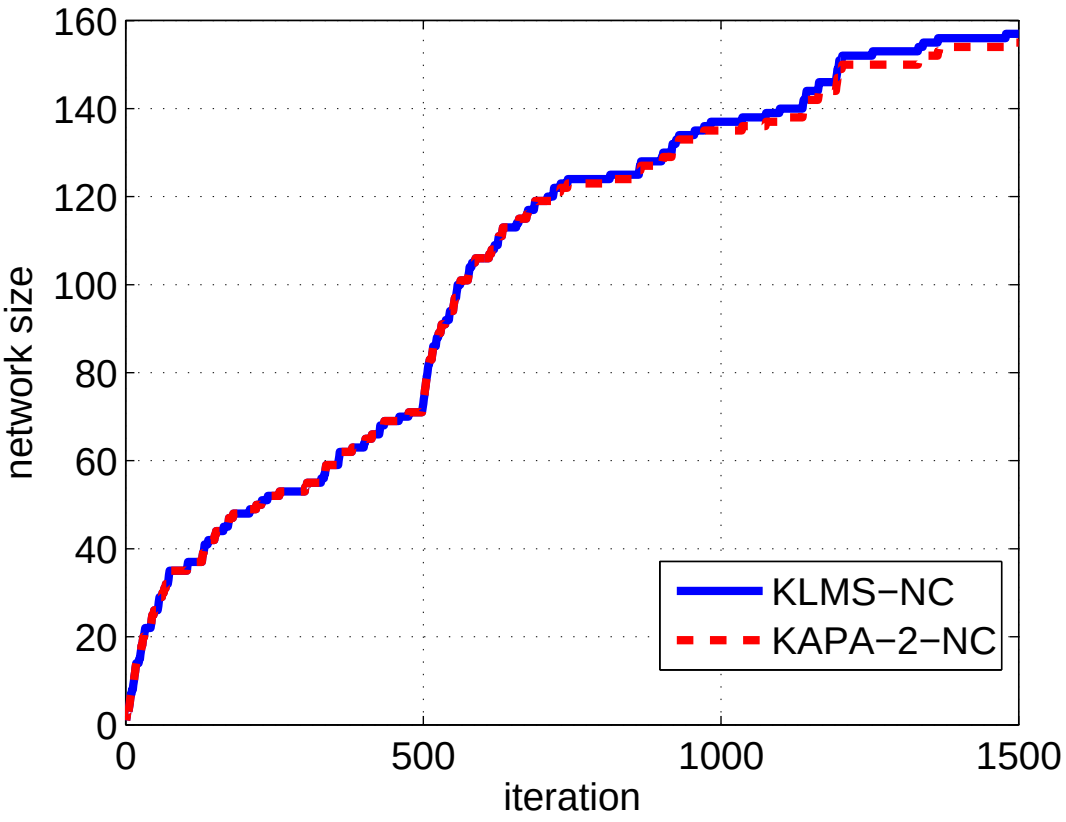

Figure 3-12. Network sizes of KLMS-NC and KAPA-2-NC over training with an abrupt change at iteration 500 in nonlinear channel equalization

complexities because the data sizes are reasonably small, but very soon this will be unbearable due to the exponential growth of database sizes. Designers will be limited to sub $O(N^2)$ computation complexities (i.e. $O(N)$, $O(NlogN)$) which requires a paradigm shift in algorithm design. Online algorithms in kernel spaces will be in the critical path to stochastically sample these large databases, and quickly find solutions in the neighborhood of the optimum.

We will briefly make here the point more clearly. Learning theory is driven by two major theories: statistical learning and optimization. Each contributes to errors in the optimal solution. In fact, although one would like to minimize the expected risk, we settle by minimizing the empirical risk incurring an error (the estimation error) on the way. Moreover, the function that minimizes the empirical risk may not belong to the class of

functions covered by the system, and we incur again an error (the approximation error). However, finding the optimal parameters of this system may be too complex to handle because of memory constraints or huge computational complexity. What we, among others [Bottou, 2008] are advocating, is to incur a third error (the computation error) that finds parameters in the neighborhood of the optimum efficiently. The KAPA algorithms are exactly the enablers of this last step. Although we are still far from linear complexity algorithms, this seems to be a very productive and relevant research direction.

Notes and References

1. **Affine Projection Algorithm.** The affine projection algorithm, due to Ozeki and Umeda [1984], is a generalization and improvement of the well-known normalized least mean square algorithm. Following this early work, Gay and Tavathia [1995] described a fast implementation of the affine projection algorithm in the time-domain, which features LMS-like complexity and RLS-like convergence in speech signal processing. Tanaka et al. [1999] proposed another fast implementation of the algorithm called *block exact fast affine projection*, using the frequency-domain approach; the algorithm exploits a fast FIR filtering technique based on the idea of fast convolution that uses the fast Fourier transform algorithm. Sankaran and Beex [2000] presented an analysis of convergence behavior of the algorithm with the following conclusions:

- The learning curve of an affine projection adaptive filter consists of the sum of exponential terms;
- An affine projection adaptive filter converges at a rate faster than that of the corresponding normalized LMS filter;
- As more delayed inputs are used, the rate of convergence improves, but the rate at which improvement is attained decreases.

For a discussion of regularization in fast affine projection implementation, please see Rombouts and Moonen [2000].

2. **Affine Projection Interpretation.** For our own study reported in the chapter, we use the recursion of APA-2 to explain why it can be interpreted as a projection problem onto an affine

space. We roughly follow the derivation in [Sayed, 2003]. To be straight, define two estimation error vectors: the a priori output estimation error

$$\mathbf{e}(i) = \mathbf{d}(i) - \mathbf{U}(i)^T \mathbf{w}(i-1) \tag{3–42}$$

and the a posteriori output estimation error

$$\mathbf{r}(i) = \mathbf{d}(i) - \mathbf{U}(i)^T \mathbf{w}(i) \tag{3–43}$$

Then, it can be shown that the recursion of APA-2 (3–10) is the exact solution to the following *local* optimization problem:

$$\begin{aligned} &\min_{\mathbf{w}(i)} ||\mathbf{w}(i) - \mathbf{w}(i-1)||^2 \\ &\text{subject to} \quad \mathbf{r}(i) = (\mathbf{I} - \eta \mathbf{U}(i)^T \mathbf{U}(i)[\mathbf{U}(i)^T \mathbf{U}(i) + \varepsilon \mathbf{I}]^{-1})\mathbf{e}(i) \end{aligned} \tag{3–44}$$

In other words, we seek a $\mathbf{w}(i)$ that is closest to $\mathbf{w}(i-1)$ in the Euclidean norm sense and subject to an equality constraint between $\mathbf{r}(i)$ and $\mathbf{e}(i)$. This constraint guarantees that $\mathbf{U}(i)^T \mathbf{w}(i)$ will be a better estimate for $\mathbf{d}(i)$ than $\mathbf{U}(i)^T \mathbf{w}(i-1)$ for any step-size parameter η in the interval $(0, 2)$.

A special case of the formulation (3–44) admits an interpretation in terms of projections onto affine subspaces. Setting $\eta = 1$ and $\varepsilon = 0$ in (3–44) yields

$$\min_{\mathbf{w}(i)} ||\mathbf{w}(i) - \mathbf{w}(i-1)||^2 \quad \text{subject to } \mathbf{r}(i) = \mathbf{0} \tag{3–45}$$

or equivalently,

$$\min_{\mathbf{w}(i)} ||\mathbf{w}(i) - \mathbf{w}(i-1)||^2 \quad \text{subject to } \mathbf{d}(i) = \mathbf{U}(i)^T \mathbf{w}(i) \tag{3–46}$$

A geometric interpretation of this equation is as follows: For any given data set $\{\mathbf{d}(i), \mathbf{U}(i)\}$, there may be infinitely many vector $\mathbf{w}$ that solve $d(i) = \mathbf{U}(i)^T \mathbf{w}$. The set of all such $\mathbf{w}$ is an affine subspace, or more precisely the intersection of K affine subspaces (it does not necessarily pass through the origin $\mathbf{w} = \mathbf{0}$). Given $\mathbf{w}(i-1)$, APA-2 selects that particular vector $\mathbf{w}(i)$ from this subspace that is closest to $\mathbf{w}(i-1)$ in the Euclidean norm sense. We therefore say that $\mathbf{w}(i)$ is obtained as the projection of $\mathbf{w}(i-1)$ onto the affine subspace.

3. **Adaptive Noise Cancellation.** Adaptive echo canceller and the adaptive line enhancer may be viewed as examples of the adaptive noise canceller, although they may be intended for different applications. The initial work on adaptive echo cancelers started around 1965. Sondhi [1967] recognized that Kelly of Bell Telephone Laboratories was the first to propose the use of an adaptive filter for echo cancellation. This invention and its refinement are described in patents by Kelly and Logan [1970] and Sondhi [1970]. The adaptive line enhancer was originated by Widrow and his coworkers at Stanford University. Widrow et al. [1975] reported their early work of building a device to cancel 60Hz interference at the output of an electrocardiographic amplifier and recorder in 1965. The adaptive line enhancer and its application as an adaptive detector were patented by McCool et al. [1980].

The first adaptive nonlinear noise cancellation appeared in [Coker and Simkins, 1980], where a simple nonlinear extension of the tapped delay line filter was trained by the LMS algorithm. Stapleton and Bass [1985] investigated a simple cascade model of a memoryless nonlinearity and a linear filter in the application of nonlinear noise control. More recently, recurrent radial-basis function networks [Billings and Fung, 1995], Volterra series [Li and Jiang, 2001] and fuzzy neural networks [Er et al., 2005] have also been investigated for adaptive noise cancellation.

4. **Kernel Affine Projection Algorithms.** Slavakis and Theodoridis [2008] derived a generalization of kernel affine projection algorithm based on the adaptive projected subgradient method. Classification is performed by metric projection mappings, sparsification is achieved by orthogonal projections, while online memory requirements and tracking are attained by oblique projections. The resulting sparsification scheme is similar to the classical sliding window adaptive schemes.

Richard et al. [2009] presented a similar algorithm using the idea of local optimization in (3–46). A sparsification method called coherence criterion was discussed to control the size of the network. The coherence criterion is similar to the novelty criterion.