Kernel Adaptive Filtering

Jose C. Principe and Weifeng Liu

Computational NeuroEngineering Laboratory (CNEL)
University of Florida
principe@cnel.ufl.edu, weifeng@amazon.com

Acknowledgments

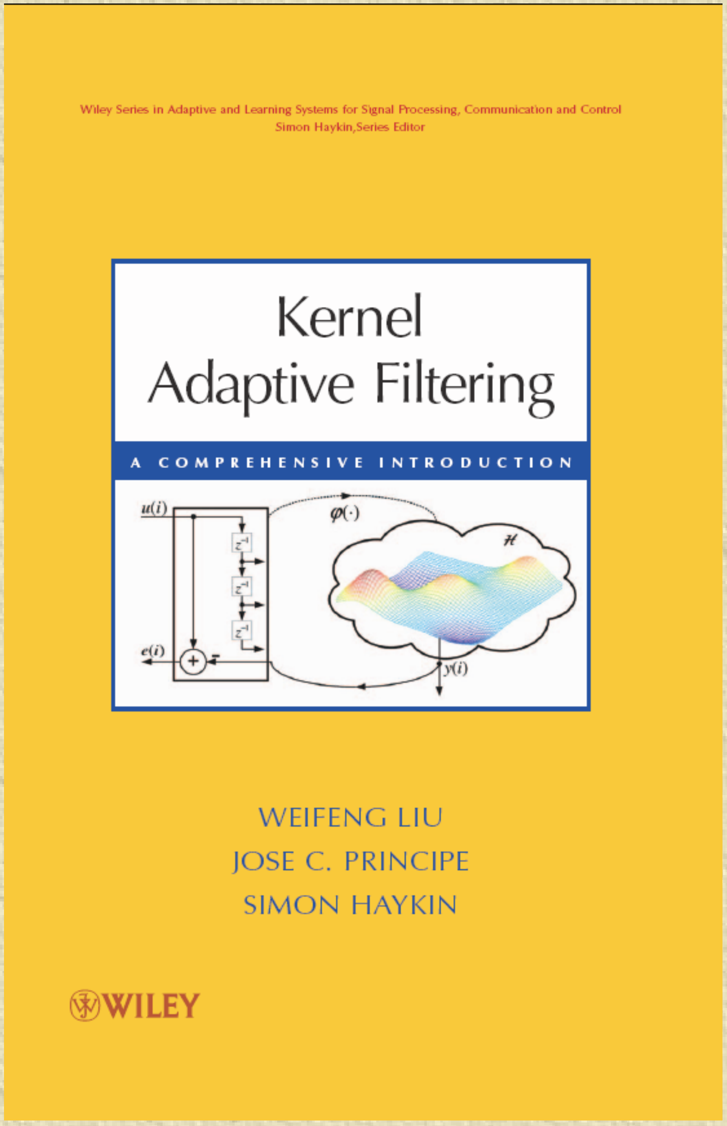
Dr. Badong Chen
Tsinghua University and Post Doc CNEL

NSF ECS – 0300340 and 0601271
(Neuroengineering program)

Outline

1. Optimal adaptive signal processing fundamentals
    - Learning strategy
    - Linear adaptive filters
2. Least-mean-square in kernel space
    - Well-posedness analysis of KLMS
3. Affine projection algorithms in kernel space
4. Extended recursive least squares in kernel space
5. Active learning in kernel adaptive filtering

Wiley Book (2010)


Wiley Series in Adaptive and Learning Systems for Signal Processing, Communication and Control
Simon Haykin, Series Editor

Kernel Adaptive Filtering

A COMPREHENSIVE INTRODUCTION

WEIFENG LIU
JOSE C. PRINCIPE
SIMON HAYKIN

WILEY

Papers are available at www.cnel.ufl.edu

Part 1: Optimal adaptive signal processing fundamentals

Machine Learning

Problem Definition for Optimal System Design

- Assumption: Examples are drawn independently from an unknown probability distribution *P*(u, y) that represents the rules of Nature.
- Expected Risk: $R(f) = \int L(f(u), y) dP(u, y)$
- Find *f** that minimizes *R*(*f*) among all functions.
- But we use a mapper class *F* and in general $f^* \notin F$
- The best we can have is $f_F^* \in F$ that minimizes *R*(*f*).
- *P*(u, y) is also unknown by definition.
- Empirical Risk: $\hat{R}_N(f) = 1/N \sum_i L(f(u_i), y_i)$
- Instead we compute $f_N \in F$ that minimizes R_n(f).
- Vapnik-Chervonenkis theory tells us when this will work, but the optimization is computationally costly.
- Exact estimation of f_N is done thru optimization.

Machine Learning Strategy

- The optimality conditions in learning and optimization theories are mathematically driven:
  - Learning theory favors cost functions that ensure a fast estimation rate when the number of examples increases (*small estimation error bound*).
  - Optimization theory favors super-linear algorithms (*small approximation error bound*)
- What about the computational cost of these optimal solutions, in particular when the data sets are huge?
- Estimation error will be small, but can not afford super linear solutions:

  Algorithmic complexity should be as close as possible to O(N).

Statistic Signal Processing
Adaptive Filtering Perspective

- Adaptive filtering also seeks optimal models for time series.
- The linear model is well understood and so widely applied. Optimal linear filtering is regression in functional spaces, where the user controls the size of the space by choosing the model order.
- Problems are fourfold:
  - Application conditions may be non stationary, i.e. the model must be continuously adapting to track changes.
  - In many important applications data arrives in real time, one sample at a time, so on-line learning methods are necessary.
  - Optimal algorithms must obey physical constrains, FLOPS, memory, response time, battery power.
  - Unclear how to go beyond the linear model.
- Although the optimalilty problem is the same as in machine learning, constraints make the computational problem different.

Machine Learning+Statistical SP

Change the Design Strategy

- Since achievable solutions are never optimal (non-reachable set of functions, empirical risk), goal should be to get quickly to the neighborhood of the optimal solution to save computation.
- The two types of errors are

$$R(f_N) - R(f^*) = R(f_F^*) - R(f^*) + \qquad \text{Approximation error}$$

$$+ R(f_N) - R(f_F^*) \qquad \text{Estimation error}$$

- But f_N is difficult to obtain, so why not create a third error (Optimization Error) to approximate the optimal solution

$$R(f_N) - R(\tilde{f}_N) = \rho$$

  provided it is computationally simpler to obtain.
- So the problem is to find *F*, *N* and ρ for each application.

Leon Bottou: The Tradeoffs of Large Scale Learning, NIPS 2007 tutorial

Learning Strategy in Biology

- In Biology optimality is stated in relative terms: the best possible response within a fixed time and with the available (finite) resources.
- Biological learning shares both constraints of small and large learning theory problems, because it is limited by the number of samples and also by the computation time.
- Design strategies for optimal signal processing are similar to the biological framework than to the machine learning framework.
- What matters is “how much the error decreases per sample for a fixed memory/ flop cost”
- It is therefore no surprise that the most successful algorithm in adaptive signal processing is the least mean square algorithm (LMS) which **never reaches** the optimal solution, but is O(L) and tracks continuously the optimal solution!

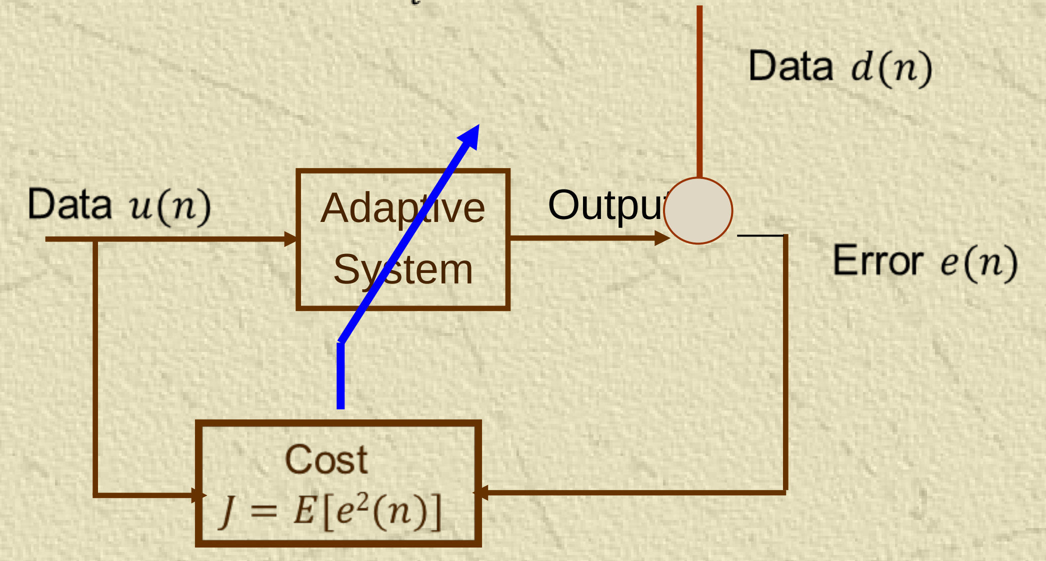
Extensions to Nonlinear Systems

- Many algorithms exist to solve the on-line linear regression problem:
  - LMS stochastic gradient descent
  - LMS-Newton handles eigenvalue spread, but is expensive
  - Recursive Least Squares (RLS) tracks the optimal solution with the available data.
- Nonlinear solutions either append nonlinearities to linear filters (not optimal) or require the availability of all data (Volterra, neural networks) and are not practical.
- Kernel based methods offers a very interesting alternative to neural networks.
  - Provided that the adaptation algorithm is written as an inner product, one can take advantage of the "kernel trick".
  - Nonlinear filters in the input space are obtained.
  - The primary advantage of doing gradient descent learning in RKHS is that the performance surface is still quadratic, so there **are no local minima**, while the filter now is nonlinear in the input space.

Adaptive Filtering Fundamentals

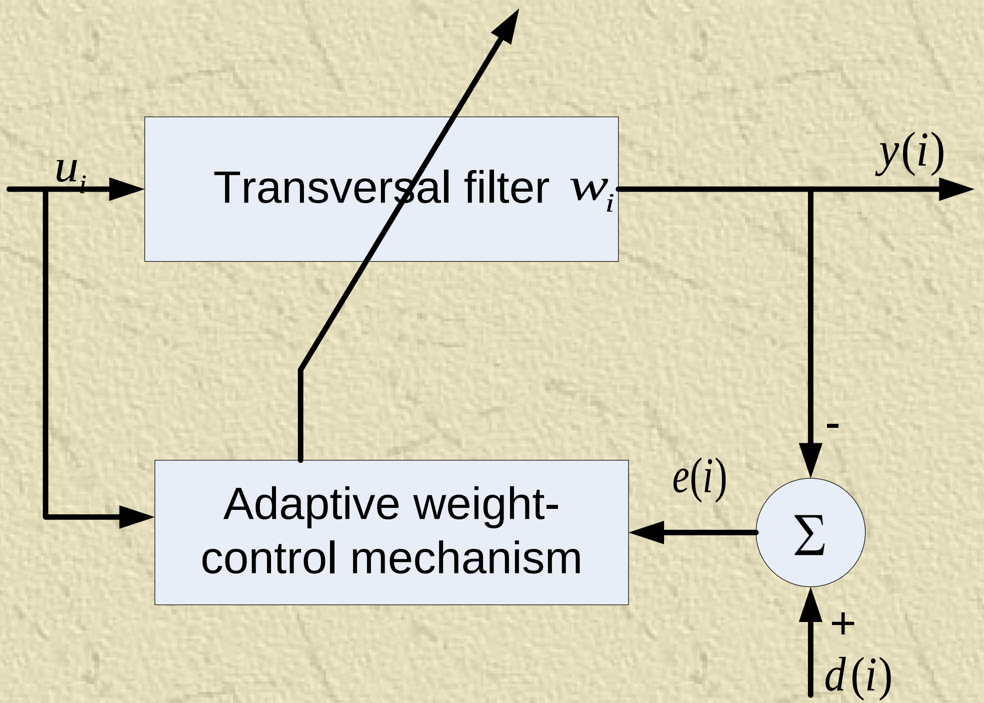
- Adaptive Filter Framework
  - Filtering is regression in functional spaces (time series) $\min_w J(e(n), w)$

$$f(u,w) = \sum_i w_i u(n-i)$$



- Optimal solution is least squares $w^* = R^{-1}p$, but now R is the autocorrelation of the data input (over the lags), and p is the crosscorrelation vector.

On-Line Learning for Linear Filters



Notation:

w_i weight estimate at time i (vector) (dim = *l*)

u_i input at time i (vector)

e(i) estimation error at time i (scalar)

d(i) desired response at time i (scalar)

e_i estimation error at iteration i (vector)

d_i desired response at iteration i (vector)

G_i capital letter matrix

- The current estimate w_i is computed in terms of the previous estimate, w_{i-1} , as:

$$\boxed{w_i = w_{i-1} + G_i e_i}$$

e_i is the model prediction error arising from the use of w_{i-1} and G_i is a Gain term

On-Line Learning for Linear Filters

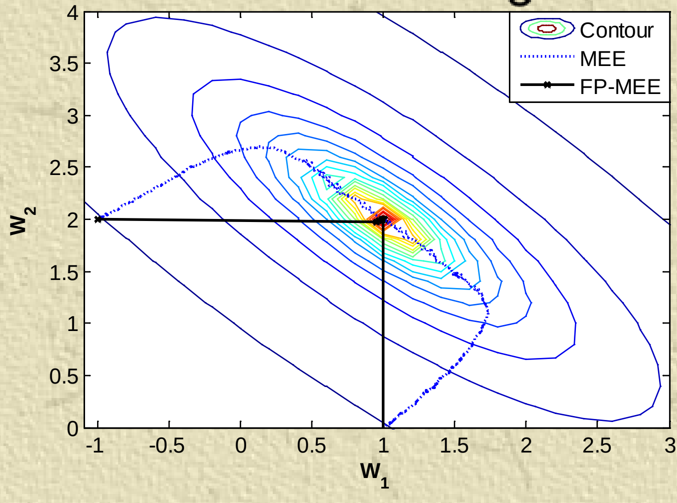
- Easiest technique is to search the performance surface *J* using **gradient descent learning** (batch).

$$w_i = w_{i-1} - \eta \nabla J_i \qquad w_i = w_{i-1} - \eta H^{-1} \nabla J_{i-1}$$

$$\lim_{i} E[w_i] = w^*$$

$$J = E[e^2(i)]$$

$$\eta \quad step\ size$$



- Gradient descent learning has well known compromises:
  - Stepsize η must be smaller than $1/\lambda_{max}$ (of R) for convergence
  - Speed of adaptation is controlled by λ_{min}
  - So eigenvalue spread of signal autocorrelation matrix controls speed of adaptation
  - The misadjustment (penalty w.r.t. optimum error) is proportional to stepsize, so fundamental compromise between adapting fast, and small misadjustment.

On-Line Learning for Linear Filters

- Gradient descent learning for linear mappers has also great properties
  - It accepts an unbiased sample by sample estimator that is easy to compute (O(L)), leading to the famous **LMS algorithm**.

$$w_i = w_{i-1} + \eta u_i e(i)$$

  - The LMS is a **robust estimator** (H^{∞}) algorithm.
  - For small stepsizes, the visited points during adaptation always **belong to the input data manifold** (dimension L), since algorithm always move in the opposite direction of the gradient.

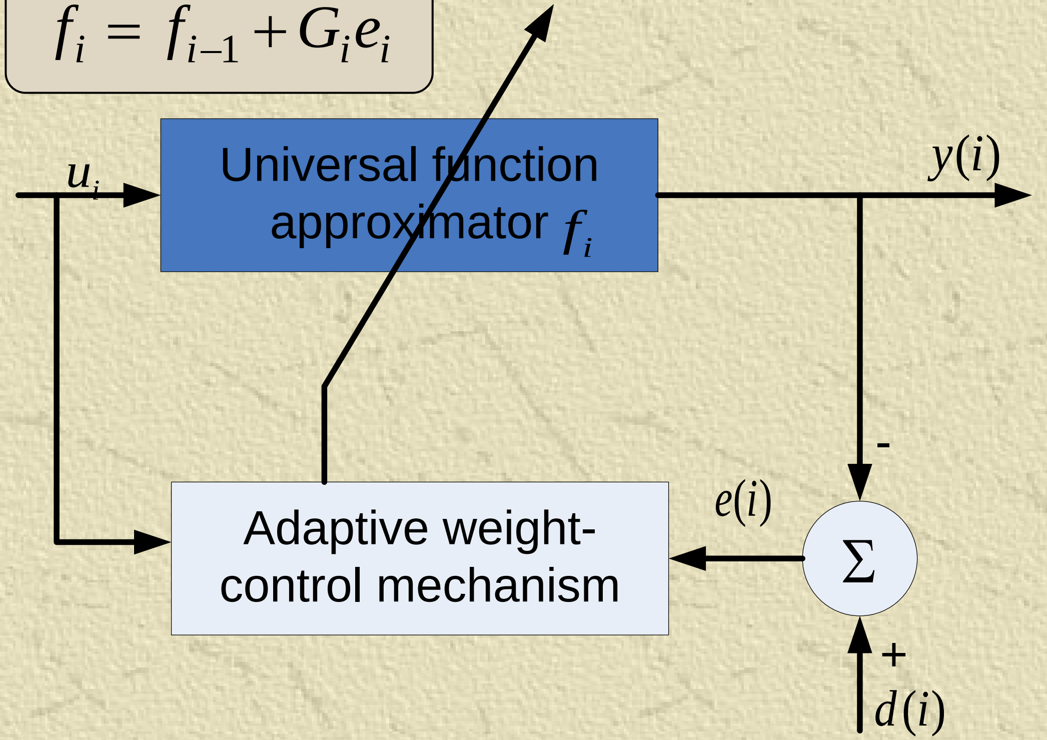
On-Line Learning for Non-Linear Filters?

- Can we generalize $w_i = w_{i-1} + G_i e_i$ to *nonlinear* models?

$$y = w^T u \longrightarrow y = f(u)$$

and create incrementally the nonlinear mapping?

$$\boxed{f_i = f_{i-1} + G_i e_i}$$



u_i
Universal function approximator f_i
$y(i)$
-
$e(i)$
Adaptive weight-control mechanism
Σ
+
$d(i)$

Part 2: Least-mean-squares in kernel space

Non-Linear Methods - Traditional
(Fixed topologies)

- Hammerstein and Wiener models
  - An explicit nonlinearity followed (preceded) by a linear filter
  - Nonlinearity is problem dependent
  - Do not possess universal approximation property
- Multi-layer perceptrons (MLPs) with back-propagation
  - Non-convex optimization
  - Local minima
- Least-mean-square for radial basis function (RBF) networks
  - Non-convex optimization for adjustment of centers
  - Local minima
- Volterra models, Recurrent Networks, etc

Non-linear Methods with kernels

- Universal approximation property (kernel dependent)
- Convex optimization (no local minima)
- Still easy to compute (kernel trick)
- But require regularization
- **Sequential (On-line) Learning with Kernels**

- (Platt 1991) Resource-allocating networks
  - Heuristic
  - No convergence and well-posedness analysis
- (Frieb 1999) Kernel adaline
  - Formulated in a batch mode
  - well-posedness not guaranteed
- (Kivinen 2004) Regularized kernel LMS
  - with explicit regularization
  - Solution is usually biased
- (Engel 2004) Kernel Recursive Least-Squares
- (Vaerenbergh 2006) Sliding-window kernel recursive least-squares

Neural Networks versus Kernel Filters

	ANNs	Kernel filters
Universal Approximators	YES	YES
Convex Optimization	NO	YES
Model Topology grows with data	NO	YES
Require Explicit Regularization	NO	YES/NO (KLMS)
Online Learning	YES	YES
Computational Complexity	LOW	MEDIUM

ANNs are semi-parametric, nonlinear approximators

Kernel filters are non-parametric, nonlinear approximators

Kernel Methods

- Kernel filters operate in a very special Hilbert space of functions called a Reproducing Kernel Hilbert Space (RKHS).
- A RKHS is an Hilbert space where all function evaluations are finite
- Operating with functions seems complicated and it is! But it becomes much easier in RKHS if we restrict the computation to inner products.
- Most linear algorithms can be expressed as inner products. Remember the FIR

$$y(n) = \sum_{i=0}^{L-1} w_i x(n-i) = \left\langle \mathbf{w}^{\mathbf{T}} \mathbf{x}(n) \right\rangle$$

Kernel Methods

- **Moore-Aronszajn theorem**
  - Every symmetric positive definite function of two real variables in E κ(*x*,*y*) defines a unique Reproducing Kernel Hilbert Space (RKHS).

$$(I)\forall x \in E \qquad \kappa(.,x) \in H$$

$$(II)\forall x \in E \qquad \forall f \in H, \qquad f(x) = < f, \kappa(.,x) >_{H_\kappa}$$

$$\kappa(x,y) = \exp(-h\|x-y\|^2)$$

- **Mercer's theorem**
  - Let *κ(x,y)* be symmetric positive definite. The kernel can be expanded in the series

$$\kappa(x,y) = \sum_{i=1}^{m} \lambda_i \varphi_i(x) \varphi_i(y)$$

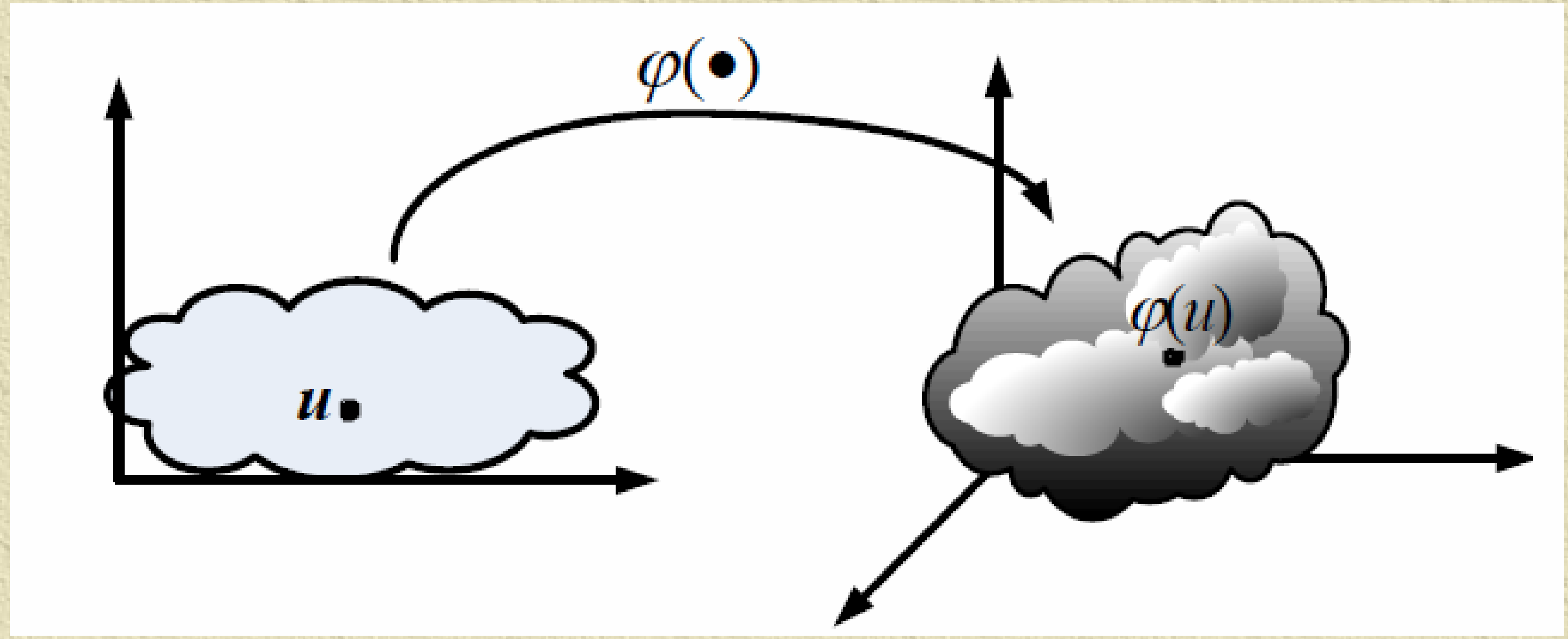
  - Construct the transform as

$$\varphi(x) = [\sqrt{\lambda_1}\varphi_1(x), \sqrt{\lambda_2}\varphi_2(x), \ldots, \sqrt{\lambda_m}\varphi_m(x)]^T$$

  - Inner product

$$\boxed{\langle \varphi(x), \varphi(y) \rangle = \kappa(x,y)}$$

Kernel methods



Mate L., Hilbert Space Methods in Science and Engineering, A. Hildger, 1989

Berlinet A., and Thomas-Agnan C., “Reproducing kernel Hilbert Spaces in probaability and Statistics, Kluwer 2004

Basic idea of on-line kernel filtering

- Transform data into a high dimensional feature space $\varphi_i := \varphi(u_i)$
- Construct a linear model in the feature space *F*

$$y = \langle \Omega, \varphi(u) \rangle_F$$

- Adapt iteratively parameters with gradient information

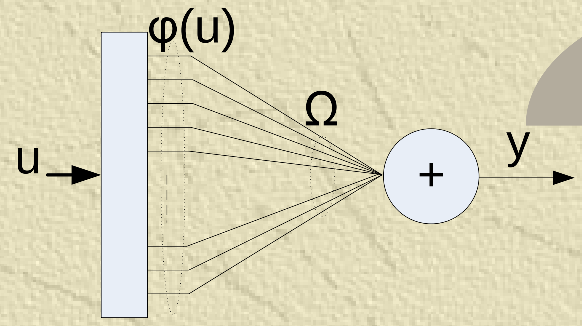
$$\Omega_i = \Omega_{i-1} + \eta \nabla J_i$$

- Compute the output

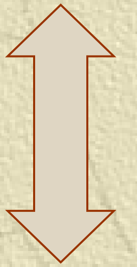
$$f_i(u) = \langle \Omega_i, \varphi(u) \rangle_F = \sum_{j=1}^{m_i} a_j \kappa(u, c_j)$$

- Universal approximation theorem
    - For the Gaussian kernel and a sufficient large m_i, $f_i(u)$ can approximate any continuous input-output mapping arbitrarily close in the L_p norm.

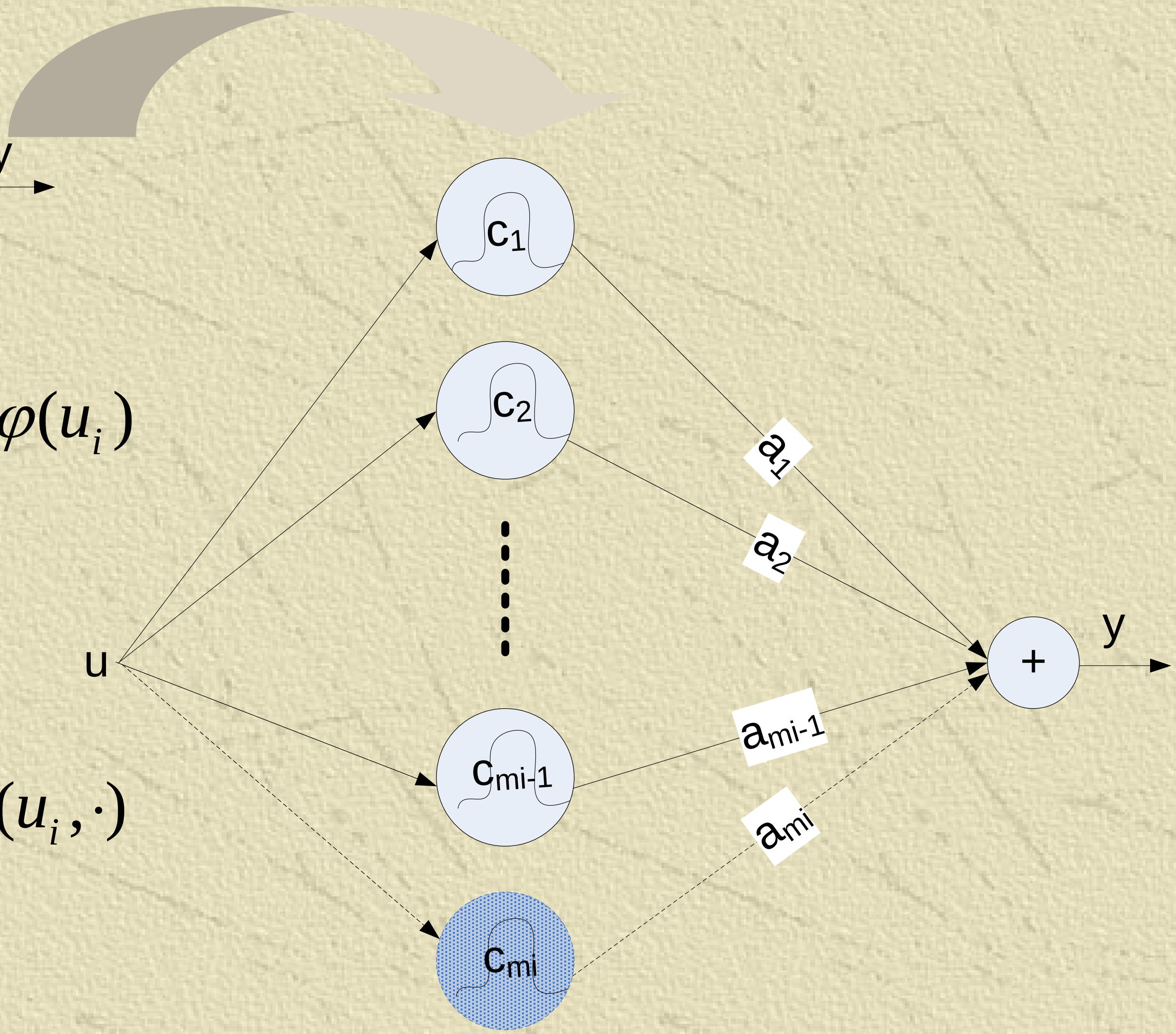
Growing network structure



$$\Omega_i = \Omega_{i-1} + \eta e(i)\varphi(u_i)$$



$$f_i = f_{i-1} + \eta e(i)\kappa(u_i, \cdot)$$



Kernel Least-Mean-Square (KLMS)

- Least-mean-square

$$w_i = w_{i-1} + \eta u_i e(i) \qquad e(i) = d(i) - w_{i-1}^T u_i \qquad w_0$$

- Transform data into a high dimensional feature space *F* $\varphi_i := \varphi(u_i)$

$$\Omega_0 = 0$$
$$e(i) = d(i) - \langle \Omega_{i-1}, \varphi(u_i) \rangle_F$$
$$\Omega_i = \Omega_{i-1} + \eta \varphi(u_i) e(i)$$
$$\Omega_i = \sum_{j=1}^{i} \eta e(j) \varphi(u_j)$$
$$f_i(u) = \langle \Omega_i, \varphi(u) \rangle_F = \sum_{j=1}^{i} \eta e(j) \kappa(u, u_j)$$

$$\Omega_0 = 0$$
$$e(1) = d(1) - \langle \Omega_0, \varphi(u_1) \rangle_F = d(1)$$
$$\Omega_1 = \Omega_0 + \eta \varphi(u_1) e(1) = a_1 \varphi(u_1)$$
$$e(2) = d(2) - \langle \Omega_1, \varphi(u_2) \rangle_F$$
$$= d(2) - \langle a_1 \varphi(u_1), \varphi(u_2) \rangle_F$$
$$= d(2) - a_1 \kappa(u_1, u_2)$$
$$\Omega_2 = \Omega_1 + \eta \varphi(u_2) e(2)$$
$$= a_1 \varphi(u_1) + a_2 \varphi(u_2)$$
$$\vdots$$

- RBF Centers are the samples, and Weights are the errors!

Kernel Least-Mean-Square (KLMS)

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

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

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

$$f_i = f_{i-1} + \eta e(i)\kappa(\mathbf{u}(i),.)$$

Free Parameters in KLMS
Step size

- Traditional wisdom in LMS still applies here.

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

where $\mathbf{G}_\varphi$ is the Gram matrix, and N its dimensionality.

- For translation invariant kernels, $\kappa(u(j),u(j))=g_0$, is a constant independent of the data.
- The Misadjustment is therefore $M = \frac{\eta}{2N} tr[\mathbf{G}_\varphi]$

Free Parameters in KLMS
Rule of Thumb for *h*

- Although KLMS is not kernel density estimation, these rules of thumb still provide a starting point.
- Silverman’s rule can be applied

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

  where σ is the input data s.d., R is the interquartile, N is the number of samples and L is the dimension.
- Alternatively: take a look at the dynamic range of the data, assume it uniformly distributed and select h to put 10 samples in 3 σ.

Free Parameters in KLMS

Kernel Design

- The Kernel defines the inner product in RKHS
  - Any positive definite function (Gaussian, polynomial, Laplacian, etc.), but we should choose a kernel that yields a class of functions that allows universal approximation.
  - A *strictly positive definite* function is preferred because it will yield universal mappers (Gaussian, Laplacian).

See Sriperumbudur et al, *On the Relation Between Universality, Characteristic Kernels and RKHS Embedding of Measures, AISTATS 2010*

Free Parameters in KLMS
Kernel Design

- Estimate and minimize the generalization error, e.g. cross validation

- Establish and minimize a generalization error upper bound, e.g. VC dimension

- Estimate and maximize the posterior probability of the model given the data using Bayesian inference

Free Parameters in KLMS
Bayesian model selection

- The posterior probability of a Model H (kernel and parameters θ) given the data is

$$p(H_i \mid \mathbf{d}, \mathbf{U}) = \frac{p(\mathbf{d} \mid \mathbf{U}, H_i)\, p(H_i)}{p(\mathbf{d} \mid \mathbf{U})}$$

where d is the desired output and U is the input vector.

This is hardly ever done for the kernel function, but it can be applied to θ and leads to Bayesian principles to adapt the kernel parameters.

Free Parameters in KLMS
Maximal marginal likelihood

- For each possible kernel, we maximize the following objective function with respective to θ

$$J(H_i) = \max_{\theta}[-1/2\,\mathbf{d}^T(\mathbf{G}+\sigma_n^2\mathbf{I})^{-1}\mathbf{d} - 1/2\log\left|\mathbf{G}+\sigma_n^2\mathbf{I}\right| - N/2\log(2\pi)]$$

where G is the Gram matrix.

Then H^o which maximizes $J(H)$ is the optimal kernel with the corresponding θ^o from the optimization itself.

Sparsification

- Filter size increases linearly with samples!
- If RKHS is compact and the environment stationary, we see that there is no need to keep increasing the filter size.
- Issue is that we would like to implement it on-line!
- Two ways to cope with growth:
  - Novelty Criterion
  - Approximate Linear Dependency
- First is very simple and intuitive to implement.

Sparsification

Novelty Criterion

- Present dictionary is $C(i) = \{c_j\}_{j=1}^{m_i}$. When a new data pair arrives (**u**(i+1),d(i+1)).
- First compute the distance to the present dictionary

$$dis = \min_{c_j \in C} \left\| u(i+1) - c_j \right\|$$

- If smaller than threshold δ_1 do not create new center
- Otherwise see if the prediction error is larger than δ_2 to augment the dictionary.
- δ_1 ~ 0.1 kernel size and δ_2 ~ sqrt of MSE

Sparsification

Approximate Linear Dependency

- Engel proposed to estimate the distance to the linear span of the centers, i.e. compute

$$dis = \min_{\vee b} \left\| \varphi(u(i+1)) - \sum_{c_j \in C} b_j \varphi(c_j) \right\|$$

Which can be estimated by

$$dis^2 = \kappa(\mathbf{u}(i+1), \mathbf{u}(i+1)) - \mathbf{h}(i+1)^T \mathbf{G}^{-1}(i)\mathbf{h}(i+1)$$

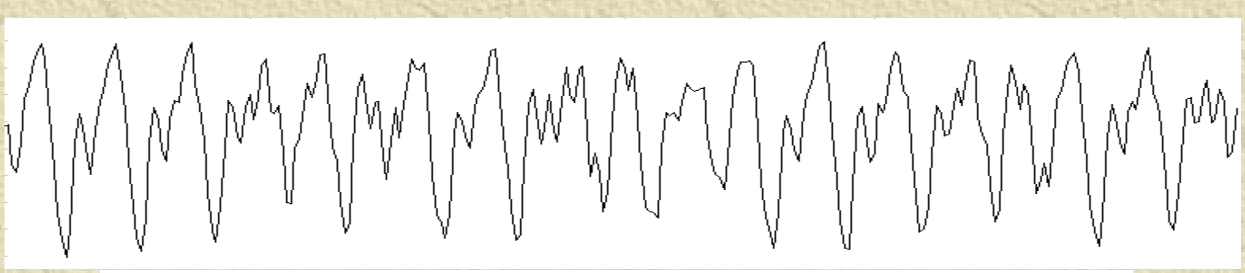
Only increase dictionary if *dis* larger than threshold

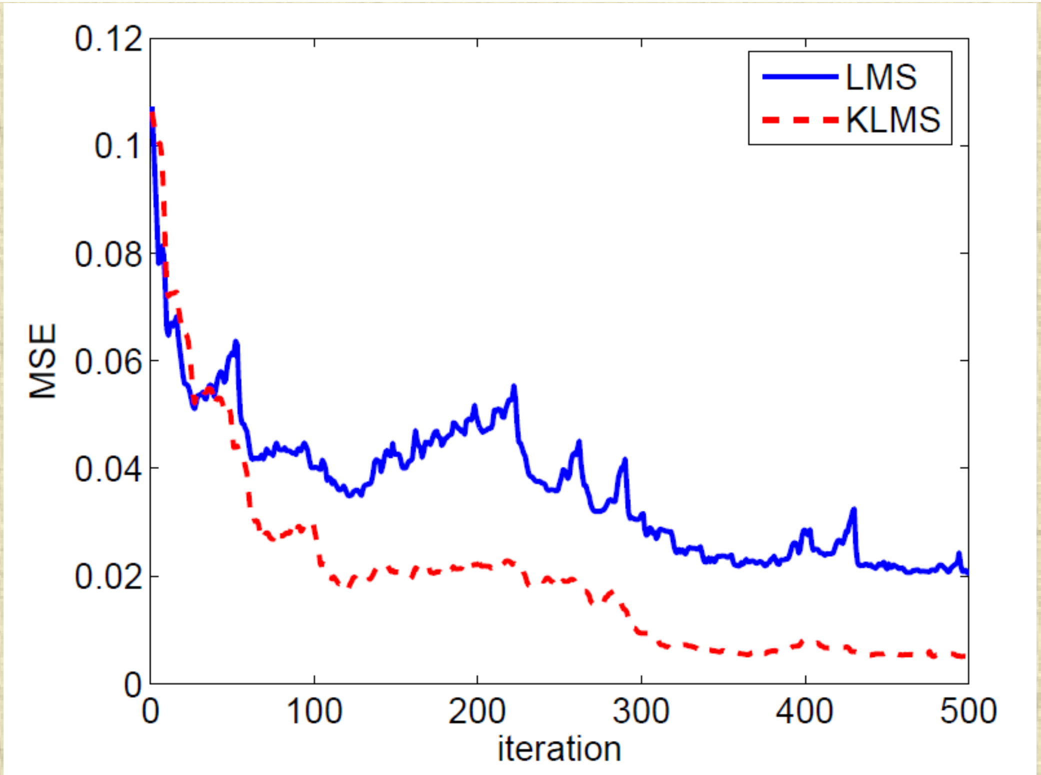
- Complexity is O(m^2)
- Easy to estimate in KRLS (*dis*~r(i+1))
- Can simplify the sum to the nearest center, and it defaults to NC

$$dis = \min_{\vee b, c_j \in C} \left\| \varphi(u(i+1)) - \varphi(c_j) \right\|$$

KLMS- Mackey-Glass Prediction

$$\dot{x}(t) = -0.1x(t) + \frac{0.2x(t-\tau)}{1+x(t-\tau)^{10}} \qquad \tau = 30$$





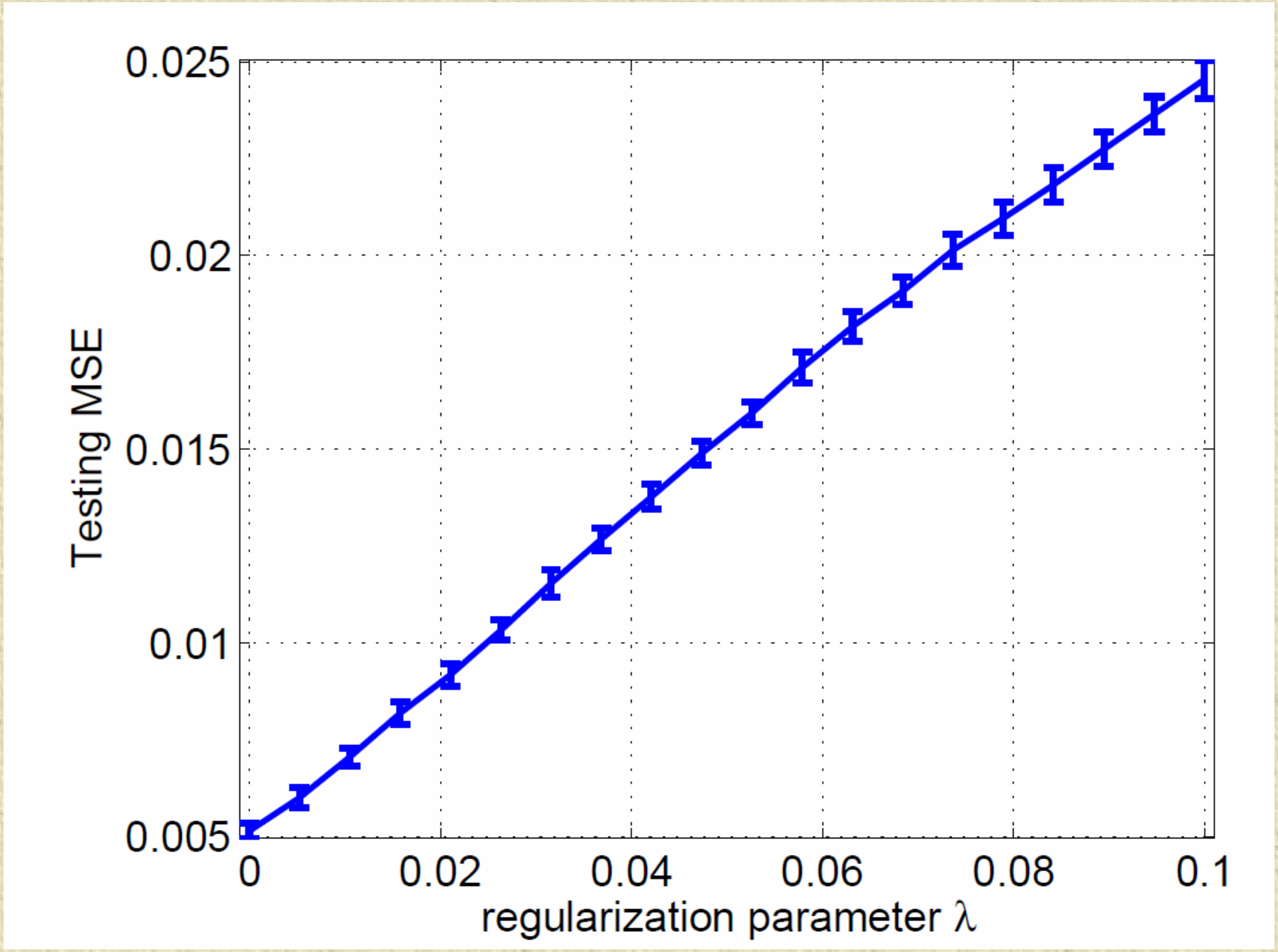
LMS

η=0.2

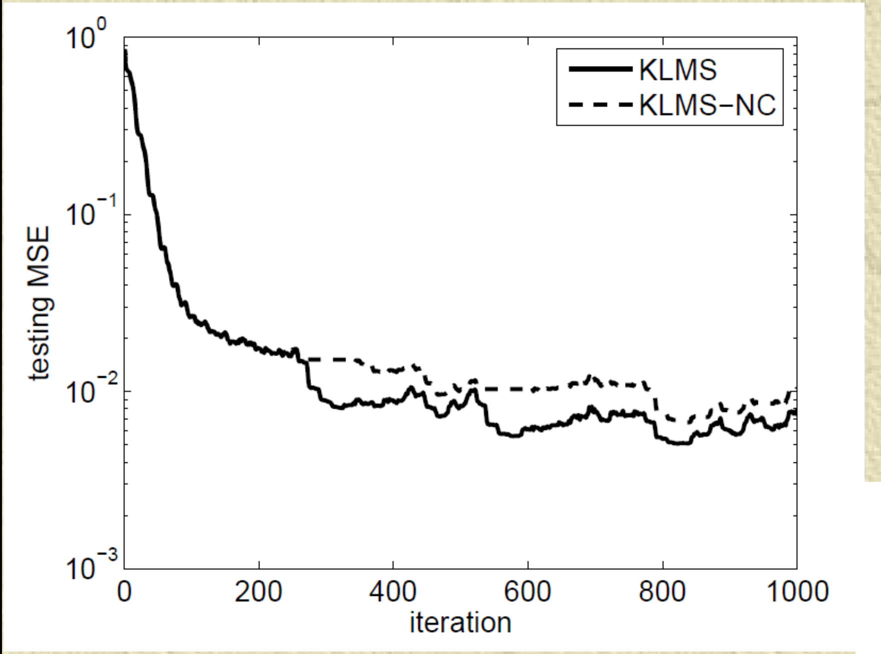
KLMS

a=1, η=0.2

Regularization worsens performance

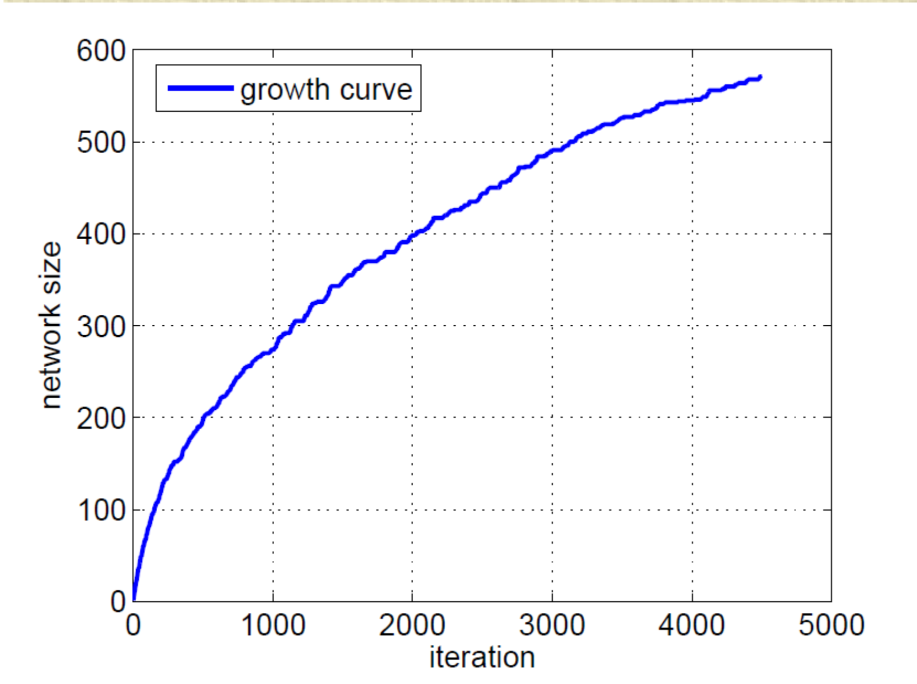


Performance Growth tradeoff



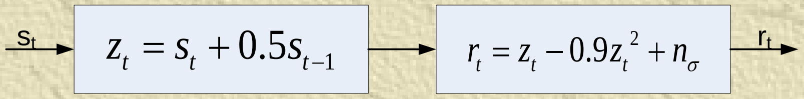
δ_1=0.1, δ_2=0.05

η=0.1, a=1

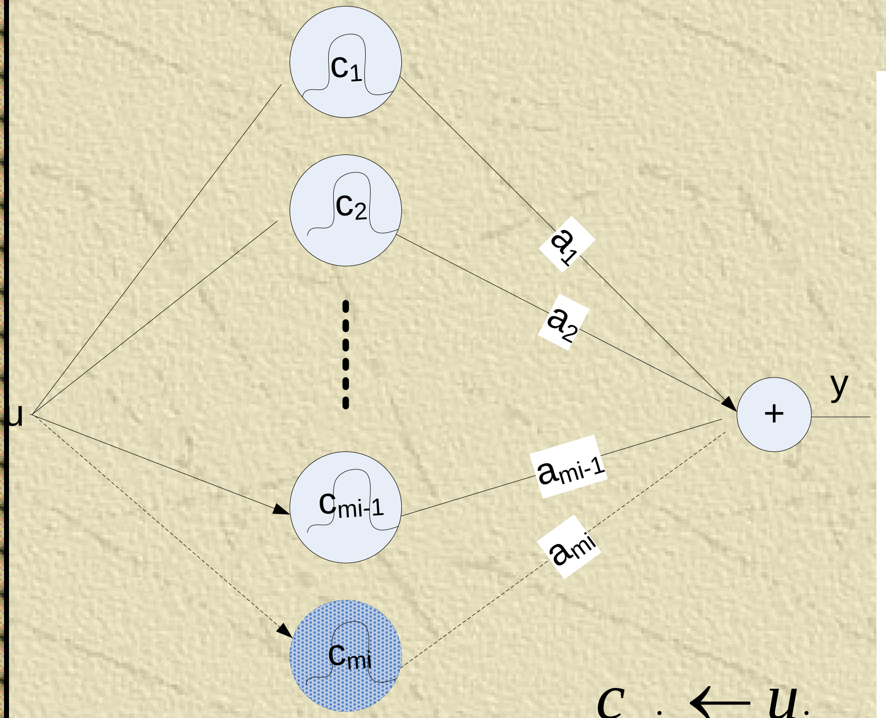


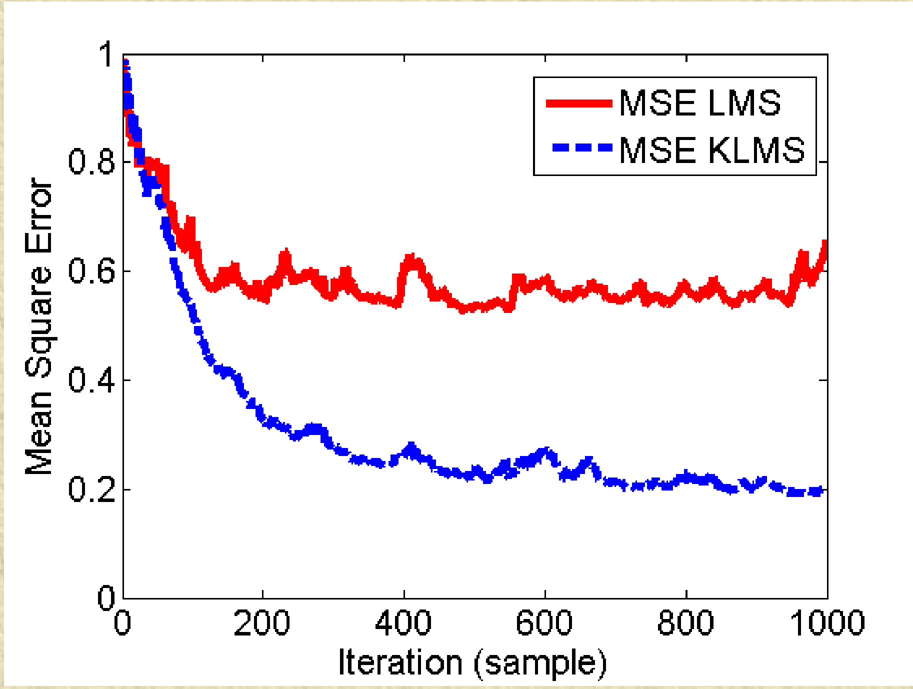
KLMS- Nonlinear channel equalization

$$f_i(u) = \langle \Omega_i, \varphi(u) \rangle_F = \sum_{j=1}^{i} \eta e(j) \kappa(u, u_j)$$



$$s_t \rightarrow \boxed{z_t = s_t + 0.5 s_{t-1}} \rightarrow \boxed{r_t = z_t - 0.9 z_t^2 + n_\sigma} \rightarrow r_t$$





$$c_{mi} \leftarrow u_i$$

$$a_{mi} \leftarrow \eta e(i)$$

Nonlinear channel equalization

Algorithms	Linear LMS (η=0.005)	KLMS (η=0.1) (NO REGULARIZATION)	RN (REGULARIZED λ=1)
BER (σ = .1)	0.162 ± 0.014	0.020 ± 0.012	0.008 ± 0.001
BER (σ = .4)	0.177 ± 0.012	0.058 ± 0.008	0.046 ± 0.003
BER (σ = .8)	0.218 ± 0.012	0.130 ± 0.010	0.118 ± 0.004

$$\kappa(u_i, u_j) = \exp(-0.1 \| u_i - u_j \|^2)$$

Algorithms	Linear LMS	KLMS	RN
Computation (training)	O(l)	O(i)	O(i^3)
Memory (training)	O(l)	O(i)	O(i^2)
Computation (test)	O(l)	O(i)	O(i)
Memory (test)	O(l)	O(i)	O(i)

Why don’t we need to explicitly regularize the KLMS?

Self-regularization property of KLMS

- Assume the data model $d(i) = \Omega^o(\varphi_i) + v(i)$ then for any unknown vector Ω^0 the following inequality holds

$$\frac{\sum_{j=1}^{i} | e(j) - v(j) |^2}{\eta^{-1} \| \Omega^o \|^2 + \sum_{j=1}^{i-1} | v(j) |^2} < 1, \quad \textit{for all } i = 1, 2, \ldots, N$$

As long as the matrix $\{\eta^{-1} I - \varphi(i)\varphi(i)^T\}$ is positive definite. So

- H^∞ robustness

$$\| \vec{e} \|^2 < \eta^{-1} \| \Omega^o \|^2 + 2 \| \vec{v} \|^2$$

- And $\Omega(n)$ is upper bounded

$$\| \Omega_N \|^2 < \sigma_1 \eta (\| \Omega^o \|^2 + 2\eta \| \vec{v} \|^2)$$

σ_1 is the largest eigenvalue of Gφ

The solution norm of KLMS is always upper bounded i.e. the algorithm is well posed in the sense of Hadamard.

Liu W., Pokarel P., Principe J., "The Kernel LMS Algorithm", IEEE Trans. Signal Processing, Vol 56, # 2, 543 – 554, 2008.

Regularization Techniques

- Learning from finite data is ill-posed and a priori information to enforce ***Smoothness*** is needed.
- The key is to constrain the solution norm
  - In Least Squares constraining the norm yields

$$J(\Omega) = \frac{1}{N}\sum_{i=1}^{N}(d(i) - \Omega^T \varphi_i)^2, \text{ subject to } \|\Omega\|^2 < C$$

Norm constraint

  - In Bayesian modeling, the norm is the prior. (Gaussian process)

$$J(\Omega) = \frac{1}{N}\sum_{i=1}^{N}(d(i) - \Omega^T \varphi_i)^2 + \lambda \|\Omega\|^2$$

Gaussian distributed prior

  - In statistical learning theory, the norm is associated with the model capacity and hence the confidence of uniform convergence! (VC dimension and structural risk minimization)

Tikhonov Regularization

- In numerical analysis the method is to constrain the condition number of the solution matrix (or its eigenvalues)
- The singular value decomposition of Φ can be written

$$\mathbf{\Phi} = \mathbf{P}\begin{bmatrix} \mathbf{S} & 0 \\ 0 & 0 \end{bmatrix}\mathbf{Q}^T \qquad S = diag\{s_1, s_2, ..., s_r\}$$

Singular value

- The pseudo inverse to estimate Ω in $d(i) = \varphi(i)^T\Omega^0 + \nu(i)$ is

$$\Omega_{PI} = \mathbf{P}diag[s_1^{-1},...,s_r^{-1},0....0]\mathbf{Q}^T\mathbf{d}$$

which can be still ill-posed (very small s_r). Tikhonov regularized the least square solution to penalize the solution norm to yield

$$J(\Omega) = \left\|\mathbf{d} - \mathbf{\Phi}^T\Omega\right\| + \lambda\left\|\Omega\right\|^2$$

$$\Omega = Pdiag(\frac{s_1}{s_1^{\,2}+\lambda},...,\frac{s_r}{s_r^{\,2}+\lambda},0,...,0)Q^Td$$

Notice that if $\lambda = 0$, when s_r is very small, $s_r/(s_r^2 + \lambda) = 1/s_r \rightarrow \infty$.

However if $\lambda > 0$, when s_r is very small, $s_r/(s_r^2 + \lambda) = s_r/\lambda \rightarrow 0$.

Tikhonov and KLMS

For finite data and using small stepsize theory:

Denote $\varphi_i = \varphi(u_i) \in R^m$ $\quad R_\varphi = \frac{1}{N}\sum_{i=1}^{N} \varphi_i \varphi_i^T \quad R_x = P \Lambda P^T$

Assume the correlation matrix is singular, and

$$\varsigma_1 \geq \ldots \geq \varsigma_k > \varsigma_{k+1} = \ldots = \varsigma_m = 0$$

From LMS it is known that

$$E[\varepsilon_n(i)] = (1-\eta\varsigma_n)^i \varepsilon_n(0)$$

$$E[|\varepsilon_i(n)|^2] = \frac{\eta J_{\min}}{2-\eta\varsigma_n} + (1-\eta\varsigma_n)^{2i}(|\varepsilon_0(n)|^2 - \frac{\eta J_{\min}}{2-\eta\varsigma_n})$$

Define $\Omega(i) - \Omega^0 = \sum_{n=1}^{m} \varepsilon_n(i) P_n$ so

$$E[\Omega(i)] = \Omega^0 + \sum_{j=1}^{M} (1-\eta\varsigma_j)^i \varepsilon_j(0) \mathbf{P}_j = \sum_{j=1}^{M} [1-(1-\eta\varsigma_j)^i] \Omega_j^0 \mathbf{P}_j \quad \Omega(0) = 0 \quad \varepsilon_j(0) = -\Omega_j^0$$

and

$$\left\| E[\Omega(i)] \right\|^2 \leq \sum_{j=1}^{M} (\Omega_j^0)^2 = \left\| \Omega^0 \right\|^2 \qquad \eta \leq 1/\varsigma_{\max}$$

Tikhonov and KLMS

- In the worst case, substitute the optimal weight by the pseudo inverse

$$E[\Omega(i)] = \mathbf{P}diag[(1-(1-\eta\varsigma_1)^i)s_1^{-1},...,(1-(1-\eta\varsigma_r)^i)s_r^{-1},0....0]\mathbf{Q}^T\mathbf{d}$$

- Regularization function for finite N in KLMS

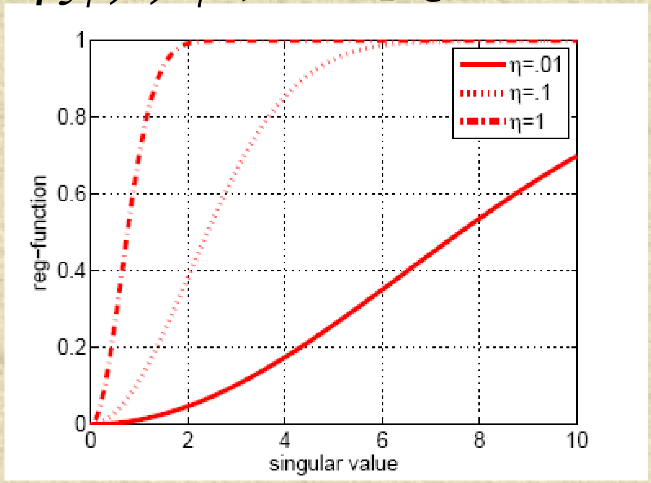
$$[1-(1-\eta s_n{}^2/N)^N]\cdot s_n{}^{-1}$$

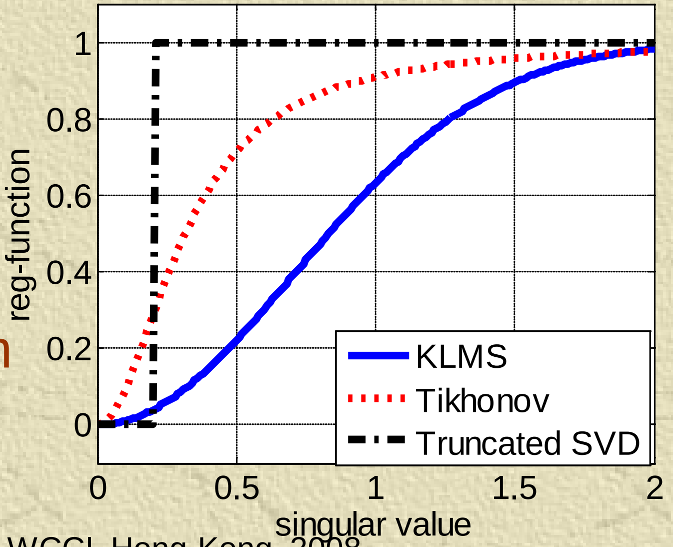
- No regularization $s_n{}^{-1}$
- Tikhonov

$$[s_n{}^2/(s_n{}^2+\lambda)]\cdot s_n{}^{-1}$$

- PCA

$$\begin{cases} s_n{}^{-1} \text{ if } s_n > \text{th} \\ 0 \quad \text{ if } s_n \le \text{th} \end{cases}$$





The stepsize and N control the reg-function in KLMS.

- Liu W., Principe J. The Well-posedness Analysis of the Kernel Adaline, Proc WCCI, Hong-Kong, 2008

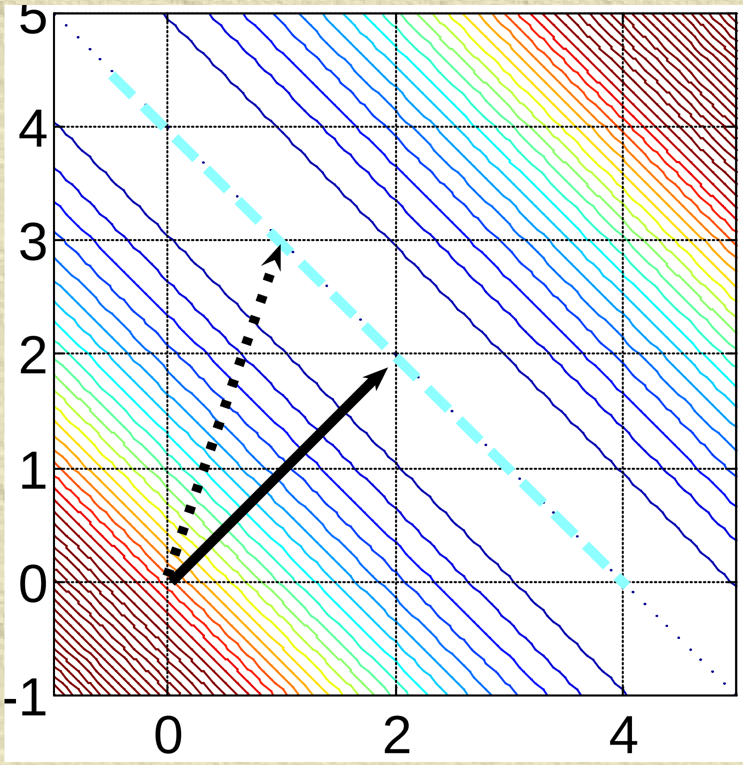
The minimum norm initialization for KLMS

- The initialization $\Omega_0 = 0$ gives the minimum possible norm solution.
- $$\Omega_i = \sum_{n=1}^{m} c_n P_n$$

$$\varsigma_1 \geq \ldots \geq \varsigma_k > 0$$

$$\varsigma_{k+1} = \ldots = \varsigma_m = 0$$

$$\| \Omega_i \|^2 = \sum_{n=1}^{k} \| c_n \|^2 + \sum_{n=k+1}^{m} \| c_n \|^2$$



KLMS and the Data Space

- KLMS search is insensitive to the 0-eigenvalue directions

$$E[\varepsilon_n(i)] = (1-\eta\varsigma_n)^i \varepsilon_n(0)$$

$$E[|\varepsilon_i(n)|^2] = \frac{\eta J_{\min}}{2-\eta\varsigma_n} + (1-\eta\varsigma_n)^{2i}(|\varepsilon_0(n)|^2 - \frac{\eta J_{\min}}{2-\eta\varsigma_n})$$

So if $\varsigma_n = 0$, $E[\varepsilon_n(i)] = \varepsilon_n(0)$ and $E[|\varepsilon_n(i)|^2] = |\varepsilon_n(0)|^2$

- The 0-eigenvalue directions do not affect the MSE

$$J(i) = E[|d - \Omega_i^T \varphi|^2]$$

$$J(i) = J_{\min} + \frac{\eta J_{\min}}{2}\sum_{n=1}^{m}\varsigma_n + \sum_{n=1}^{m}\varsigma_n(|\varepsilon_n(0)|^2 - \frac{\eta J_{\min}}{2})(1-\eta\varsigma_n)^{2i}$$

KLMS only finds solutions on the data subspace! It does not care about the null space!

Liu W., Pokarel P., Principe J., "The Kernel LMS Algorithm", IEEE Trans. Signal Processing, Vol 56, # 2, 543 – 554, 2008.

Energy Conservation Relation

The fundamental energy conservation relation holds in RKHS!

- Energy conservation in RKHS

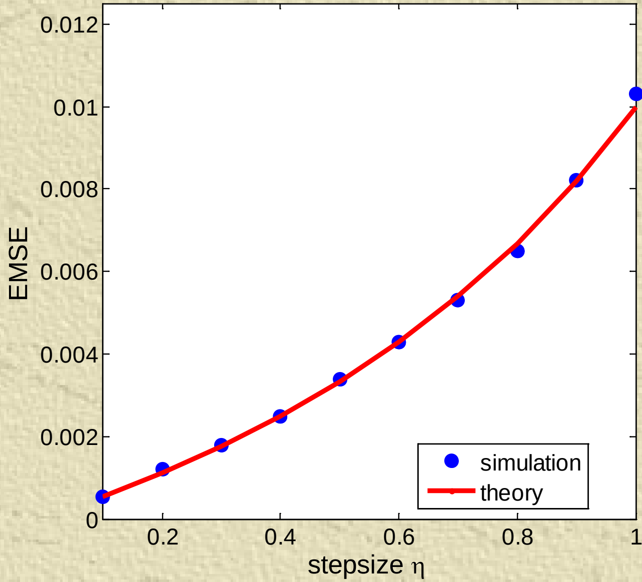
$$\left\|\tilde{\mathbf{\Omega}}(i)\right\|_{\mathrm{F}}^2 + \frac{e_a^2(i)}{\kappa\left(\boldsymbol{u}(i),\boldsymbol{u}(i)\right)} = \left\|\tilde{\mathbf{\Omega}}(i-1)\right\|_{\mathrm{F}}^2 + \frac{e_p^2(i)}{\kappa\left(\boldsymbol{u}(i),\boldsymbol{u}(i)\right)}$$

- Upper bound on step size for mean square convergence

$$\eta \le \frac{2E\left[\left\|\mathbf{\Omega}^*\right\|_{\mathrm{F}}^2\right]}{E\left[\left\|\mathbf{\Omega}^*\right\|_{\mathrm{F}}^2\right] + \sigma_v^2}$$

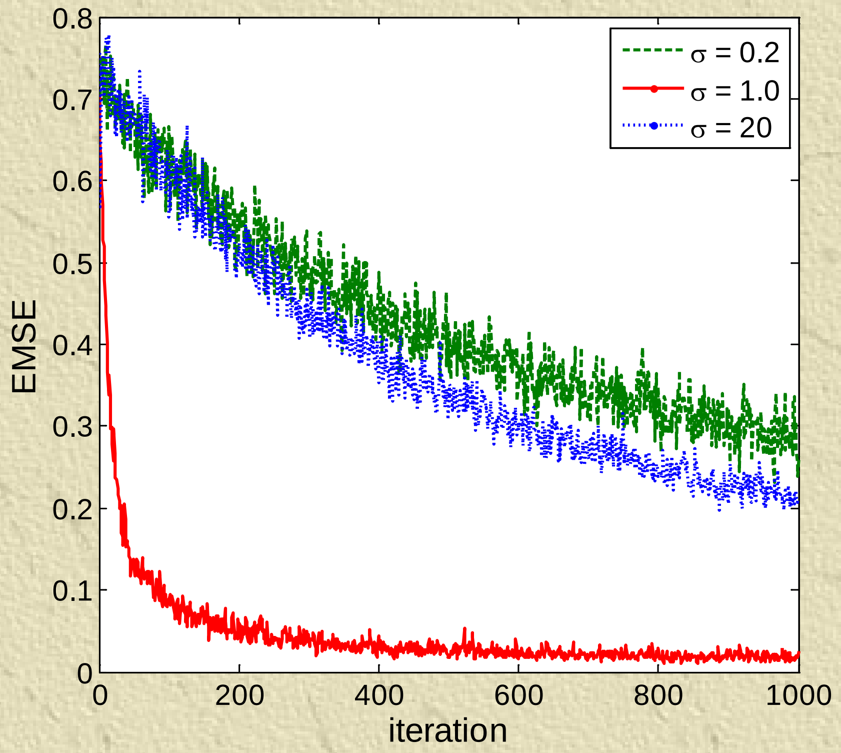
- Steady-state mean square performance

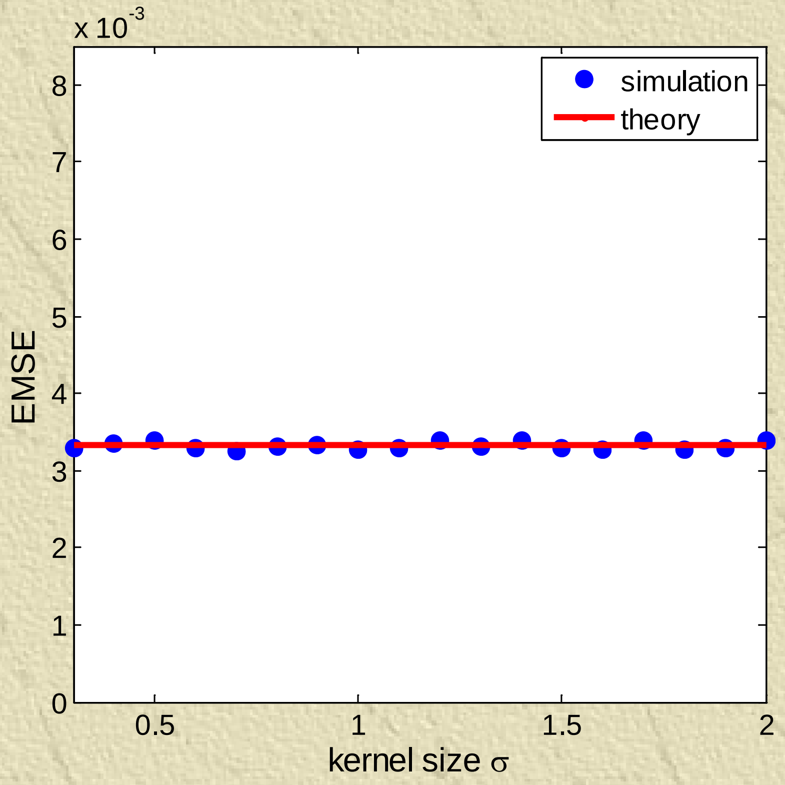
$$\lim_{i\to\infty} E\left[e_a^2(i)\right] = \frac{\eta\sigma_v^2}{2-\eta}$$



Chen B., Zhao S., Zhu P., Principe J. **Mean Square Convergence Analysis of the Kernel Least Mean Square Algorithm**, submitted to IEEE Trans. Signal Processing

Effects of Kernel Size



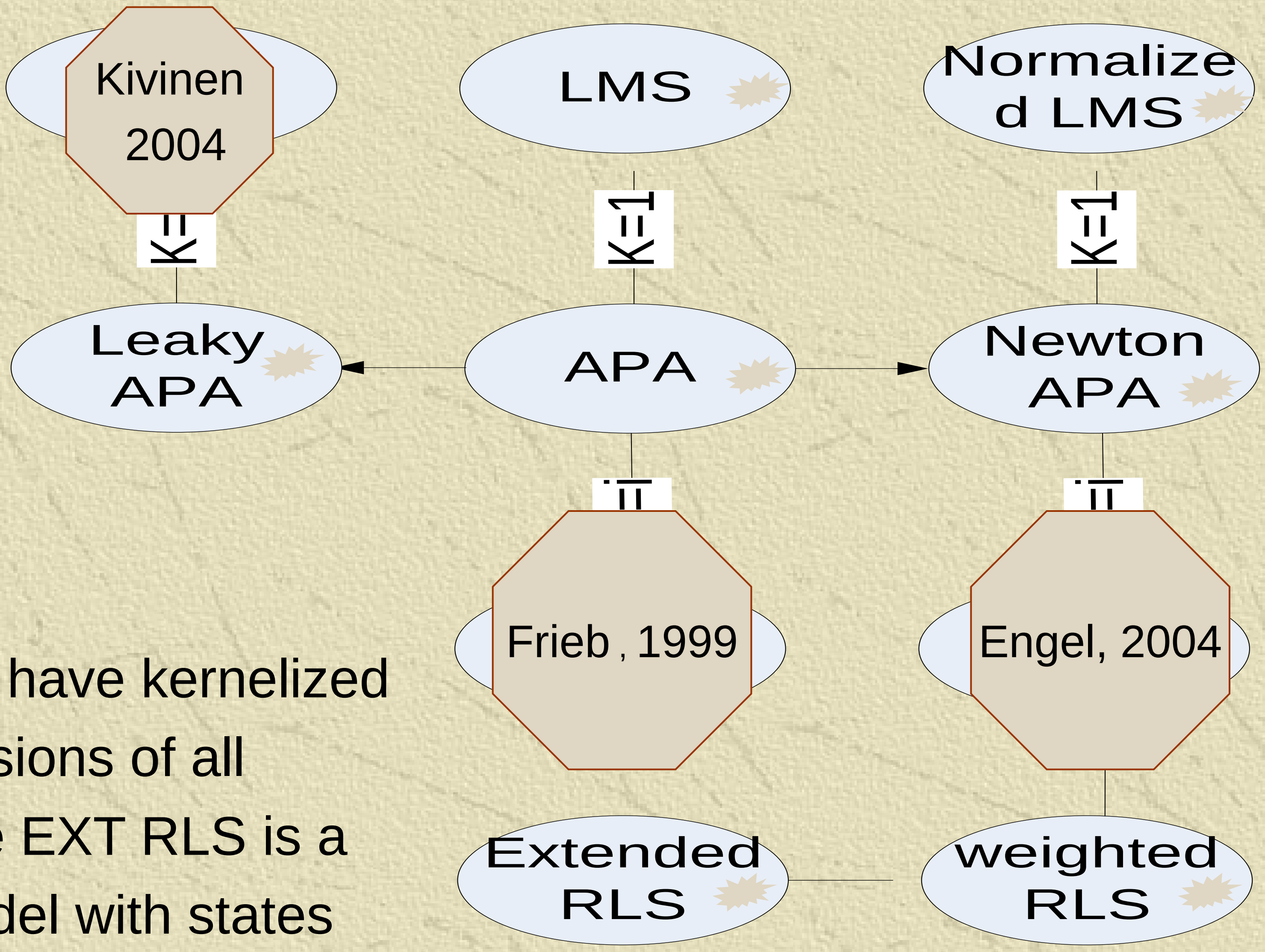


- Kernel size affects the convergence speed! (How to choose a suitable kernel size is still an open problem)

- However, it does not affect the final misadjustment! (universal approximation with infinite samples)

Part 3: Affine projection algorithms in kernel space

The big picture for gradient based learning



We have kernelized versions of all

The EXT RLS is a model with states

Liu W., Principe J., “Kernel Affine Projection Algorithms”, European J. of Signal Processing, ID 784292, 2008.

Affine projection algorithms

- Solve $\min_{w} J(\mathbf{w}) = E\left|d - \mathbf{w}^T\mathbf{u}\right|^2$ which yields $\mathbf{w}^0 = \mathbf{R_u^{-1} r_{du}}$
- There are several ways to approximate this solution iteratively using
  - Gradient Descent Method

$$\mathbf{w}(0) \qquad \mathbf{w}(i) = \mathbf{w}(i-1) + \eta[\mathbf{r_{du}} - \mathbf{R_u}\mathbf{w}(i-1)]$$

  - Newton's recursion

$$\mathbf{w}(0) \qquad \mathbf{w}(i) = \mathbf{w}(i-1) + \eta(\mathbf{R_u} + \varepsilon\mathbf{I})^{-1}[\mathbf{r_{du}} - \mathbf{R_u}\mathbf{w}(i-1)]$$

- LMS uses a stochastic gradient that approximates

$$\hat{\mathbf{R}}_\mathbf{u} = \mathbf{u}(i)\mathbf{u}(i)^T \qquad \hat{\mathbf{r}}_\mathbf{du} = d(i)\mathbf{u}(i)$$

- Affine projection algorithms (APA) utilize better approximations
- Therefore APA is a family of online gradient based algorithms of intermediate complexity between the LMS and RLS.

Affine projection algorithms

- APA are of the general form

$$\mathbf{U}(i) = [\mathbf{u}(i-K+1),\ldots,\mathbf{u}(i)]_{LxK} \qquad \mathbf{d}(i) = [d(i-K+1),\ldots,d(i)]^T$$

$$\hat{\mathbf{R}}_{\mathbf{u}} = \frac{1}{K}\mathbf{U}(i)\mathbf{U}(i)^T \qquad \hat{\mathbf{r}}_{\mathbf{du}} = \frac{1}{K}\mathbf{U}(i)\mathbf{d}(i)$$

Gradient $\mathbf{w}(0)$ $\quad \mathbf{w}(i) = \mathbf{w}(i-1) + \eta\mathbf{U}(i)[\mathbf{d}(i) - \mathbf{U}(i)^T\mathbf{w}(i-1)]$

Newton

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

- Notice that

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

- So

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

Affine projection algorithms

- If a regularized cost function is preferred

$$\min_{w} J(\mathbf{w}) = E\left|d - \mathbf{w}^T\mathbf{u}\right|^2 + \lambda\left\|\mathbf{w}\right\|^2$$

- The gradient method becomes

$$\mathbf{w}(0) \qquad \mathbf{w}(i) = (1-\eta\lambda)\mathbf{w}(i-1) + \eta\mathbf{U}(i)[\mathbf{d}(i) - \mathbf{U}(i)^T\mathbf{w}(i-1)]$$

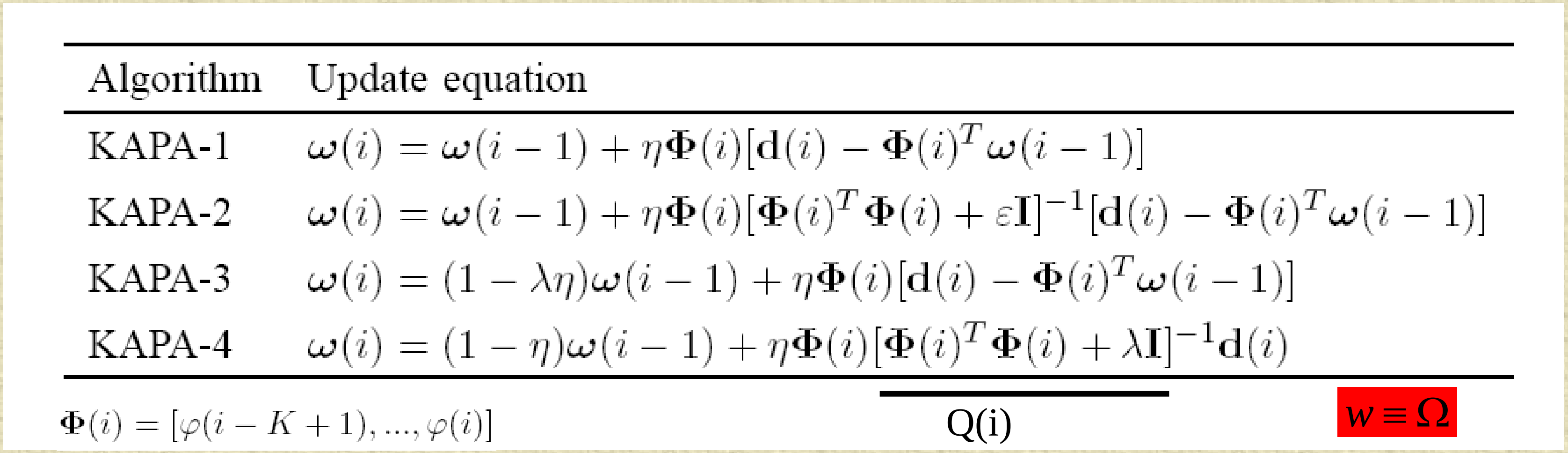
Newton

$$\mathbf{w}(i) = (1-\eta\lambda)\mathbf{w}(i-1) + \eta(\mathbf{U}(i)\mathbf{U}(i)^T + \varepsilon\mathbf{I})^{-1}\mathbf{U}(i)\mathbf{d}(i)$$

- Or

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

Kernel Affine Projection Algorithms



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

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

Q(i)

$w \equiv \Omega$

KAPA 1,2 use the least squares cost, while KAPA 3,4 are regularized

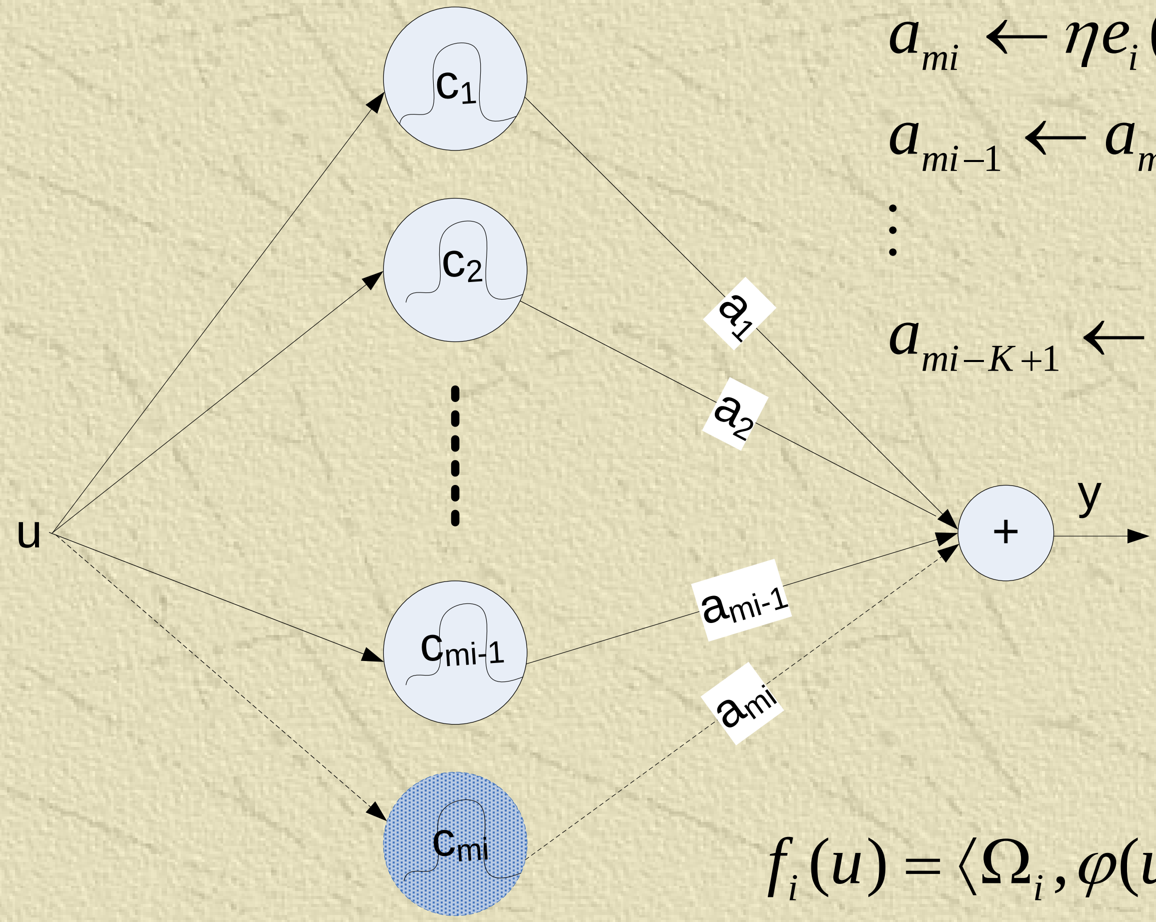
KAPA 1,3 use gradient descent and KAPA 2,4 use Newton update

Note that KAPA 4 does not require the calculation of the error by rewriting the error with the matrix inversion lemma and using the kernel trick

Note that one does not have access to the weights, so need recursion as in KLMS.

Care must be taken to minimize computations.

KAPA-1



$$c_{mi} \leftarrow u_i$$

$$a_{mi} \leftarrow \eta e_i(i)$$

$$a_{mi-1} \leftarrow a_{mi-1} + \eta e_i(i-1)$$

$$\vdots$$

$$a_{mi-K+1} \leftarrow a_{mi-K+1} + \eta e_i(i-K+1)$$

$$f_i(u) = \langle \Omega_i, \varphi(u) \rangle_F = \sum_{j=1}^{i} a_j \kappa(u, u_j)$$

KAPA-1

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

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

$$\mathbf{a}_j(i) = \mathbf{a}_j(i-1) + \eta e(i;j) \qquad j = 1-K+1,....,i-1$$

$$\mathbf{a}_j(i) = \mathbf{a}_j(i-1) \qquad j = 1,...,i-K$$

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

Error reusing to save computation

- For KAPA-1, KAPA-2, and KAPA-3
- To calculate K errors is expensive (kernel evaluations)

$$e_i(k) = d(k) - \varphi_k{}^T \Omega_{i-1}, (i - K + 1 \le k \le i)$$

- K times computations? No, save previous errors and use them

$$e_{i+1}(k) = d(k) - \varphi_k{}^T \Omega_i = d(k) - \varphi_k{}^T (\Omega_{i-1} + \eta \Phi_i e_i)$$

$$= (d(k) - \varphi_k{}^T \Omega_{i-1}) + \eta \varphi_k{}^T \Phi_i e_i$$

$$= e_i(k) + \eta \varphi_k{}^T \Phi_i e_i$$

$$= e_i(k) + \sum_{j=i-K+1}^{i} \eta e_i(j) \varphi_k{}^T \varphi_j .$$

Still needs $e_i(i+1)$

which requires i kernel evals,

So O(i+K^2)

KAPA-4

- KAPA-4: Smoothed Newton’s method.

$$\Phi_i = [\varphi_i, \varphi_{i-1}, ..., \varphi_{i-K+1}]$$

$$d_i = [d(i), d(i-1), ..., d(i-K+1)]^T$$

- There is no need to compute the error

$$\mathbf{w}(i) = (1-\eta\lambda)\mathbf{w}(i-1) + \eta\Phi(i)[\Phi(i)^T\Phi(i) + \lambda\mathbf{I}]^{-1}\mathbf{d}(i)$$

- The topology can still be put in the same RBF framework.
- Efficient ways to compute the inverse are necessary. The sliding window computation yields a complexity of O(K^2)

KAPA-4

$$\mathbf{a}_k(i) = \eta \tilde{d}(i) \qquad k = i$$

$$\mathbf{a}_k(i) = (1-\eta)\mathbf{a}_k(i-1) + \eta \tilde{d}(k) \qquad i-K+1 \le k \le i-1$$

$$\mathbf{a}_k(i) = (1-\eta)\mathbf{a}_k(i-1) \qquad 1 \le k \le i-K+1$$

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

- How to invert the K-by-K matrix $(\varepsilon I + \Phi_i^T \Phi_i)$ and avoid O(K^3)?

Sliding window Gram matrix inversion

$$\Phi_i = [\varphi_i, \varphi_{i-1}, \ldots, \varphi_{i-K+1}] \qquad Gr_i = \Phi_i^T \Phi_i$$

$$Gr_i + \lambda I = \begin{bmatrix} a & b^T \\ b & D \end{bmatrix} \xrightarrow{\text{Sliding window}} Gr_{i+1} + \lambda I = \begin{bmatrix} D & h \\ h^T & g \end{bmatrix}$$

Assume known

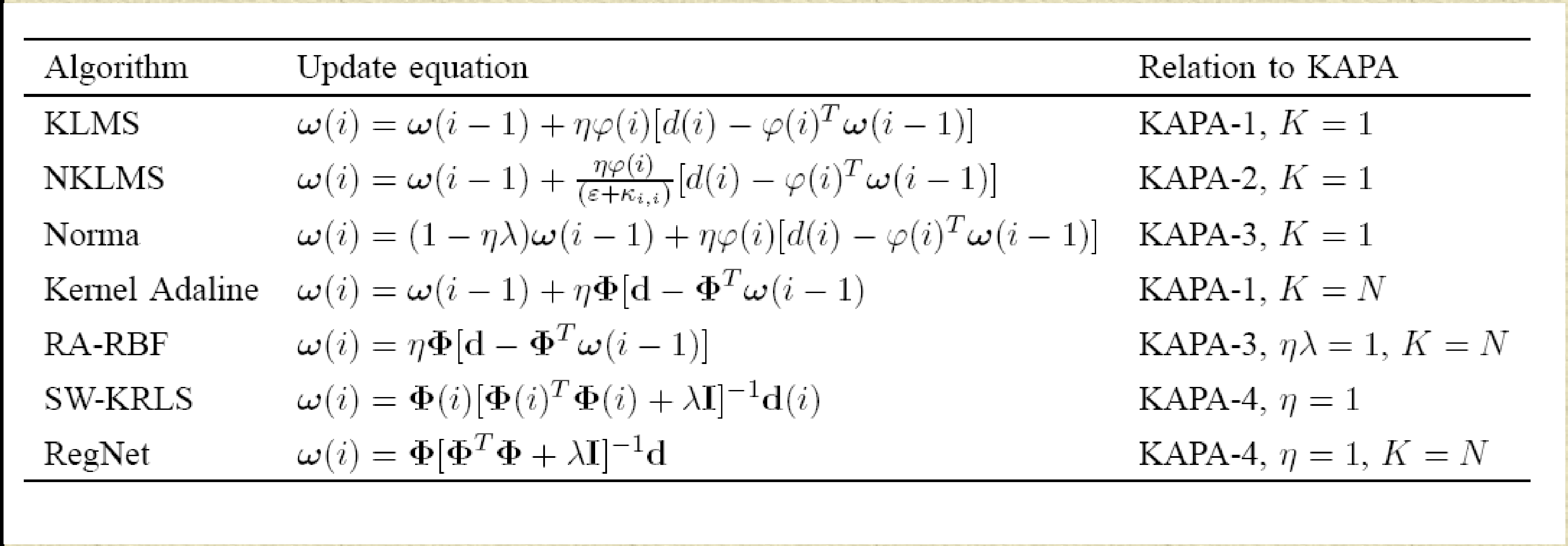
$$(Gr_i + \lambda I)^{-1} = \begin{bmatrix} e & f^T \\ f & H \end{bmatrix} \qquad D^{-1} = H - ff^T / e$$ 1

2
$$s = (g - h^T D^{-1} h)^{-1}$$ Schur complement of D

3
$$(Gr_{i+1} + \lambda I)^{-1} = \begin{bmatrix} D^{-1} + (D^{-1}h)(D^{-1}h)^T s & -(D^{-1}h)s \\ -(D^{-1}h)^T s & s \end{bmatrix}$$

Complexity is K^2

Relations to other methods



Algorithm	Update equation	Relation to KAPA
KLMS	$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\varphi(i)[d(i) - \varphi(i)^T\boldsymbol{\omega}(i-1)]$	KAPA-1, $K = 1$
NKLMS	$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \frac{\eta\varphi(i)}{(\varepsilon+\kappa_{i,i})}[d(i) - \varphi(i)^T\boldsymbol{\omega}(i-1)]$	KAPA-2, $K = 1$
Norma	$\boldsymbol{\omega}(i) = (1-\eta\lambda)\boldsymbol{\omega}(i-1) + \eta\varphi(i)[d(i) - \varphi(i)^T\boldsymbol{\omega}(i-1)]$	KAPA-3, $K = 1$
Kernel Adaline	$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}[\mathbf{d} - \boldsymbol{\Phi}^T\boldsymbol{\omega}(i-1)$	KAPA-1, $K = N$
RA-RBF	$\boldsymbol{\omega}(i) = \eta\boldsymbol{\Phi}[\mathbf{d} - \boldsymbol{\Phi}^T\boldsymbol{\omega}(i-1)]$	KAPA-3, $\eta\lambda = 1$, $K = N$
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, $\eta = 1$, $K = N$

Recursive Least-Squares

- The RLS algorithm estimates a weight vector **w**(i-1) by minimizing the cost function

$$\min_{w} \sum_{j=1}^{i-1} \left| d(j) - \mathbf{u}(j)^T \mathbf{w} \right|^2$$

- The solution becomes

$$\mathbf{w}(i-1) = (\mathbf{U}(i-1)\mathbf{U}(i-1)^T)^{-1}\mathbf{U}(i-1)\mathbf{d}(i-1)$$

And can be recursively computed as

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \frac{\mathbf{P}(i-1)\mathbf{u}(i)}{1+\mathbf{u}(i)^T\mathbf{P}(i-1)\mathbf{u}(i)}[d(i) - \mathbf{u}(i)^T\mathbf{w}(i-1)]$$

Where $\mathbf{P}(i) = (\mathbf{U}(i)\mathbf{U}(i)^T)^{-1}$. Start with zero weights and $\mathbf{P}(0) = \lambda^{-1}I$

$$r(i) = 1 + \mathbf{u}(i)^T\mathbf{P}(i-1)\mathbf{u}(i) \qquad \mathbf{w}(i) = \mathbf{w}(i-1) + \mathbf{k}(i)e(i)$$

$$\mathbf{k}(i) = \mathbf{P}(i-1)\mathbf{u}(i)/r(i) \qquad \mathbf{P}(i) = [\mathbf{P}(i-1) - \mathbf{k}(i)\mathbf{k}(i)^T r(i)]$$

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

Kernel Recursive Least-Squares

- The KRLS algorithm estimates a weight function **w**(i) by minimizing

$$\min_{w} \sum_{j=1}^{i-1} \left| d(j) - \mathbf{w}^T \varphi(j) \right|^2 + \lambda \|\mathbf{w}\|^2$$

  - The solution in RKHS becomes

$$\mathbf{w}(i) = \Phi(i)\left[\lambda I + \Phi(i)^T \Phi(i)\right]^{-1} \mathbf{d}(i) = \Phi(i)\mathbf{a}(i) \qquad \mathbf{a}(i) = \mathbf{Q}(i)\mathbf{d}(i)$$

$\mathbf{Q}^{-1}(i)$ can be computed recursively as

$$\mathbf{Q}^{-1}(i) = \begin{bmatrix} \mathbf{Q}(i-1)^{-1} & \mathbf{h}(i) \\ \mathbf{h}(i)^T & \lambda + \varphi(i))^T \varphi(i) \end{bmatrix} \qquad \mathbf{h}(i) = \Phi(i-1)^T \varphi(i)$$

From this we can also recursively compute Q(i)

$$\mathbf{Q}(i) = r(i)^{-1} \begin{bmatrix} \mathbf{Q}(i-1)r(i) + \mathbf{z}(i)^T \mathbf{z}(i) & -\mathbf{z}(i) \\ \text{-}\,\mathbf{z}(i)^T & 1 \end{bmatrix} \qquad \begin{array}{c} \mathbf{z}(i) = \mathbf{Q}(i-1)\mathbf{h}(i) \\ r(i) = \lambda + \kappa(\mathbf{u}(i), \mathbf{u}(i)) - \mathbf{z}(i)^T \mathbf{h}(i) \end{array}$$

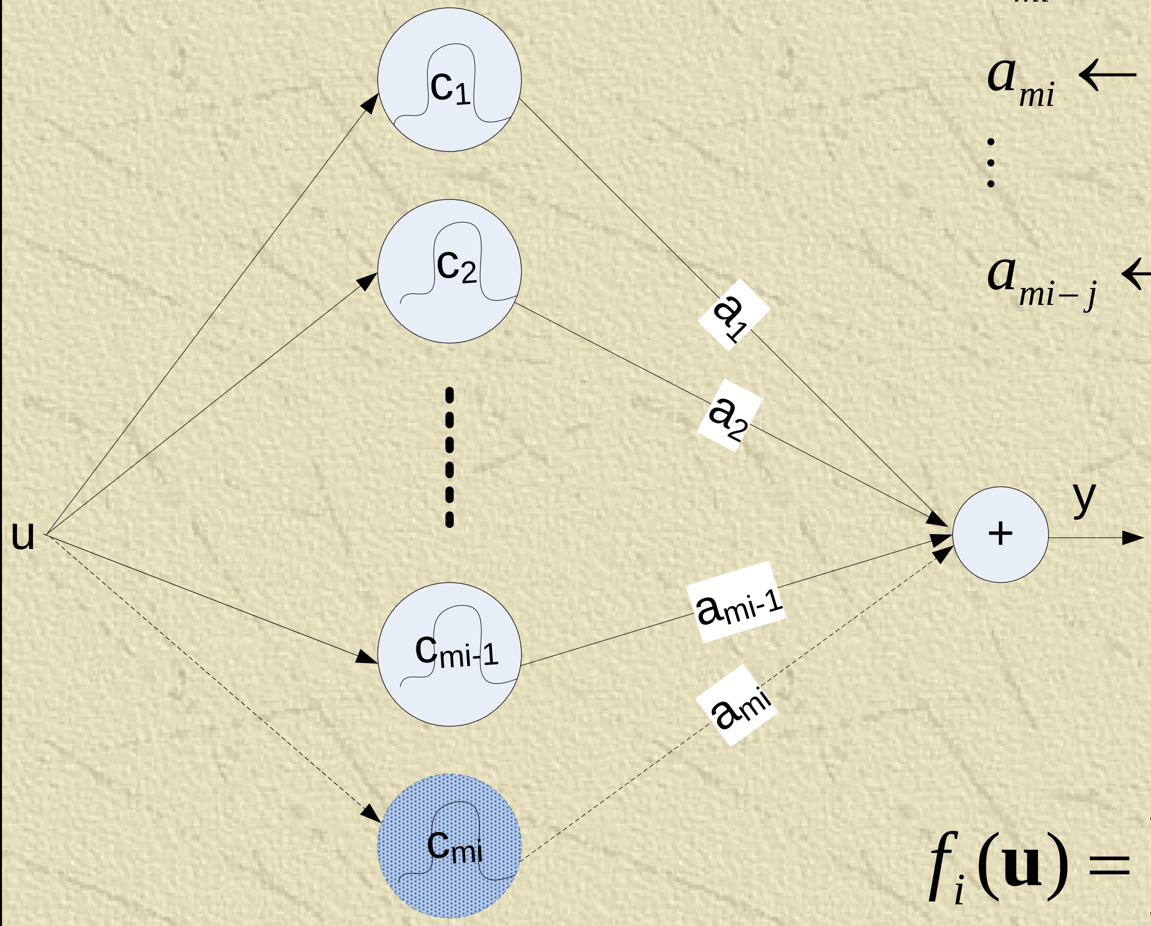
And compose back a(i) recursively

$$\mathbf{a}(i) = \begin{bmatrix} \mathbf{a}(i) - \mathbf{z}(i) r^{-1}(i) e(i) \\ r^{-1}(i) e(i) \end{bmatrix} \qquad e(i) = d(i) - \mathbf{h}(i)^T \mathbf{a}(i-1)$$

with initial conditions

$$\mathbf{Q}(1) = \left[\lambda + \kappa(\mathbf{u}(i), \mathbf{u}(i)^T)\right]^{-1}, \qquad \mathbf{a}(1) = \mathbf{Q}(1) d(1)$$

KRLS



$$c_{mi} \leftarrow u_i$$

$$a_{mi} \leftarrow r(i)^{-1} e(i)$$

$$\vdots$$

$$a_{mi-j} \leftarrow a_{mi-j} - r(i)^{-1} e(i) \mathbf{z}_j(i)$$

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

Engel Y., Mannor S., Meir R. “The kernel recursive least square algorithm”, IEEE Trans. Signal Processing, 52 (8), 2275-2285, 2004.

KRLS

$$f_i = f_{i-1} + r(i)^{-1}\left[\kappa(\mathbf{u}(i),\cdot) - \sum_{j=1}^{i-1} \mathbf{z}_j(i)\kappa(\mathbf{u}(j),\cdot)\right] e(i)$$

$$\mathbf{a}_i(i) = r(i)^{-1} e(i)$$

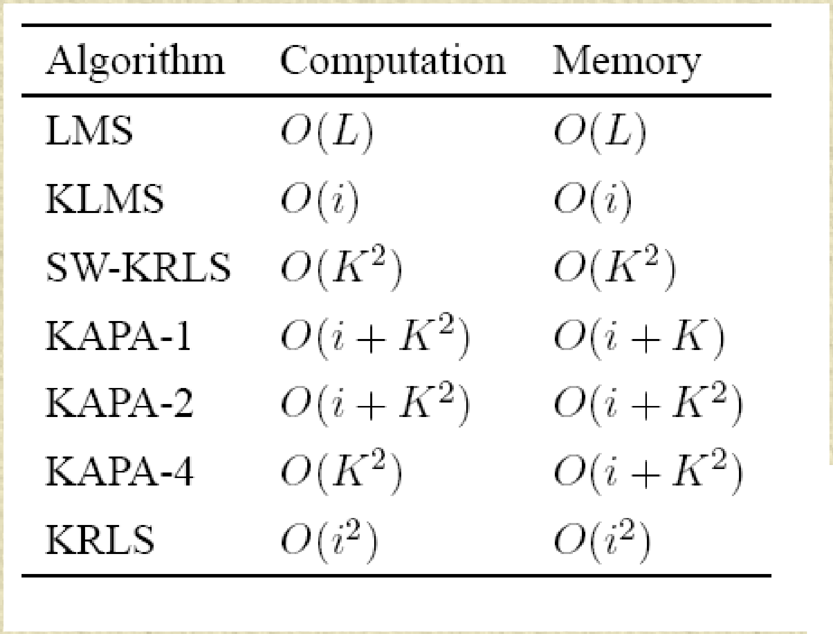
$$\mathbf{a}_j(i) = \mathbf{a}_j(i) - r(i)^{-1} e(i)\mathbf{z}_j(i) \qquad j = 1,\ldots,i-1$$

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

Regularization

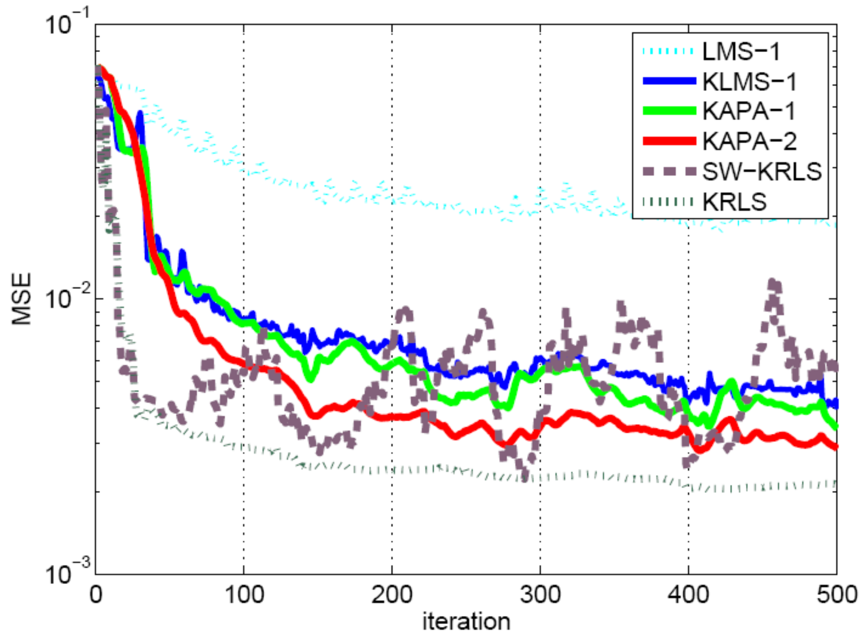
- The well-posedness discussion for the KLMS hold for any other gradient decent methods like KAPA-1 and KAPA-3
- If Newton method is used, additional regularization is needed to invert the Hessian matrix like in KAPA-2 and normalized KLMS
- Recursive least squares embed the regularization in the initialization

Computation complexity



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	$O(i^2)$	$O(i^2)$

Prediction of Mackey-Glass



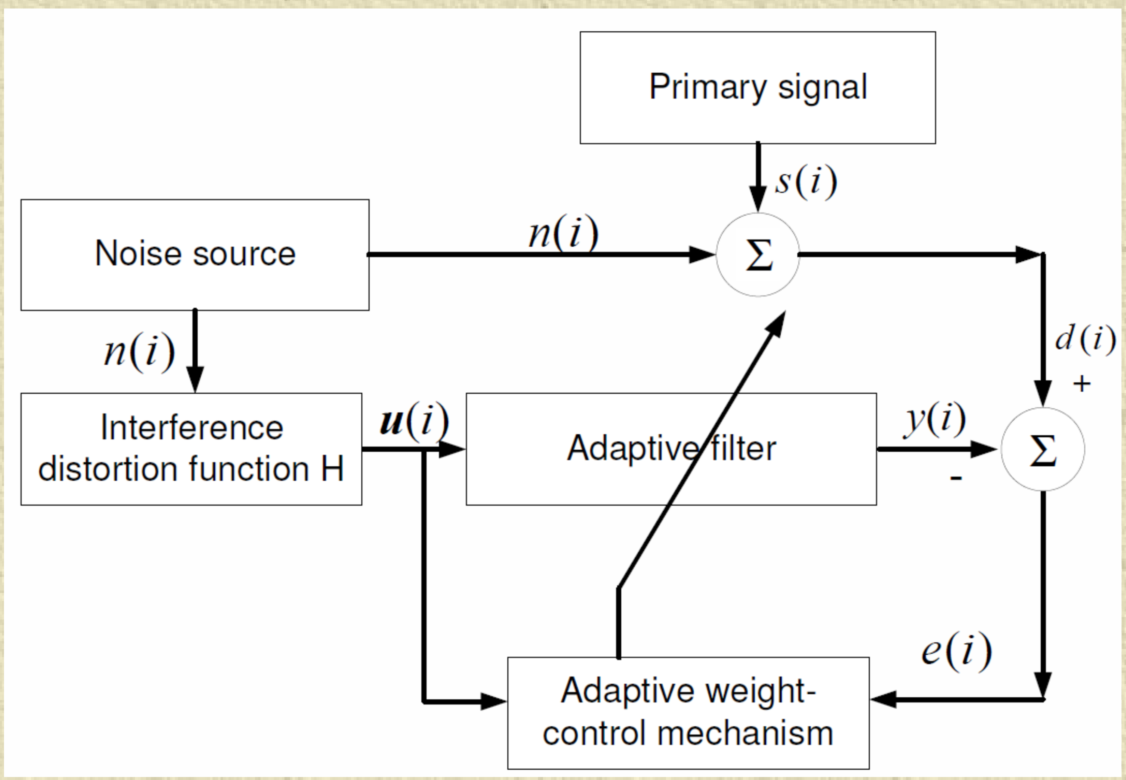
L=10

K=10

K=50 SW KRLS

Simulation 1: noise cancellation

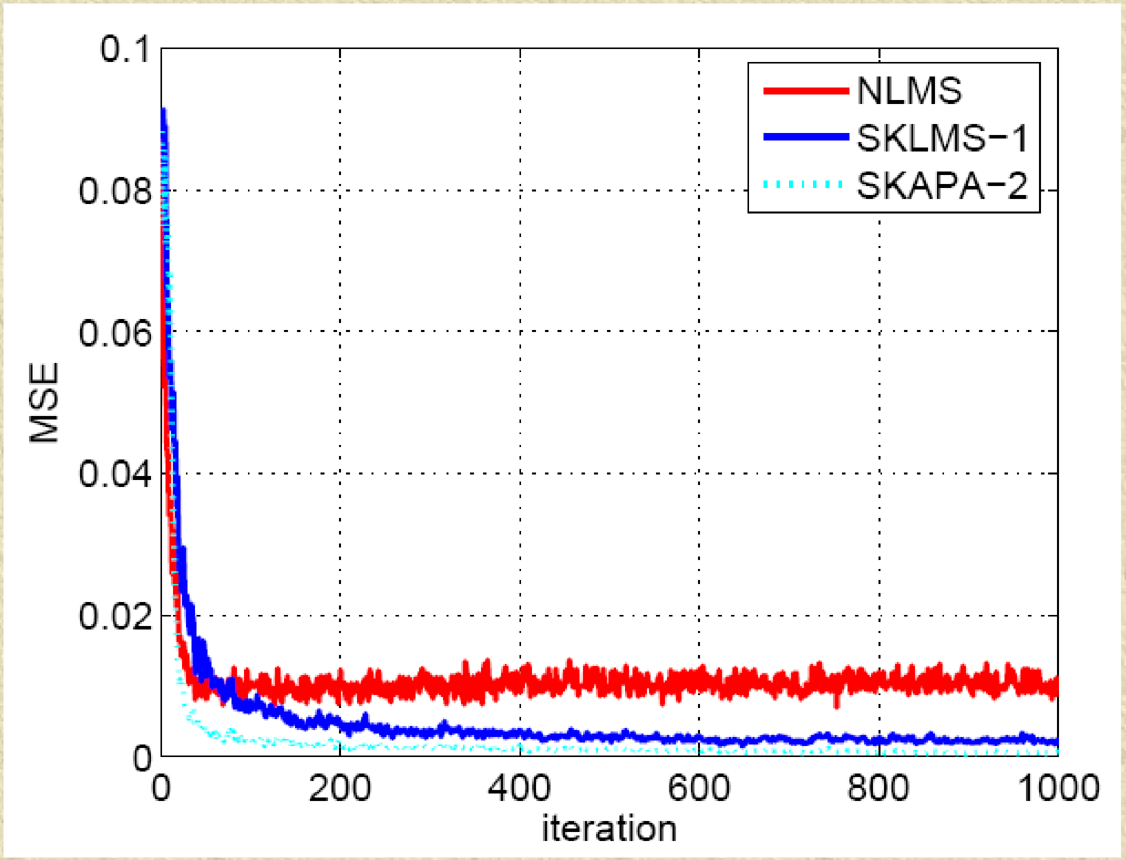
n(i) ~ uniform [-0.5, 05]



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

$$= H(n(i), n(i-1), u(i-1), u(i-2))$$

Simulation 1: Noise Cancellation

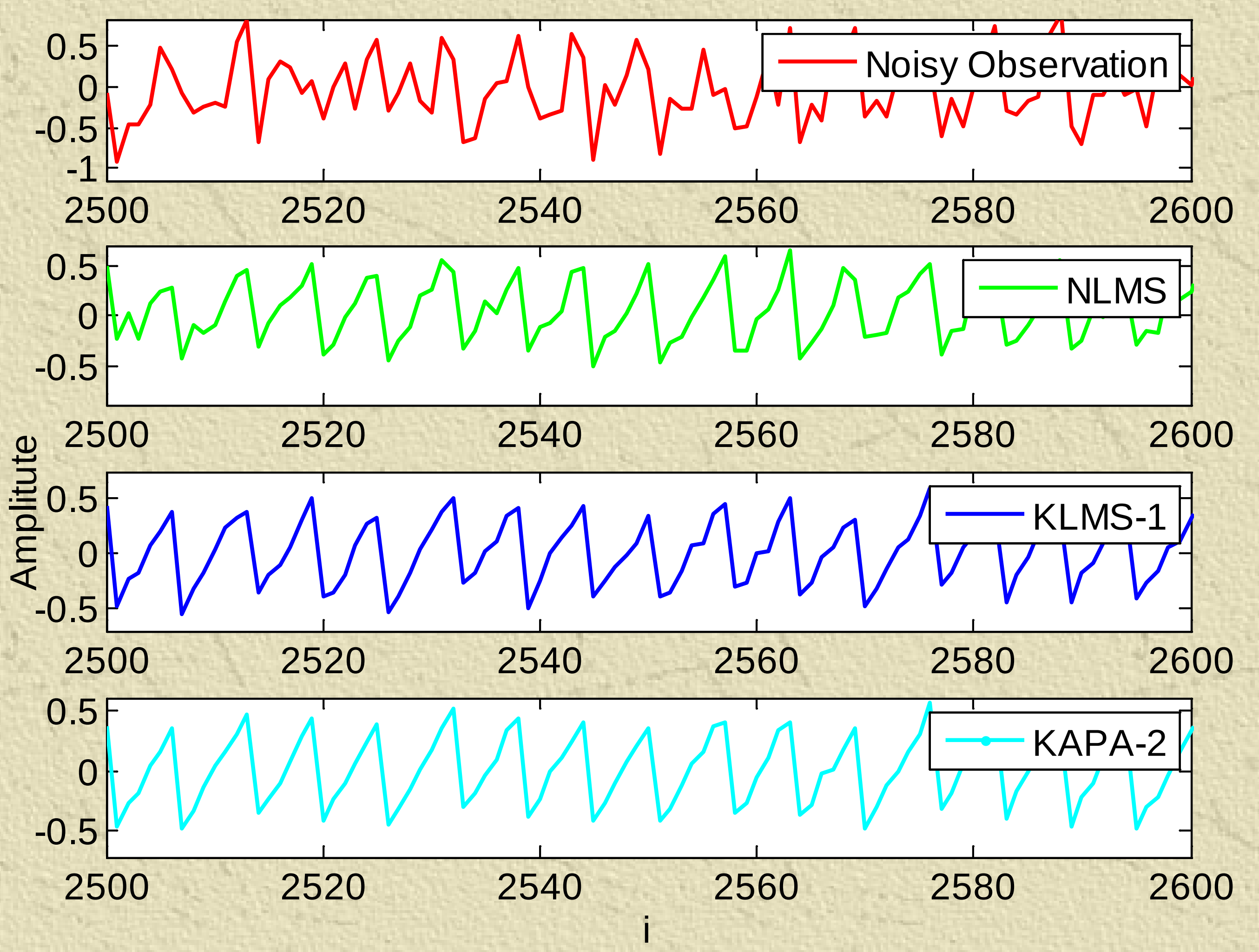


Algorithm	Network Size	NR(dB)
NLMS	N/A	9.09±0.45
SKLMS-1	407±14	15.58±0.48
SKAPA-2	370±14	21.99±0.80

$$\kappa(u(i),u(j)) = \exp(-\| u(i)-u(j) \|^2)$$

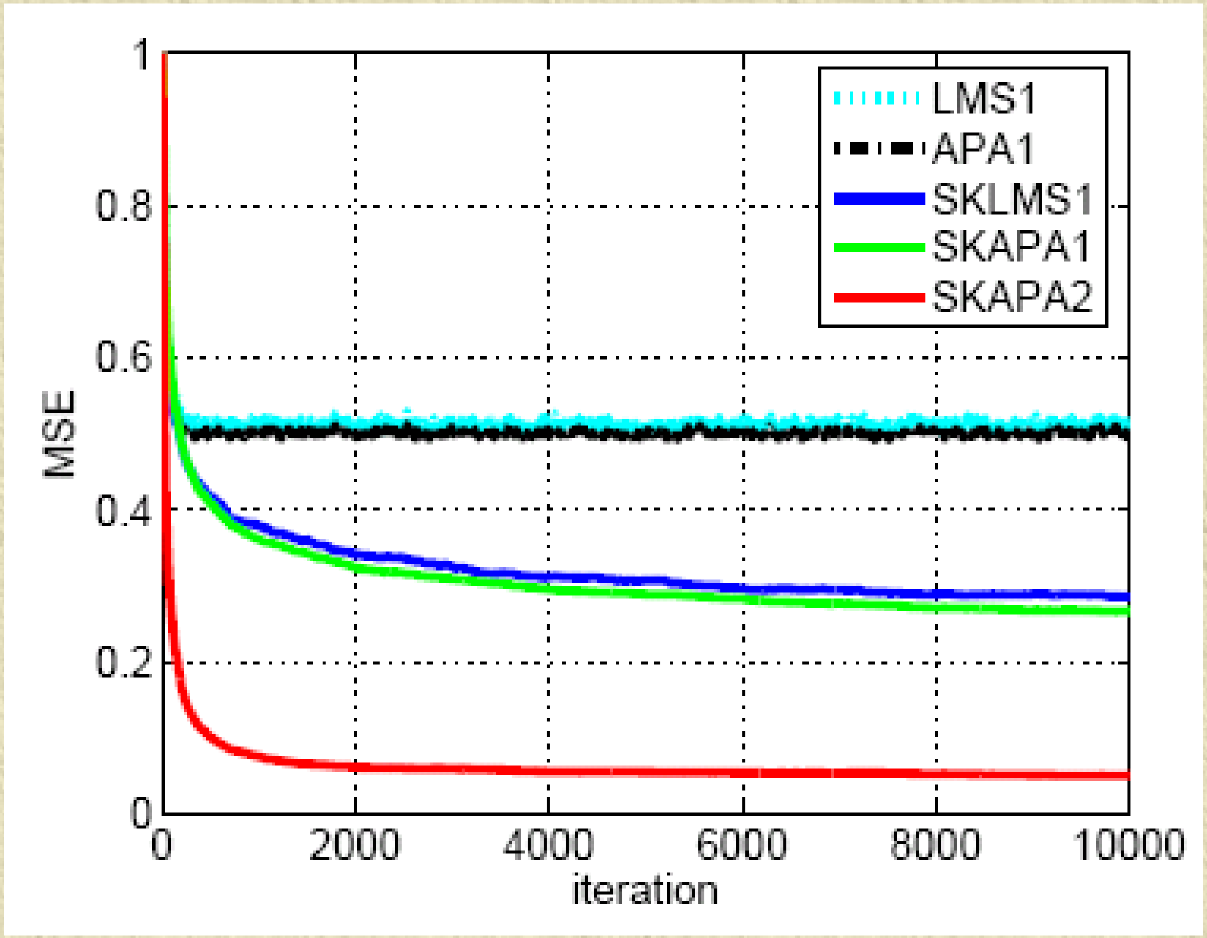
K=10

Simulation 1:Noise Cancellation



Simulation-2: nonlinear channel equalization

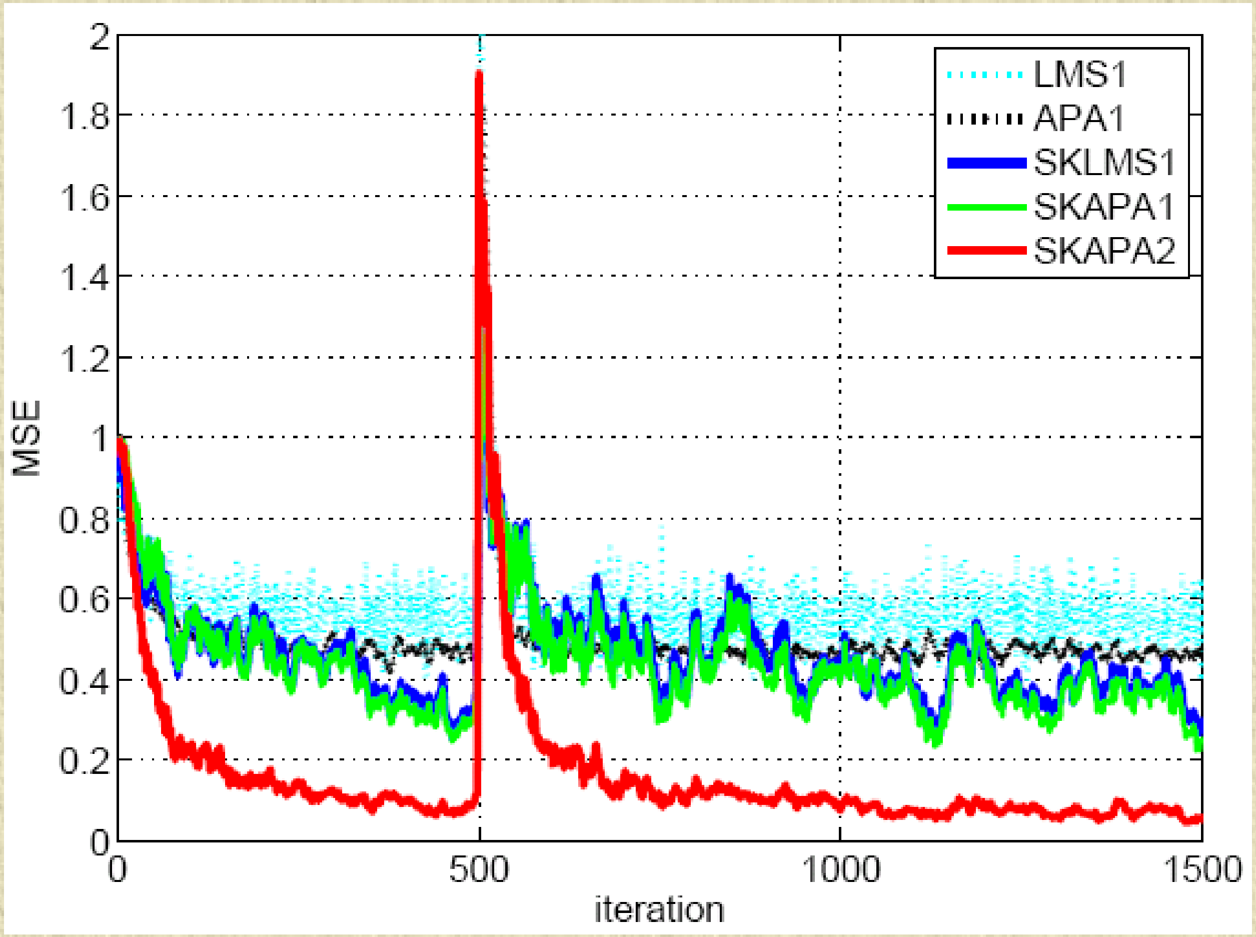
$$\begin{tikzcd} {} \arrow[r, "s_t"] & \boxed{z_t = s_t + 0.5 s_{t-1}} \arrow[r] & \boxed{r_t = z_t - 0.9 z_t^{\,2} + n_\sigma} \arrow[r, "r_t"] & {} \end{tikzcd}$$



K=10

σ=0.1

Simulation-2: nonlinear channel equalization



↑

Nonlinearity changed (inverted signs)

Gaussian Processes

- A Gaussian process is a stochastic process (a family of random variables) where all the pairwise correlations are Gaussian distributed. The family however is not necessarily over time (as in time series).
- For instance in regression, if we denote the output of a learning system by y(i) given the input u(i) for every i, the conditional probability

$$p(y(1),...y(n) \mid u(1),...,u(n) = \mathrm{N}(0, \sigma_n^2 I + G(i))$$

Where σ is the observation Gaussian noise and G(i) is the Gram matrix

$$G(i) = \begin{bmatrix} \kappa(\mathbf{u}(1),\mathbf{u}(1)) & \cdots & \kappa(\mathbf{u}(1),\mathbf{u}(i)) \\ \cdots & \cdots & \cdots \\ \kappa(\mathbf{u}(i),\mathbf{u}(1)) & \cdots & \kappa(\mathbf{u}(i),\mathbf{u}(i)) \end{bmatrix}$$

and κ is the covariance function (symmetric and positive definite). Just like the Gaussian kernel used in KLMS.

Gaussian processes can be used with advantage in Bayesian inference.

Gaussian Processes and Recursive Least-Squares

- The standard linear regression model with Gaussian noise is

$$f(\mathbf{u}) = \mathbf{u}^T \mathbf{w} \quad , d = f(\mathbf{u}) + \nu$$

where the noise is IID, zero mean and variance σ_n^2

- The likelihood of the observations given the input and weight vector is

$$p(\mathbf{d}(i) \mid \mathbf{U}(i), \mathbf{w}) = \prod_{j=1}^{i} p(d(j) \mid \mathbf{u}(j), \mathbf{w}) = \mathrm{N}(\mathbf{U}(i)^T \mathbf{w}, \boldsymbol{\sigma}_{\mathbf{n}}^{\mathbf{2}} I)$$

- To compute the posterior over the weight vector we need to specify the prior, here a Gaussian and use Bayes rule

$$p(w) = \mathrm{N}(0, \sigma_w^2 I) \qquad p(\mathbf{w} \mid \mathbf{U}(i), \mathbf{d}(i)) = \frac{p(\mathbf{d}(i) \mid \mathbf{U}(i), \mathbf{w})\, p(\mathbf{w})}{p(\mathbf{d}(i) \mid \mathbf{U}(i))}$$

Since the denominator is a constant, the posterior is shaped by the numerator, and it is approximately given by

$$p(w \mid U, d) \propto \exp\left[-\frac{1}{2} (\mathbf{w} - \mathbf{w}(i))^T \left(\frac{1}{\sigma_n^2} \mathbf{U}(i)\mathbf{U}(i)^T + \sigma_w^{\mathbf{2}} I \right) (\mathbf{w} - \mathbf{w}(i)) \right]$$

with mean $\mathbf{w}(i) = \left(\mathbf{U}(i)\mathbf{U}(i)^T + \sigma_n^2 \sigma_w^{\mathbf{2}} I \right)^{-1} \mathbf{U}(i)\mathbf{d}(i)$ and covariance $\left({}^{1}\!/_{\sigma_n^2}\, \mathbf{U}(i)\mathbf{U}(i)^T + \sigma_w^{\mathbf{2}} I \right)^{-1}$

Therefore, RLS computes the posterior in a Gaussian process one sample at a time.

KRLS and Nonlinear Regression

- It easy to demonstrate that KRLS does in fact estimate online nonlinear regression with a Gaussian noise model i.e.

$$f(\mathbf{u}) = \varphi(\mathbf{u})^T \mathbf{w} \quad , d = f(\mathbf{u}) + \nu$$

where the noise is IID, zero mean and variance σ_n^2

- By a similar derivation we can say that the mean and variance are

$$\mathbf{w}(i) = \left(\Phi(i)\Phi(i)^T + \sigma_n^2 \sigma_w^{\mathbf{2}} I\right)^{-1} \Phi(i)\mathbf{d}(i) \qquad \left({}^{1}\!/\!{}_{\sigma_n^2} \Phi(i)\Phi(i)^T + \sigma_w^{\mathbf{2}} I \right)^{-1}$$

- Although the weight function is not accessible we can create predictions at any point in the space by the KRLS as

$$\hat{E}[f(\mathbf{u})] = \varphi(\mathbf{u})^T \Phi(i)\left(\Phi(i)\Phi(i)^T + \sigma_n^2 \sigma_w^{\mathbf{2}} I\right)^{-1} \mathbf{d}(i)$$

with variance

$$\sigma^2(f(\mathbf{u})) = \sigma_w^2 \varphi(\mathbf{u})^T \varphi(\mathbf{u}) - \sigma_w^2 \varphi(\mathbf{u})^T \Phi(i)\left(\Phi(i)\Phi(i)^T + \sigma_n^2 \sigma_w^{\mathbf{2}} I\right)^{-1} \Phi(i)^T \varphi(\mathbf{u})$$

Part 4: Extended Recursive least squares in kernel space

Extended Recursive Least-Squares

- STATE model

$$x_{i+1} = F_i x_i + n_i$$

$$d_i = U_i^T x_i + v_i$$

  - Start with

$$w_{0|-1}, P_{0|-1} = \Pi^{-1}$$

  - Special cases
    - Tracking model (F is a time varying scalar)

$$x_{i+1} = \alpha x_i + n_i, d(i) = u_i^T x_i + v(i)$$

    - Exponentially weighted RLS

$$x_{i+1} = \alpha x_i, d(i) = u_i^T x_i + v(i)$$

    - standard RLS

$$x_{i+1} = x_i, d(i) = u_i^T x_i + v(i)$$

Notations:

x_i state vector at time i

$w_{i|i-1}$ state estimate at time i using data up to i-1

Recursive equations

- The recursive update equations

$$w_{0|-1} = 0, P_{0|-1} = \lambda^{-1}\beta^{-1}\mathrm{I}$$

Conversion factor

$$r_e(i) = \lambda^i + u_i^T P_{i|i-1} u_i$$

gain factor

$$k_{p,i} = \alpha P_{i|i-1} u_i / r_e(i)$$

error

$$e(i) = d(i) - u_i^T w_{i|i-1}$$

weight update

$$w_{i+1|i} = \alpha w_{i|i-1} + k_{p,i} e(i)$$

$$P_{i+1|i} = |\alpha|^2 [P_{i|i-1} - P_{i|i-1} u_i u_i^T P_{i|i-1} / r_e(i)] + \lambda^i q \mathrm{I}$$

- Notice that

$$u^T \hat{w}_{i+1|i} = \alpha u^T w_{i|i-1} + \alpha u^T P_{i|i-1} u_i e(i) / r_e(i)$$

If we have transformed data, how to calculate $\varphi(u_k)^T P_{i|i-1} \varphi(u_j)$ for any *k*, *i*, *j*?

New Extended Recursive Least-squares

- Theorem 1: $P_{j|j-1} = \rho_{j-1}\mathrm{I} - H_{j-1}^T Q_{j-1} H_{j-1}, \forall j$

  where ρ_{j-1} is a scalar, $H_{j-1} = [u_0, \ldots, u_{j-1}]^T$ and Q_{j-1} is a *j*x*j* matrix, for all *j*.

- Proof:

$$P_{0|-1} = \lambda^{-1}\beta^{-1}\mathrm{I}, \quad \rho_{-1} = \lambda^{-1}\beta^{-1}, \quad Q_{-1} = 0$$

$$P_{i+1|i} = |\alpha|^2 [P_{i|i-1} - \frac{P_{i|i-1}u_i u_i^T P_{i|i-1}}{r_e(i)}] + \lambda^i q\mathrm{I}$$

$$= |\alpha|^2 [\rho_{i-1} - H_{i-1}^T Q_{i-1} H_{i-1} -$$

$$\frac{(\rho_{i-1} - H_{i-1}^T Q_{i-1} H_{i-1}) u_i u_i^T (\rho_{i-1} - H_{i-1}^T Q_{i-1} H_{i-1})}{r_e(i)}] + \lambda^i q\mathrm{I}$$

$$= (|\alpha|^2 \rho_{i-1} + \lambda^i q)\mathrm{I} - |\alpha|^2 H_i^T \begin{pmatrix} Q_{i-1} + f_{i-1,i} f_{i-1,i}{}^T r_e{}^{-1}(i) & -\rho_{i-1} f_{i-1,i} r_e{}^{-1}(i) \\ -\rho_{i-1} f_{i-1,i}{}^T r_e{}^{-1}(i) & \rho^2{}_{i-1} r_e{}^{-1}(i) \end{pmatrix} H_i$$

By mathematical induction!

Liu W., Principe J., "Extended Recursive Least Squares in RKHS", in Proc. 1st Workshop on Cognitive Signal Processing, Santorini, Greece, 2008.

New Extended Recursive Least-squares

- Theorem 2: $\hat{w}_{j|j-1} = H_{j-1}^T a_{j|j-1}, \forall j$

  where $H_{j-1} = [u_0, \ldots, u_{j-1}]^T$ and $a_{j|j-1}$ is a $j \times 1$ vector, for all *j*.

- Proof:

$$\hat{w}_{0|-1} = 0, \quad a_{0|-1} = 0$$

$$\begin{aligned}
\hat{\hat{w}}_{i+1|i} &= \alpha w_{i|i-1} + k_{p,i} e(i) \\
&= \alpha H_{i-1}^T a_{i|i-1} + \alpha P_{i|i-1} u_i e(i) / r_e(i) \\
&= \alpha H_{i-1}^T a_{i|i-1} + \alpha (\rho_{i-1} \mathrm{I} - H_{i-1}^T Q_{i-1} H_{i-1}) u_i e(i) / r_e(i) \\
&= \alpha H_{i-1}^T a_{i|i-1} + \alpha \rho_{i-1} u_i e(i) / r_e(i) - \alpha H_{i-1}^T f_{i-1,i} e(i) / r_e(i) \\
&= H_i^T \begin{pmatrix} \alpha a_{i|i-1} - \alpha f_{i-1,i} e(i) r_e^{-1}(i) \\ \alpha \rho_{i-1} e(i) r_e^{-1}(i) \end{pmatrix}
\end{aligned}$$

By mathematical induction again!

Extended RLS	New Equations
$w_{0\mid-1}=0, P_{0\mid-1}=\lambda^{-1}\beta^{-1}\mathrm{I}$	$a_{0\mid-1}=0, \rho_{-1}=\lambda^{-1}\beta^{-1}, Q_{-1}=0$
	$k_{i-1,i}=u_i^T H_{i-1}^T$ $f_{i-1,i}=Q_{i-1}k_{i-1,i}$
$r_e(i)=\lambda^i+u_i^T P_{i\mid i-1}u_i$ $k_{p,i}=\alpha P_{i\mid i-1}u_i/r_e(i)$ $e(i)=d(i)-u_i^T w_{i\mid i-1}$ $w_{i+1\mid i}=\alpha w_{i\mid i-1}+k_{p,i}e(i)$ $P_{i+1\mid i}=\lvert\alpha\rvert^2[P_{i\mid i-1}-$ $P_{i\mid i-1}u_i u_i^T P_{i\mid i-1}/r_e(i)]+\lambda^i q\mathrm{I}$	$r_e(i)=\lambda^i+\rho_{i-1}u_i^T u_i-k_{i-1,i}^T f_{i-1,i}$ $e(i)=d(i)-k_{i-1,i}^T a_{i\mid i-1}$ $a_{i+1\mid i}=\alpha\begin{pmatrix} a_{i\mid i-1}-f_{i-1,i}r_e^{-1}(i)e(i) \\ \rho_{i-1}r_e^{-1}(i)e(i)\end{pmatrix}$ $\rho_i=\lvert\alpha\rvert^2\rho_{i-1}+\lambda^i q$ $Q_i=\lvert\alpha\rvert^2\begin{pmatrix} Q_{i-1}+f_{i-1,i}f_{i-1,i}^T r_e^{-1}(i) & -\rho_{i-1}f_{i-1,i}r_e^{-1}(i) \\ -\rho_{i-1}f_{i-1,i}^T r_e^{-1}(i) & \rho_{i-1}^2 r_e^{-1}(i)\end{pmatrix}$

An important theorem

- Assume a general nonlinear state-space model

$$s(i+1) = g(s(i))$$

$$d(i) = h(\mathbf{u}(i), \mathbf{s}(i)) + v(i)$$

$$\Longleftrightarrow$$

- There exists a transformed input vector $\varphi(u)$, a transformed state vector $x(s)$

$$x(\mathbf{s}(i+1)) = \mathbf{A}x(\mathbf{s}(i))$$

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

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

Extended Kernel Recursive Least-squares

$$a_{0|-1} = 0, \rho_{-1} = \lambda^{-1}\beta^{-1}, Q_{-1} = 0$$

- Initialize

$$k_{i-1,i} = [\kappa(u_0, u_i), \ldots, \kappa(u_{i-1}, u_i)]^T$$

$$f_{i-1,i} = Q_{i-1} k_{i-1,i}$$

$$r_e(i) = \lambda^i + \rho_{i-1}\kappa(u_i, u_i) - k_{i-1,i}^T f_{i-1,i}$$

$$e(i) = d(i) - k_{i-1,i}^T a_{i|i-1}$$

$$a_{i+1|i} = \alpha \begin{pmatrix} a_{i|i-1} - f_{i-1,i} r_e^{-1}(i) e(i) \\ \rho_{i-1} r_e^{-1}(i) e(i) \end{pmatrix}$$

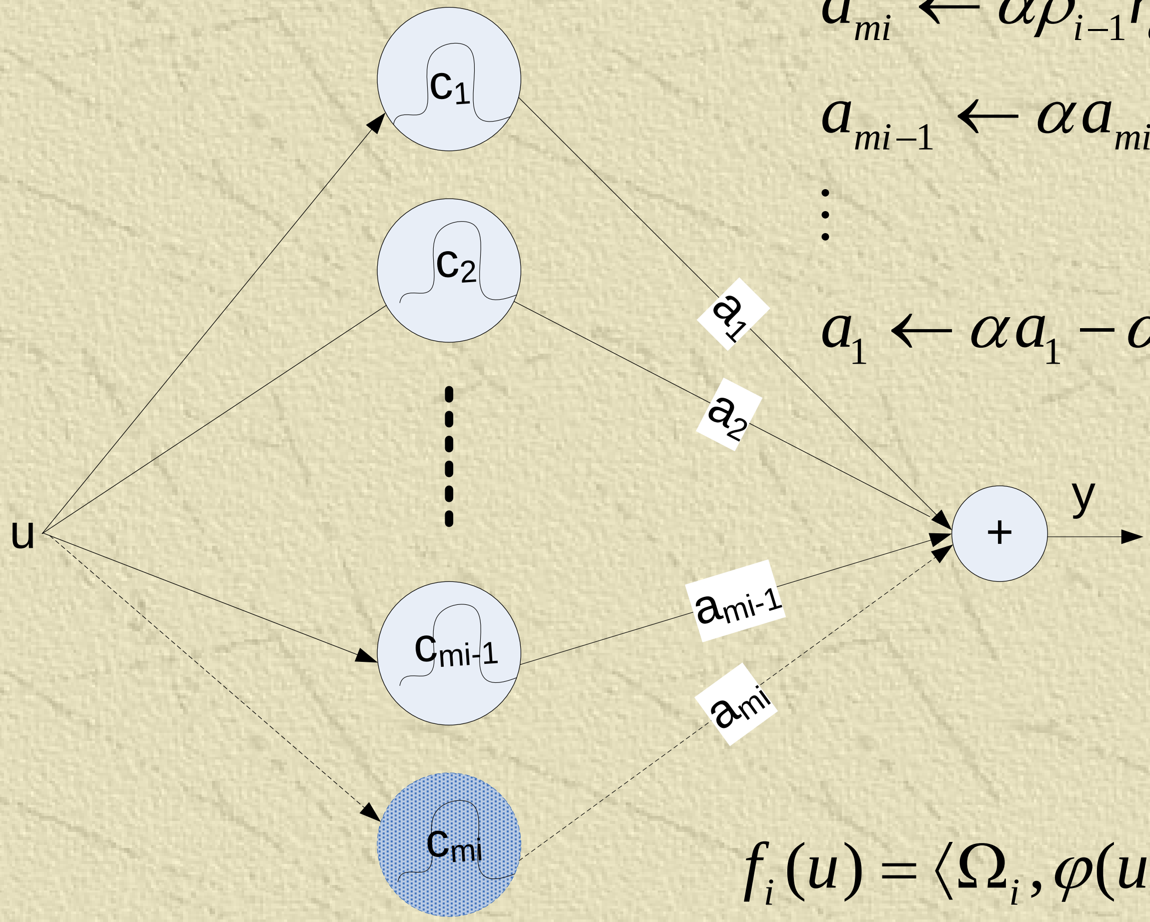
Update on weights

$$\rho_i = |\alpha|^2 \rho_{i-1} + \lambda^i q$$

$$Q_i = |\alpha|^2 \begin{pmatrix} Q_{i-1} + f_{i-1,i} f_{i-1,i}{}^T r_e^{-1}(i) & -\rho_{i-1} f_{i-1,i} r_e^{-1}(i) \\ -\rho_{i-1} f_{i-1,i}{}^T r_e^{-1}(i) & \rho^2{}_{i-1} r_e^{-1}(i) \end{pmatrix}$$

Update on P matrix

Ex-KRLS



$$c_{mi} \leftarrow u_i$$

$$a_{mi} \leftarrow \alpha \rho_{i-1} r_e^{-1}(i) e(i)$$

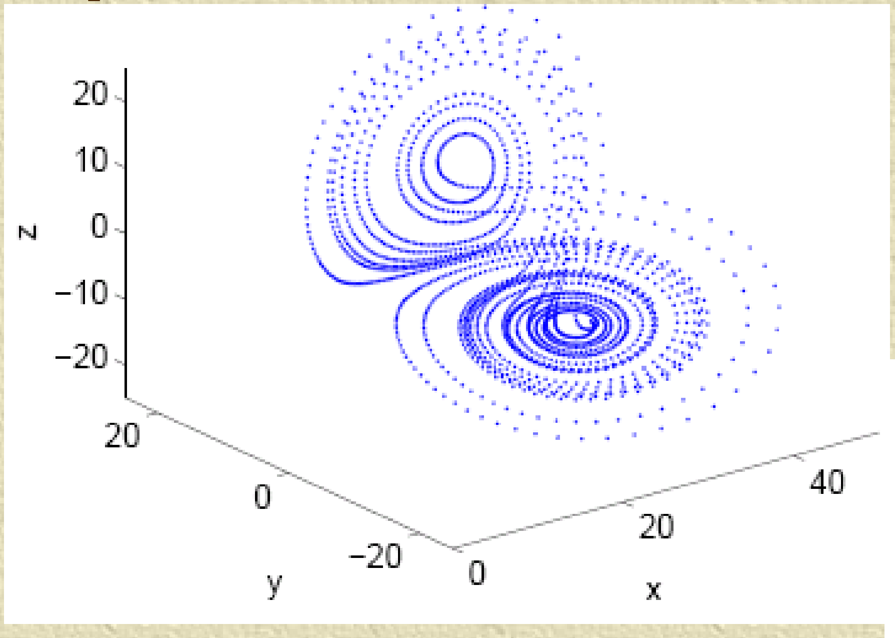
$$a_{mi-1} \leftarrow \alpha a_{mi-1} - \alpha f_{i-1,i}(i) r_e^{-1}(i) e(i)$$

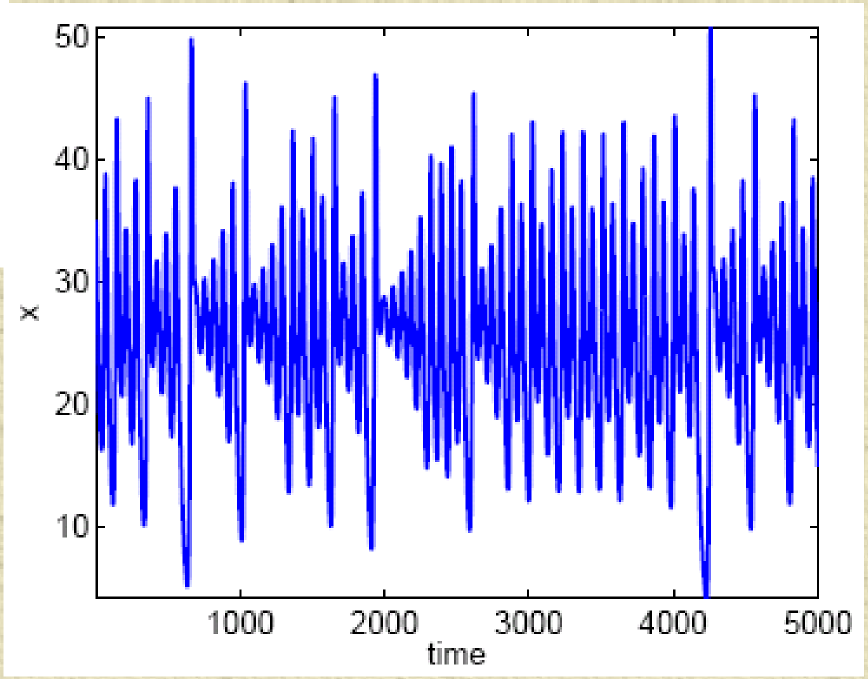
$$\vdots$$

$$a_1 \leftarrow \alpha a_1 - \alpha f_{i-1,i}(1) r_e^{-1}(i) e(i)$$

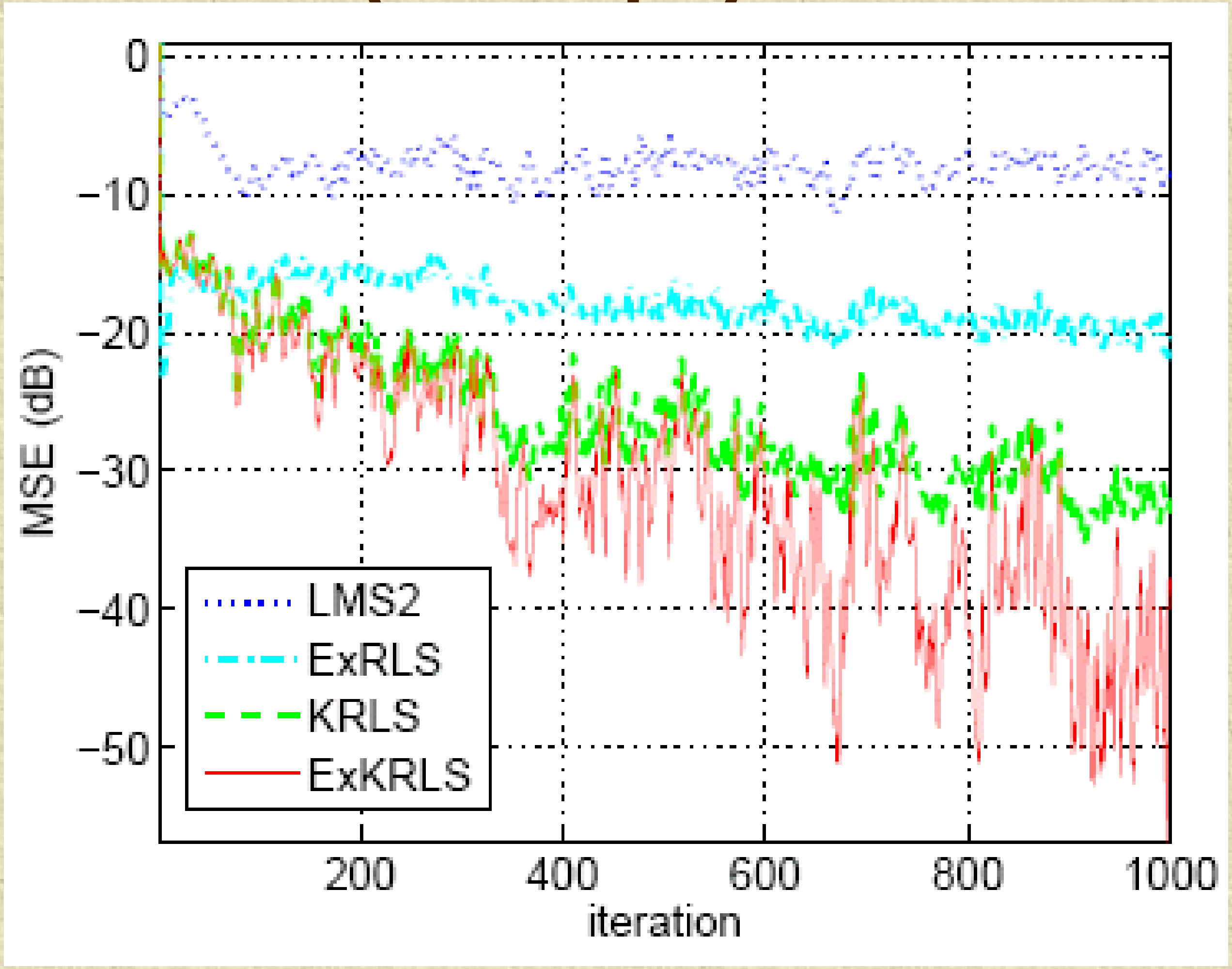
$$f_i(u) = \langle \Omega_i, \varphi(u) \rangle_F = \sum_{j=1}^{i} a_j \kappa(u, u_j)$$

Simulation-3: Lorenz time series prediction

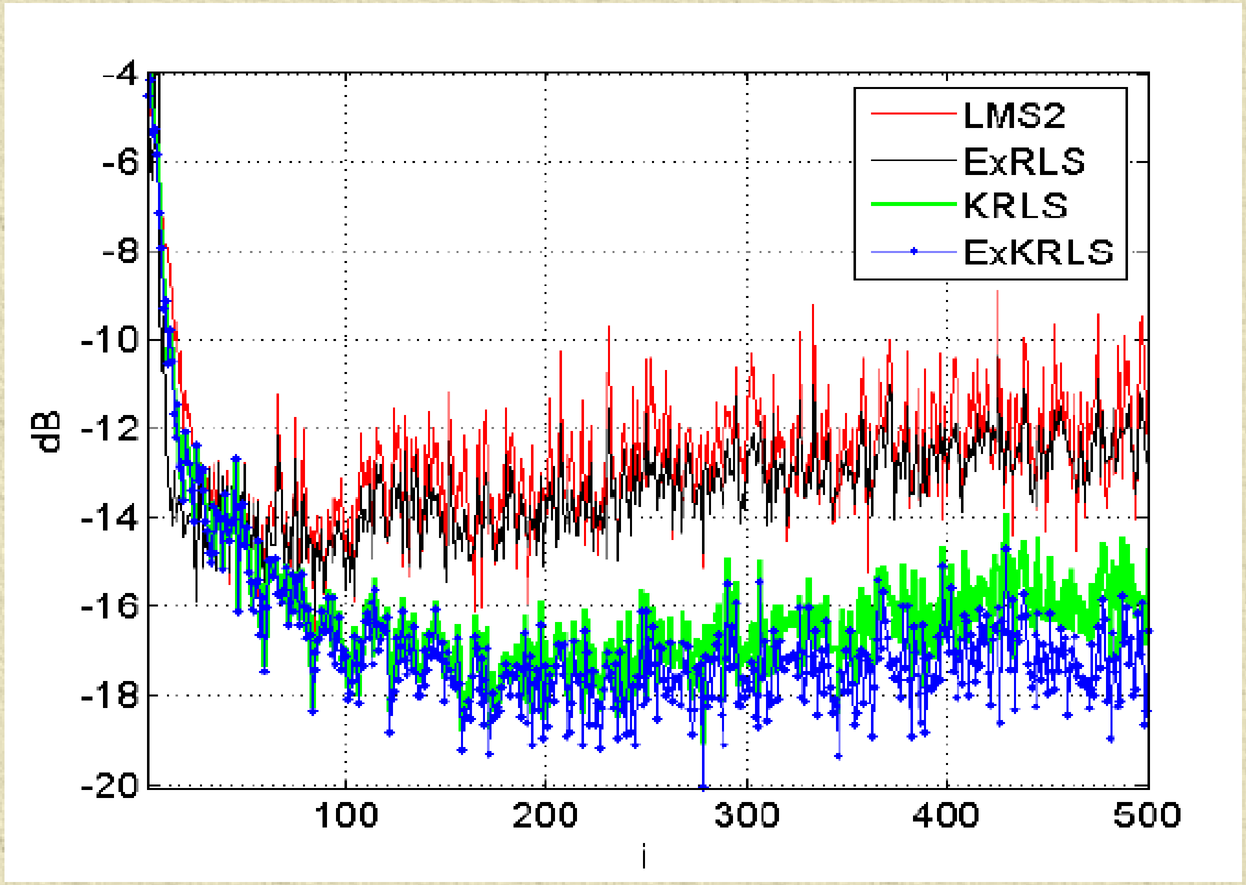




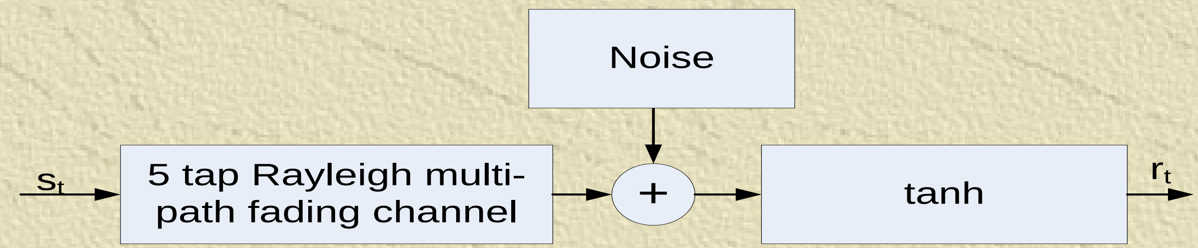
Simulation-3: Lorenz time series prediction (10 steps)



Simulation 4: Rayleigh channel tracking



1,000 symbols



f_D=100Hz, t=8x10^{-5}s σ=0.005

Rayleigh channel tracking

Algorithms	MSE (dB) (noise variance 0.001 and f_D = 50 Hz)	MSE (dB) (noise variance 0.01 and f_D = 200 Hz)
ε-NLMS	-13.51	-9.39
RLS	-14.25	-9.55
Extended RLS	-14.26	-10.01
Kernel RLS	-20.36	-12.74
Kernel extended RLS	-20.69	-13.85

$$\kappa(u_i, u_j) = \exp(-0.1 \| u_i - u_j \|^2)$$

Computation complexity

Algorithms	Linear LMS	KLMS	KAPA	ex-KRLS
Computation (training)	O(l)	O(i)	$O(i+K^2)$	$O(i^2)$
Memory (training)	O(l)	O(i)	O(i+K)	$O(i^2)$
Computation (test)	O(l)	O(i)	O(i)	O(i)
Memory (test)	O(l)	O(i)	O(i)	O(i)

At time or iteration i

Part 5: Active learning in kernel adaptive filtering

Active data selection

- Why?
  - Kernel trick may seem a “free lunch”!
  - The price we pay is memory and pointwise evaluations of the function.
  - Generalization (Occam’s razor)
- But remember we are working on an on-line scenario, so most of the methods out there need to be modified.

Active data selection

- The goal is to build a constant length (fixed budget) filter in RKHS. There are two complementary methods of achieving this goal:
    - Discard unimportant centers (pruning)
    - Accept only some of the new centers (sparsification)
- Apart from heuristics, in either case a methodology to evaluate the importance of the centers for the overall nonlinear function approximation is needed.
- Another requirement is that this evaluation should be no more expensive computationally than the filter adaptation.

Previous Approaches – Sparsification

- Novelty condition (Platt, 1991)
  - Compute the distance to the current dictionary

$$dis = \min_{c_j \in D(i)} \left\| u(i+1) - c_j \right\|$$

  - If it is less than a threshold δ_1 discard
  - If the prediction error

$$e(i+1) = d(i+1) - \varphi(i+1)^T \Omega(i)$$

  - Is larger than another threshold δ_2 include new center.
- Approximate linear dependency (Engel, 2004)
  - If the new input is a linear combination of the previous centers discard

$$dis_2 = \min \left\| \varphi(u(i+1) - \sum_{c_j \in D(i)} b_j \varphi(c_j) \right\|$$

which is the Schur Complement of Gram matrix and fits KAPA 2 and 4 very well. Problem is computational complexity

Previous Approaches – Pruning

- Sliding Window (Vaerenbergh, 2010)
  - Impose $m_i<B$ in $f_i = \sum_{j=1}^{m_i} a_j(i)\kappa(c_j,.)$
  - Create the Gram matrix of size B+1 recursively from size B

$$\breve{G}(i+1) = \begin{bmatrix} G(i) & h \\ h^T & \kappa(c_{B+1}, c_{B+1}) \end{bmatrix} \qquad h = [\kappa(c_{B+1}, c_1), \ldots, \kappa(c_{B+1}, c_B)]^T$$

$$Q(i) = (\lambda I + G(i))^{-1} \qquad z = Q(i)h \qquad r = \lambda + \kappa(c_{B+1}, c_{B+1}) - z^T h$$

$$\breve{Q}(i+1) = \begin{bmatrix} Q(i) + zz^T/r & -z/r \\ -z^T/r & 1/r \end{bmatrix}$$

  - Downsize: reorder centers and include last (see KAPA2)

$$Q(i+1) = H - ff^T/e \quad a(i+1) = Q(i+1)d(i+1) \qquad f_{i+1} = \sum_{j=1}^{B} a_j(i+1)\kappa(c_j,.)$$

  - See also the Forgetron and the Projectron that provide error bounds for the approximation.

O. Dekel, S. Shalev-Shwartz, and Y. Singer, "The Forgetron: A kernel-based perceptron on a fixed budget," in *Advances in Neural Information Processing Systems 18. Cambridge, MA: MIT Press, 2006, pp. 1342–1372.*

F. Orabona, J. Keshet, and B. Caputo, "Bounded kernel-based online learning," *Journal of Machine Learning Research*, vol. 10, pp. 2643–2666, 2009.

Problem statement

- The learning system $y(u;T(i))$
  - Already processed (your dictionary)

$$D(i) = \{u(j), d(j)\}_{j=1}^{i}$$

- A new data pair $\{u(i+1), d(i+1)\}$
  - How much new information it contains?
  - Is this the right question?

Or

How much information it contains with respect to the learning system $y(u;T(i))$?

Information measure

- Hartley and Shannon’s definition of information
  - How much information it contains?

$$I(i+1) = -\ln p(u(i+1), d(i+1))$$

- Learning is unlike digital communications:

  The machine never knows the joint distribution!
- When the same message is presented to a learning system information (the degree of uncertainty) changes because the system learned with the first presentation!
- Need to bring back MEANING into information theory!

Surprise as an information measure

- Learning is very much like an experiment that we do in the laboratory.
- Fedorov (1972) proposed to measure the importance of an experiment as the Kulback Leibler distance between the prior (the hypothesis we have) and the posterior (the results after measurement).
- Mackay (1992) formulated this concept under a Bayesian approach and it has become one of the key concepts in active learning.

Surprise as an information measure

- Pfaffelhuber in 1972 formulated the concept of subjective or redundant information for learning systems as

$$I_S(x) = -\log(q(x))$$

  the PDF of the data is $p(x)$ and $q(x)$ is the learner's subjective estimation of it.

- Palm in 1981 defined surprise (or conditional information) for a learning system $y(u;T(i))$ as

$$S_{T(i)}(u(i+1)) = CI(i+1) = -\ln p(u(i+1) \mid T(i))$$

Shannon versus Surprise

Shannon (absolute information)	Surprise (conditional information)
Objective	Subjective
Receptor independent	Receptor dependent (on time and agent)
Message is meaningless	Message has meaning for the agent

Evaluation of conditional information (surprise)

- Gaussian process theory

$$CI(i+1) = -\ln[p(\mathbf{u}(i+1), d(i+1) \mid T(i))] =$$

$$\ln\sqrt{2\pi} + \ln\sigma(i+1) + \frac{(d(i+1) - \hat{d}(i+1))^2}{2\sigma^2(i+1)} - \ln[p(\mathbf{u}(i+1) \mid T(i))]$$

- where

$$\hat{d}(i+1) = \mathbf{h}(i+1)^T[\sigma_n^2\mathbf{I} + \mathbf{G}(i)]^{-1}d(i)$$

$$\sigma^2(i+1) = \sigma_n^2 + \kappa(\mathbf{u}(i+1), \mathbf{u}(i+1)) - \mathbf{h}(i+1)^T[\sigma_n^2\mathbf{I} + \mathbf{G}(i)]^{-1}\mathbf{h}(i+1)$$

Interpretation of conditional information (surprise)

$$CI(i+1) = -\ln[p(\mathbf{u}(i+1), d(i+1) \mid T(i))] =$$

$$\ln\sqrt{2\pi} + \ln\sigma(i+1) + \frac{(d(i+1)-\hat{d}(i+1))^2}{2\sigma^2(i+1)} - \ln[p(\mathbf{u}(i+1) \mid T(i))]$$

- Prediction error $e(i+1) = d(i+1) - \hat{d}(i+1)$
  - Large error → large conditional information
- Prediction variance $\sigma^2(i+1)$
  - Small error, large variance → large CI
  - Large error, small variance → large CI (abnormal)
- Input distribution $p(\mathbf{u}(i+1) \mid T(i))$
  - Rare occurrence → large CI

Input distribution

$$p(\mathbf{u}(i+1) \mid T(i))$$

- Memoryless assumption

$$p(\mathbf{u}(i+1) \mid T(i)) = p(\mathbf{u}(i+1))$$

- Memoryless uniform assumption

$$p(u(i+1) \mid T(i)) = const.$$

Unknown desired signal

- Average CI over the posterior distribution of the output

$$\overline{CI}(i+1) = \ln\sigma(i+1) - \ln[p(\mathbf{u}(i+1) \mid T(i))]$$

- Memoryless uniform assumption

$$\overline{CI}(i+1) = \ln\sigma(i+1)$$

- This is equivalent to approximate linear dependency!

Redundant, abnormal and learnable

$$Abnormal: \quad CI(i+1) > T_1$$

$$Learnable: \quad T_1 \geq CI(i+1) \geq T_2$$

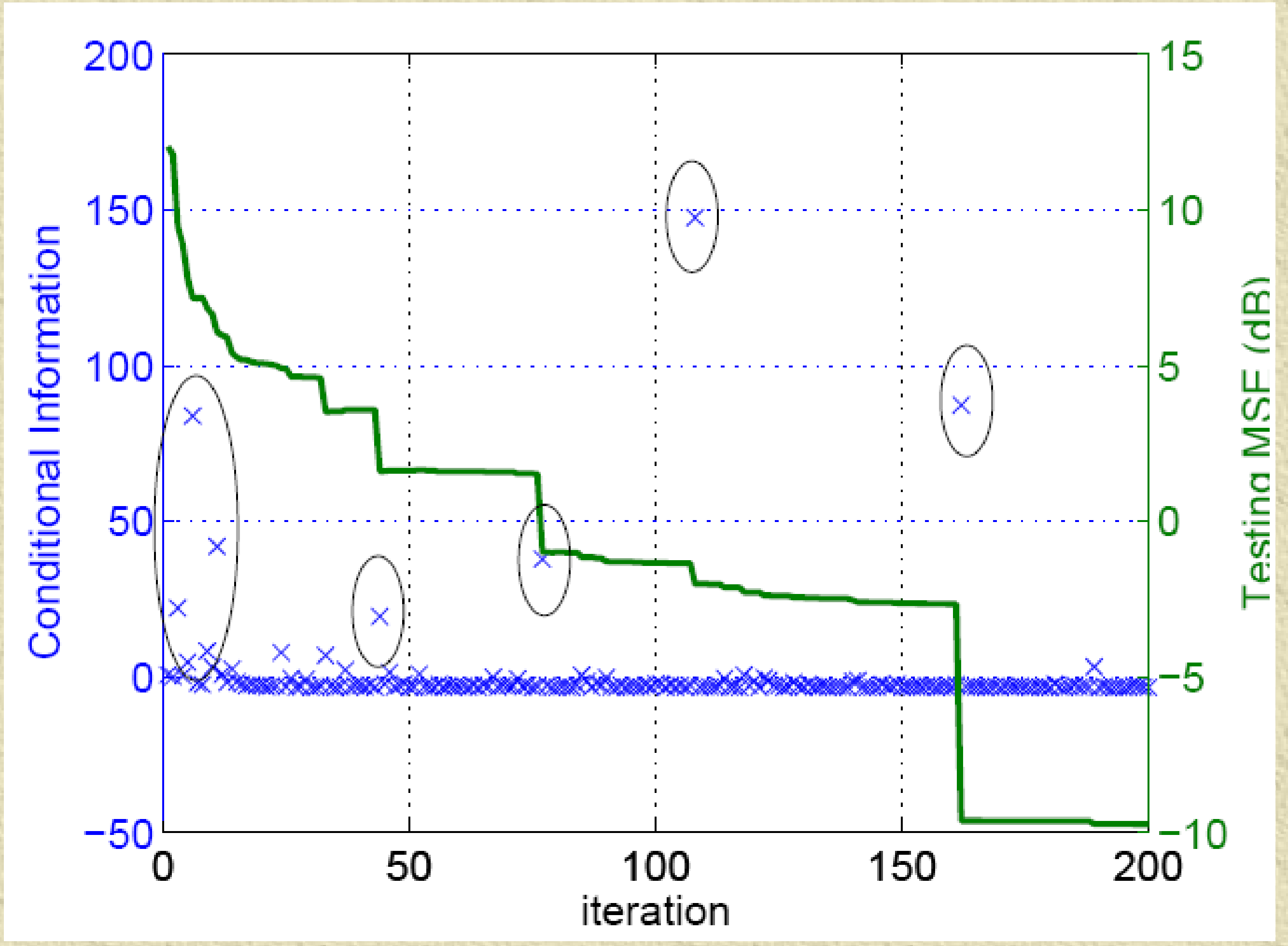
$$\mathrm{Re}\,dundant: \quad CI(i+1) < T_2$$

- Still need to find a systematic way to select these thresholds which are hyperparameters.

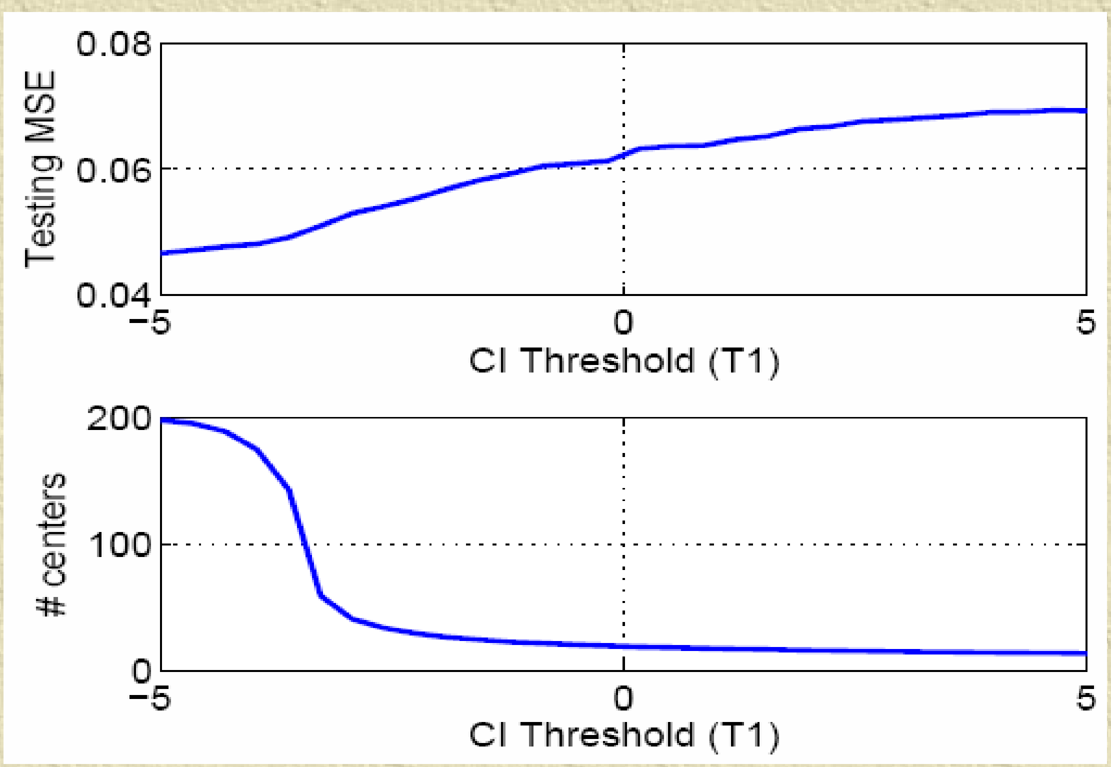
Active online GP regression (AOGR)

- Compute conditional information
- If redundant
  - Throw away
- If abnormal
  - Throw away (outlier examples)
  - Controlled gradient descent (non-stationary)
- If learnable
  - Kernel recursive least squares (stationary)
  - Extended KRLS (non-stationary)

Simulation-5: nonlinear regression—learning curve

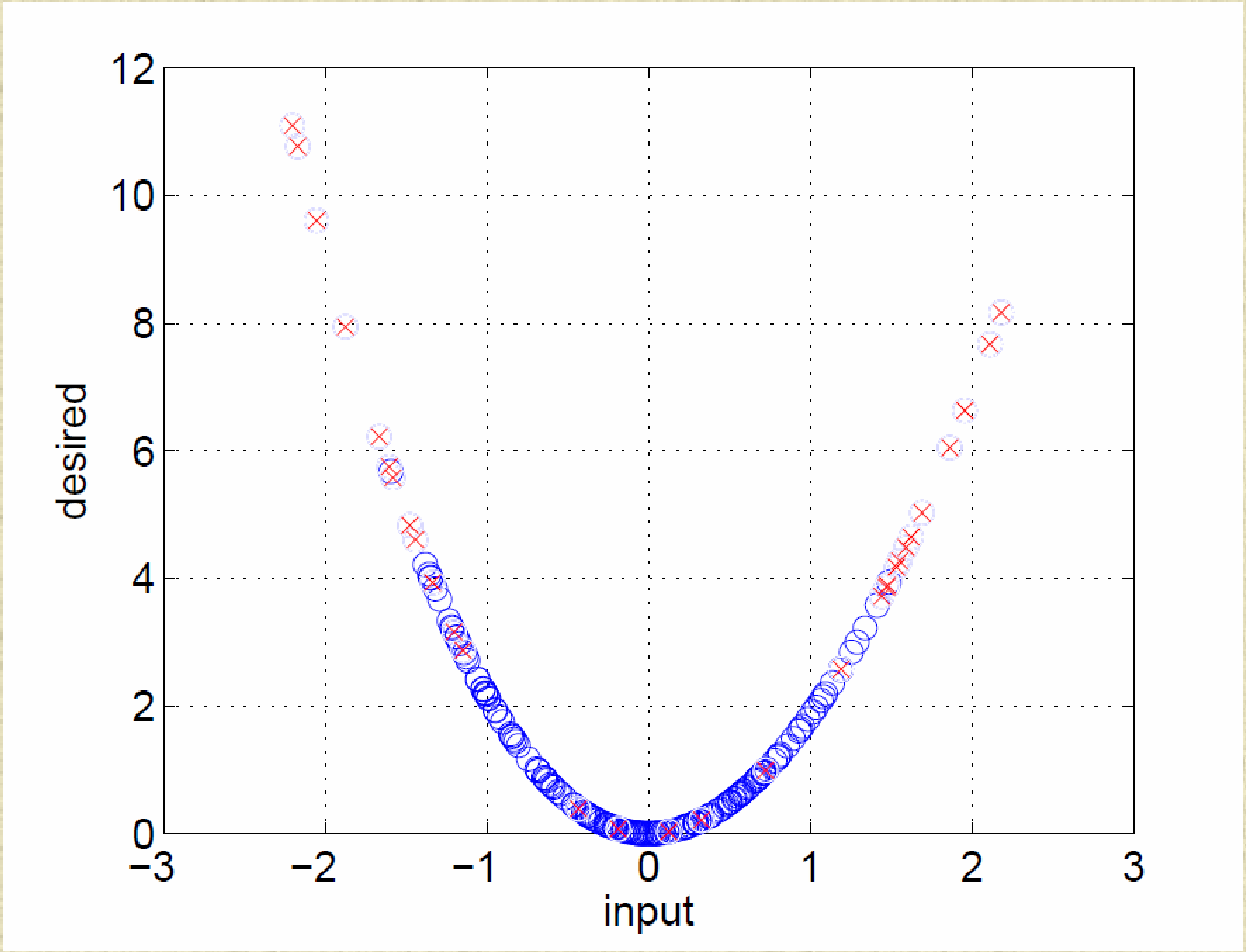


Simulation-5: nonlinear regression—redundancy removal

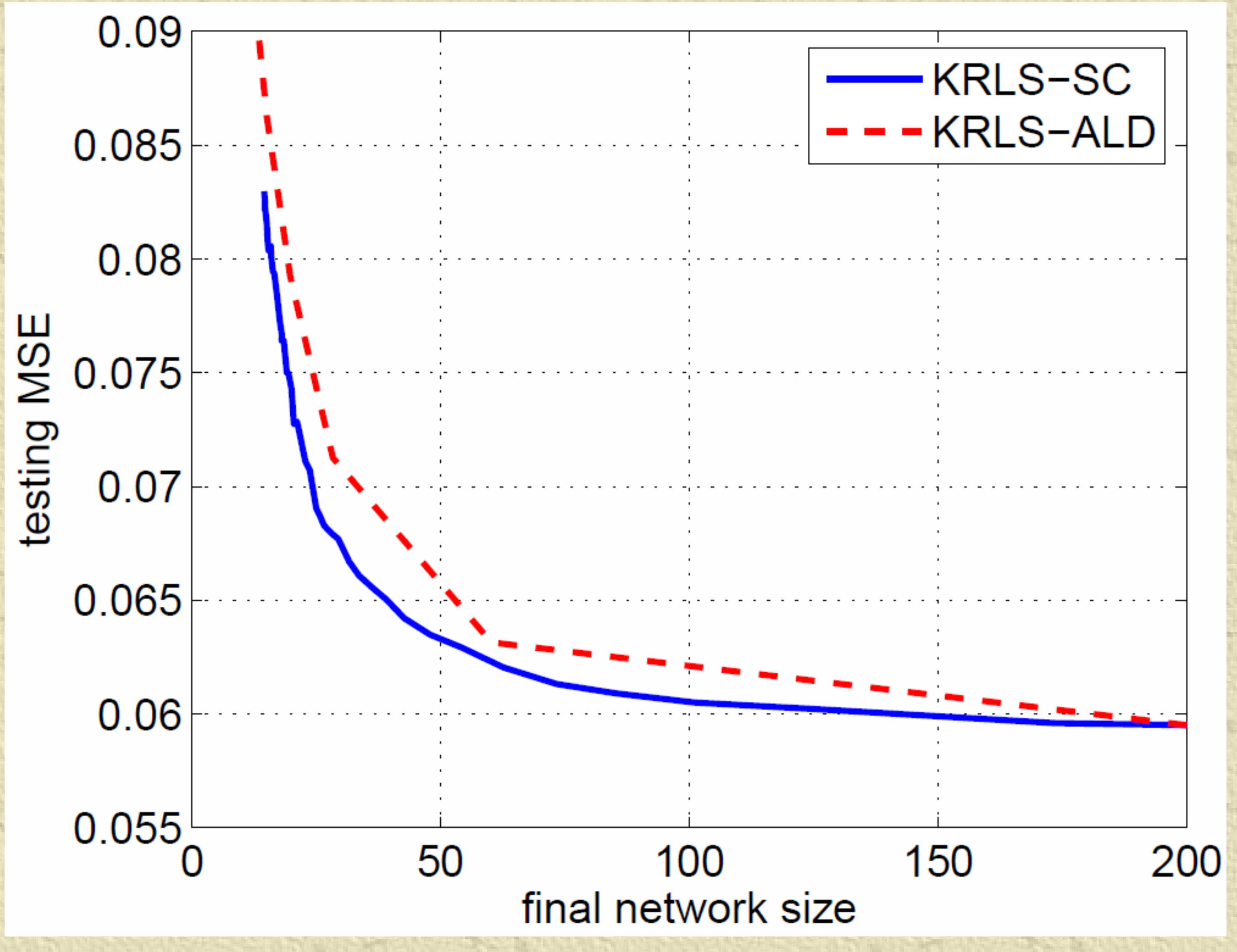


T1 is wrong, should be T2

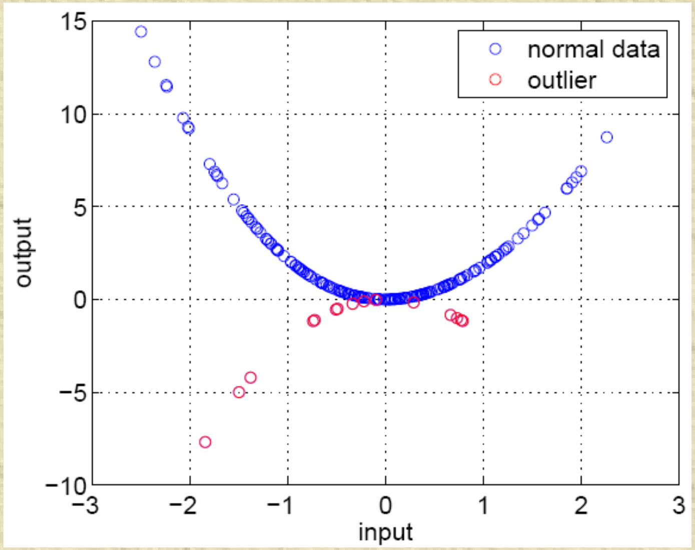
Simulation-5: nonlinear regression– most surprising data

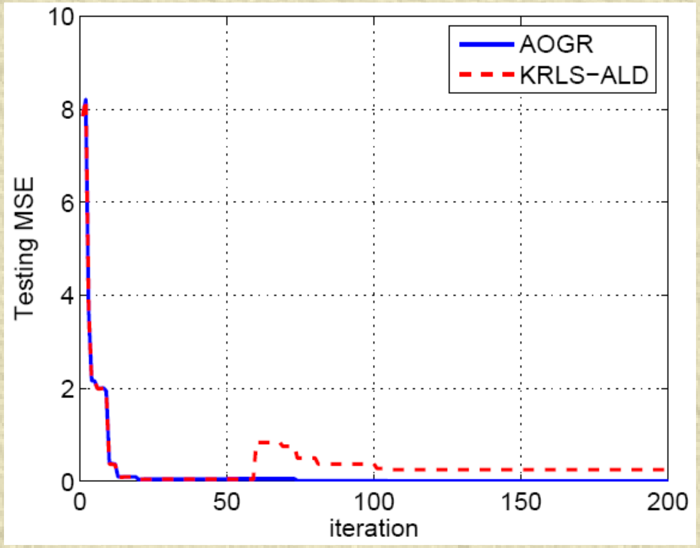


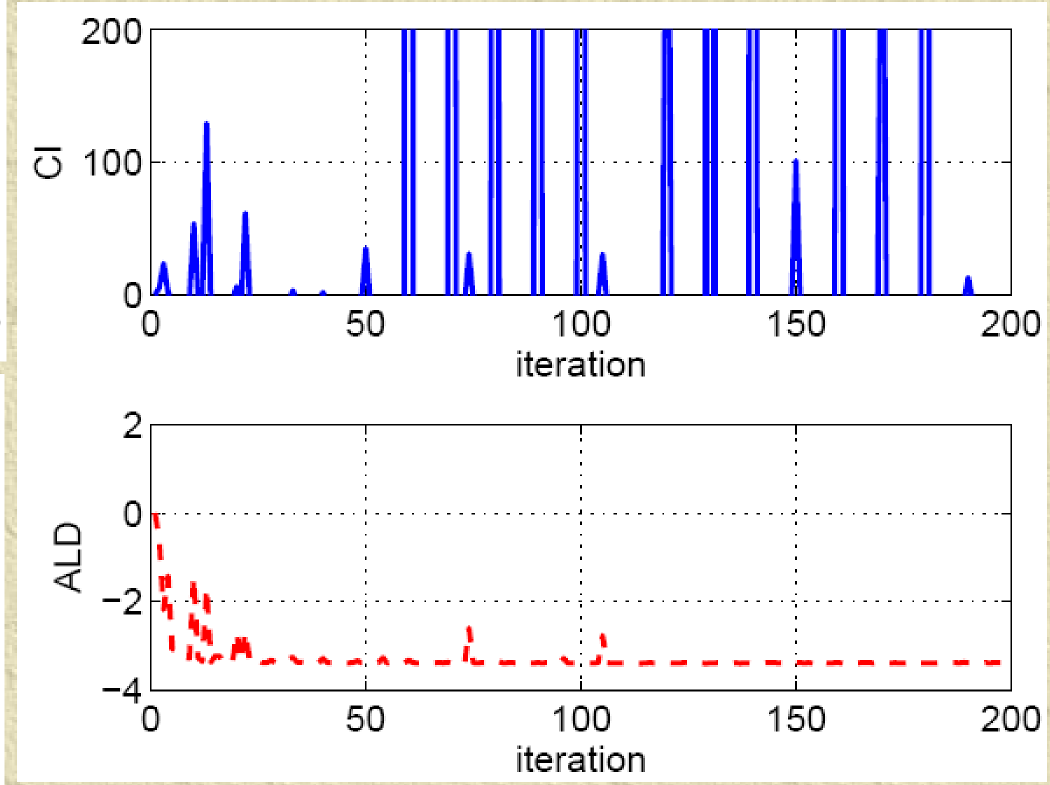
Simulation-5: nonlinear regression



Simulation-5: nonlinear regression—abnormality detection (15 outliers)

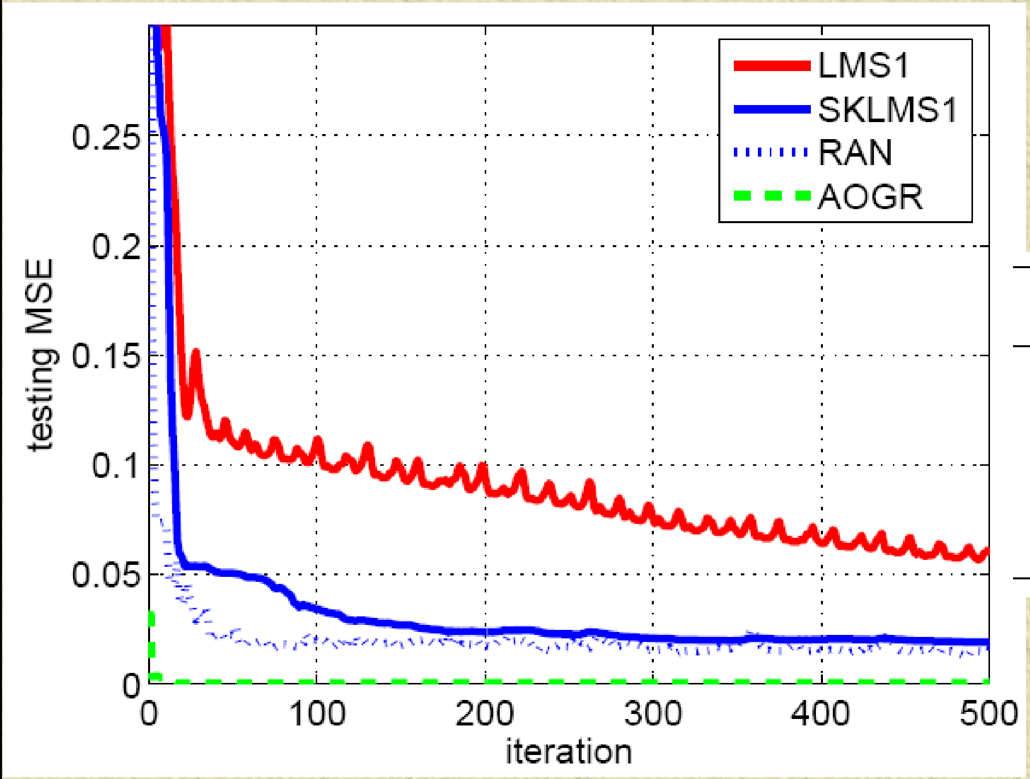


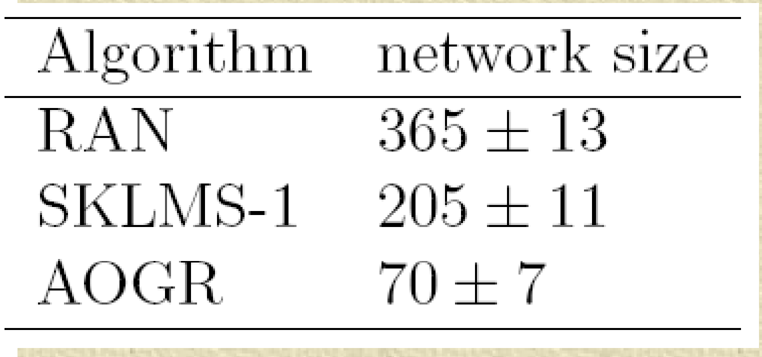




AOGR=KRLS

Simulation-6: Mackey-Glass time series prediction

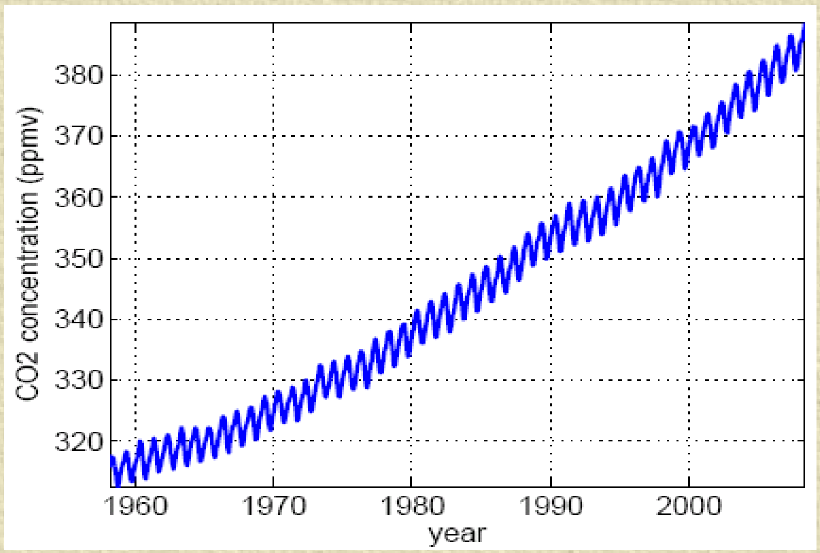


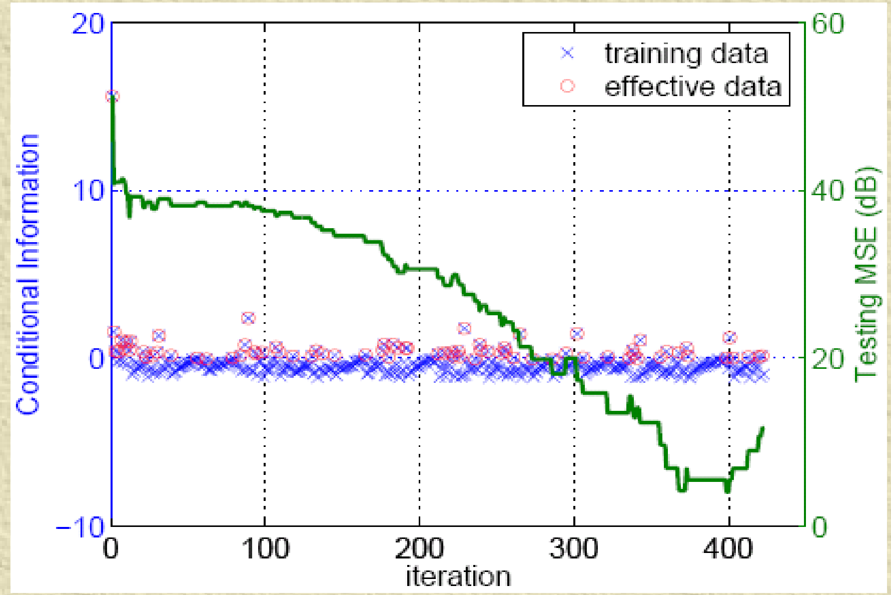


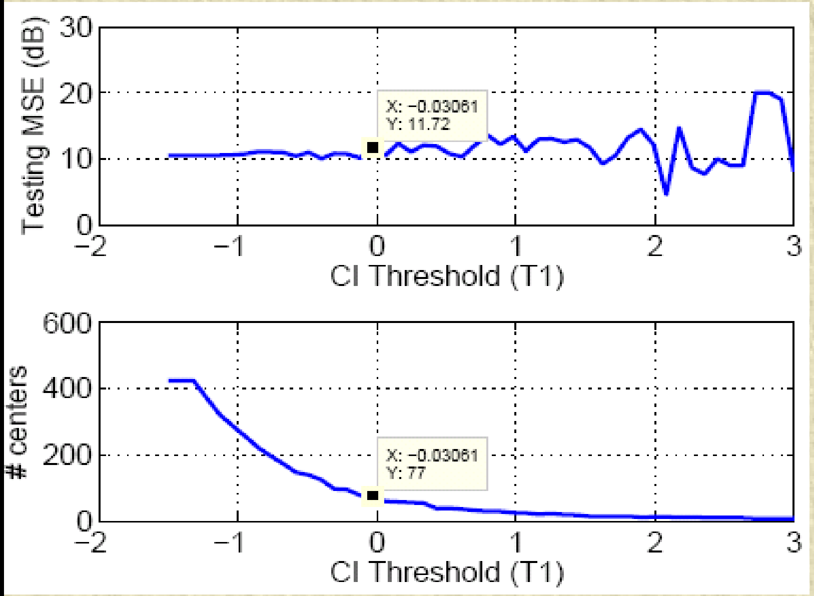
Algorithm	network size
RAN	365 ± 13
SKLMS-1	205 ± 11
AOGR	70 ± 7

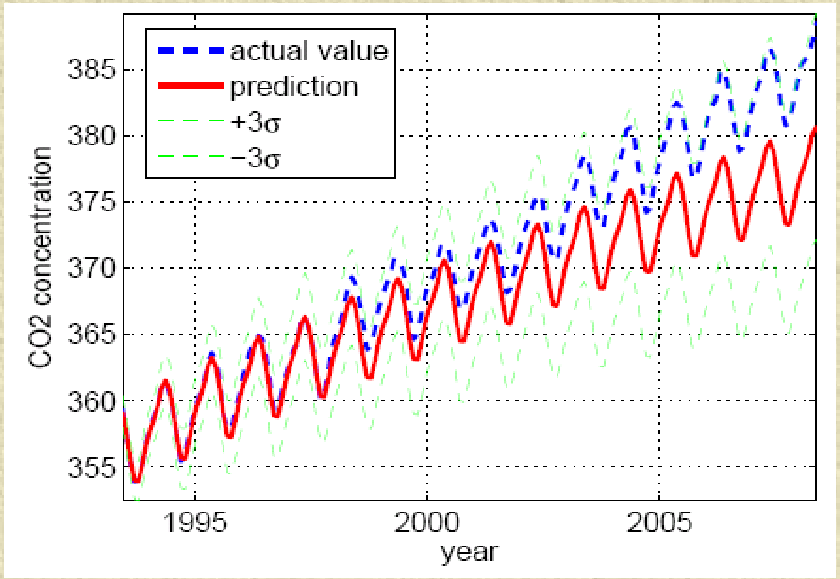
AOGR=KRLS

Simulation-7: CO2 concentration forecasting









Quantized Kernel Least Mean Square

- A common drawback of sparsification methods**:** the *redundant* input data are purely discarded!
- Actually the redundant data are *very useful* and can be, for example, utilized to update the coefficients of the current network, although they are not so important for structure update (adding a new center).
- Quantization approach: the input space is quantized, if the current quantized input has already been assigned a center, we don't need to add a new, but update the coefficient of that center with the new information!
- Intuitively, the coefficient update can enhance the utilization efficiency of that center, and hence may yield better accuracy and a more compact network.

Chen B., Zhao S., Zhu P., Principe J. **Quantized Kernel Least Mean Square Algorithm**, submitted to IEEE Trans. Neural Networks

Quantized Kernel Least Mean Square

- Quantization in Input Space

$$\begin{cases} f_0 = 0 \\ e(i) = d(i) - f_{i-1}(\boldsymbol{u}(i)) \\ f_i = f_{i-1} + \eta e(i)\kappa\left(Q\left[\boldsymbol{u}(i)\right],.\right) \end{cases}$$

- Quantization in RKHS

$$\begin{cases} \boldsymbol{\Omega}(0) = \boldsymbol{0} \\ e(i) = d(i) - \boldsymbol{\Omega}(i-1)^T \boldsymbol{\varphi}(i) \\ \boldsymbol{\Omega}(i) = \boldsymbol{\Omega}(i-1) + \eta e(i)\mathcal{Q}\left[\boldsymbol{\varphi}(i)\right] \end{cases}$$

- Using the quantization method to

compress the input (or feature) space

and hence to compact the RBF structure of the kernel adaptive filter

$\boxed{\text{Quantization operator}}$

Quantized Kernel Least Mean Square

- The key problem is the vector quantization (VQ): Information Theory? Information Bottleneck? ……
- Most of the existing VQ algorithms, however, are not suitable for online implementation because the codebook must be supplied in advance (which is usually trained on an offline data set), and the computational burden is rather heavy.
- A simple online VQ method:

1. Compute the distance between u(i) and C(i-1)
   : $dis\left(\mathbf{u}(i), C(i-1)\right) = \min_{1 \le j \le size(C(i-1))} \left\| \mathbf{u}(i) - C_j(i-1) \right\|$

2. If $dis\left(\mathbf{u}(i), C(i-1)\right) \le \varepsilon_{\mathrm{U}}$ keep the codebook unchanged, and quantize u(i) into the closest code-vector by $\boldsymbol{a}_{j^*}(i) = \boldsymbol{a}_{j^*}(i-1) + \eta e(i)$ $j^* = \arg\min_{1 \le j \le size(C(i-1))} \left\| \mathbf{u}(i) - C_j(i-1) \right\|$

3. Otherwise, update the codebook: $C(i) = \left\{ C(i-1), \mathbf{u}(i) \right\}$, and quantize u(i) as itself

Quantized Kernel Least Mean Square

- Quantized Energy Conservation Relation

$$\left\|\tilde{\mathbf{\Omega}}(i)\right\|_{\mathrm{F}}^{2}+\frac{e_a^2(i)}{\kappa\left(\boldsymbol{u}_q(i),\boldsymbol{u}(i)\right)^2}=\left\|\tilde{\mathbf{\Omega}}(i-1)\right\|_{\mathrm{F}}^{2}+\frac{e_p^2(i)}{\kappa\left(\boldsymbol{u}_q(i),\boldsymbol{u}(i)\right)^2}+\beta_q$$

- A Sufficient Condition for Mean Square Convergence

$$\forall i,\quad \begin{cases} E\left[e_a(i)\tilde{\mathbf{\Omega}}(i-1)^T\boldsymbol{\varphi}_q(i)\right]>0 & (C1) \\ 0<\eta\le\dfrac{2E\left[e_a(i)\tilde{\mathbf{\Omega}}(i-1)^T\boldsymbol{\varphi}_q(i)\right]}{E\left[e_a^2(i)\right]+\sigma_v^2} & (C2) \end{cases}$$

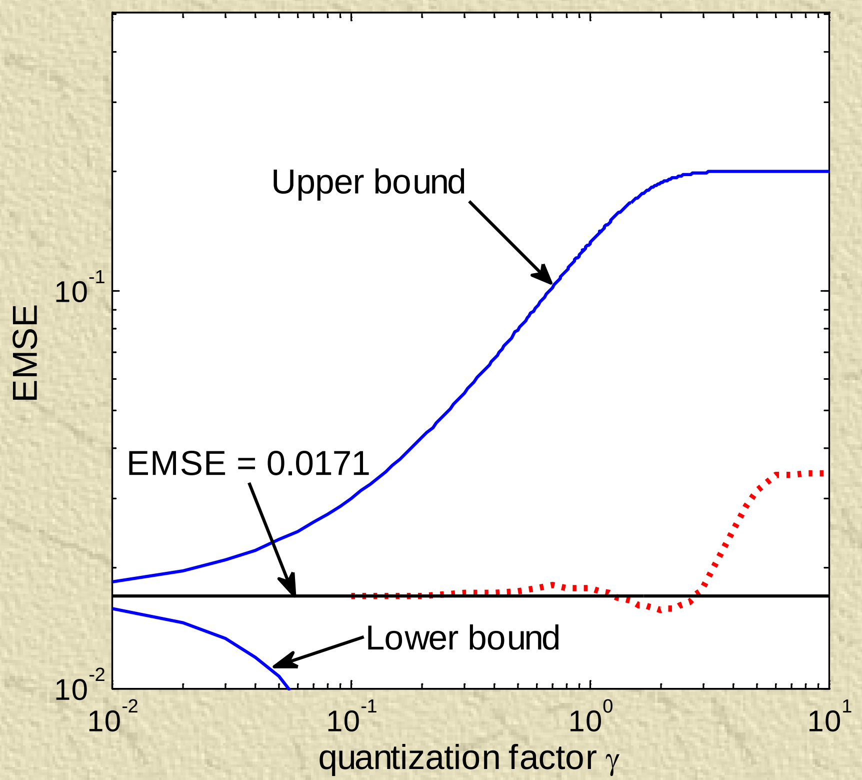
- Steady-state Mean Square Performance

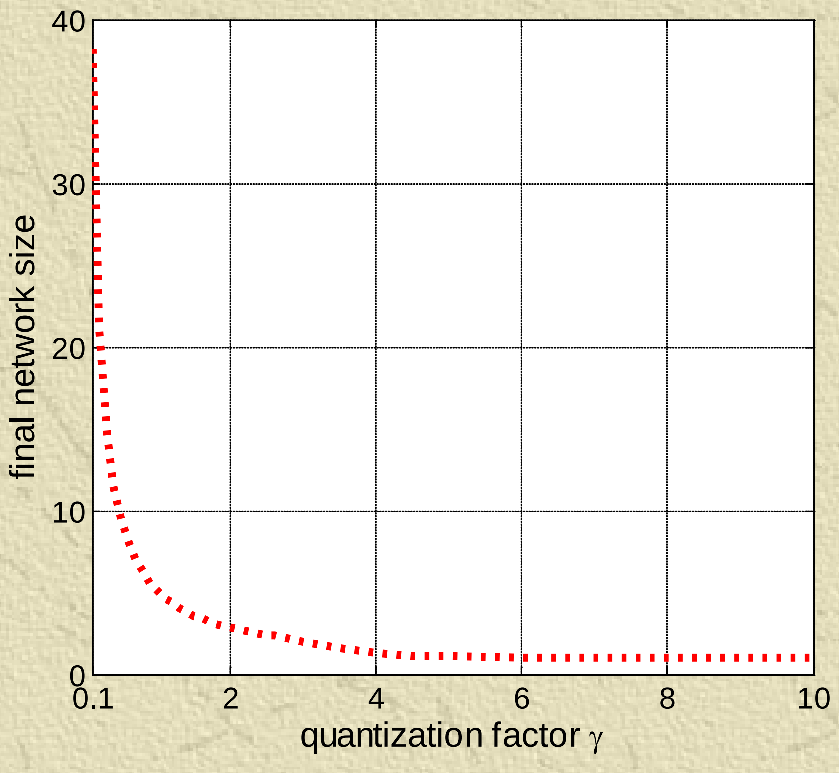
$$\max\left\{\frac{\eta\sigma_v^2-\mathbf{2}\xi_\gamma}{2-\eta},0\right\}\le\lim_{i\to\infty}E\left[e_a^2(i)\right]\le\frac{\eta\sigma_v^2+\mathbf{2}\xi_\gamma}{2-\eta}$$

Quantized Kernel Least Mean Square

- Static Function Estimation

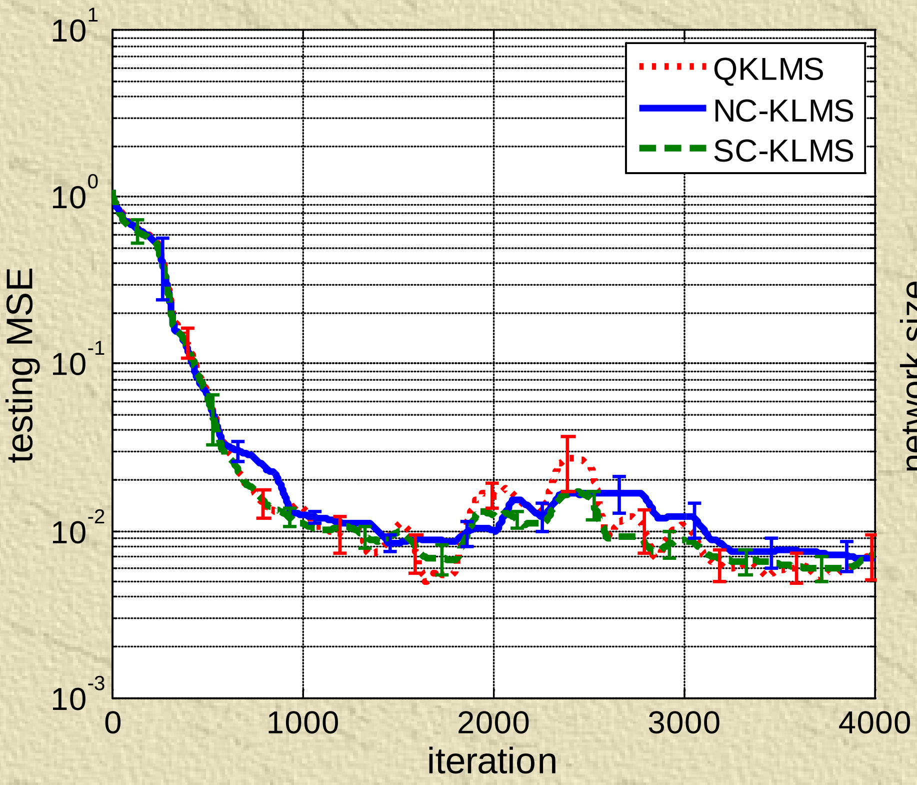
$$d(i) = 0.2 \times \left[\exp\left(-\frac{(u(i)+1)^2}{2} \right) + \exp\left(-\frac{(u(i)-1)^2}{2} \right) \right] + v(i)$$

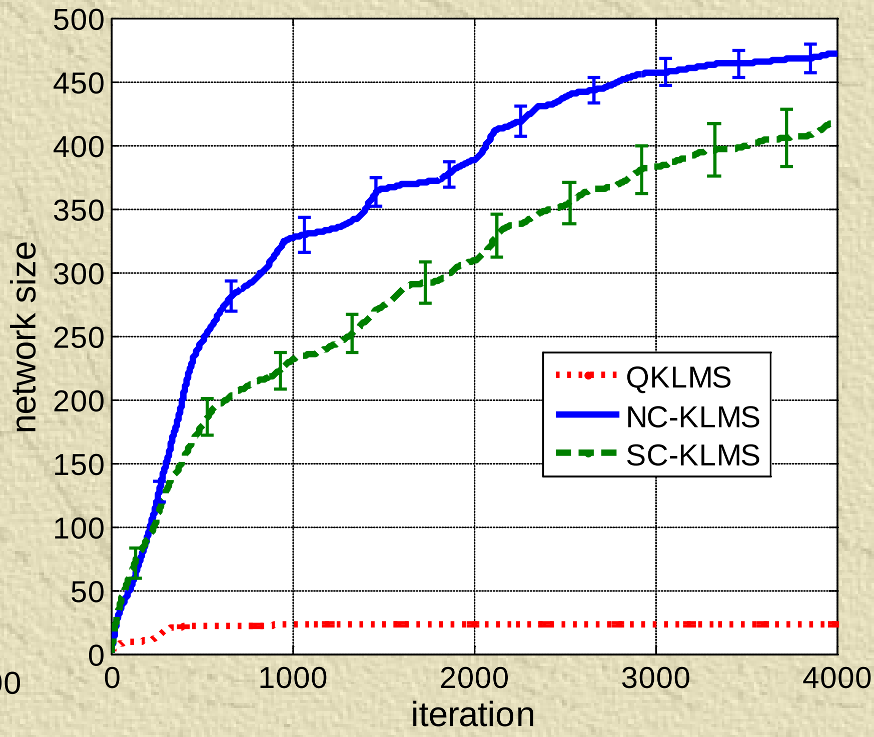




Quantized Kernel Least Mean Square

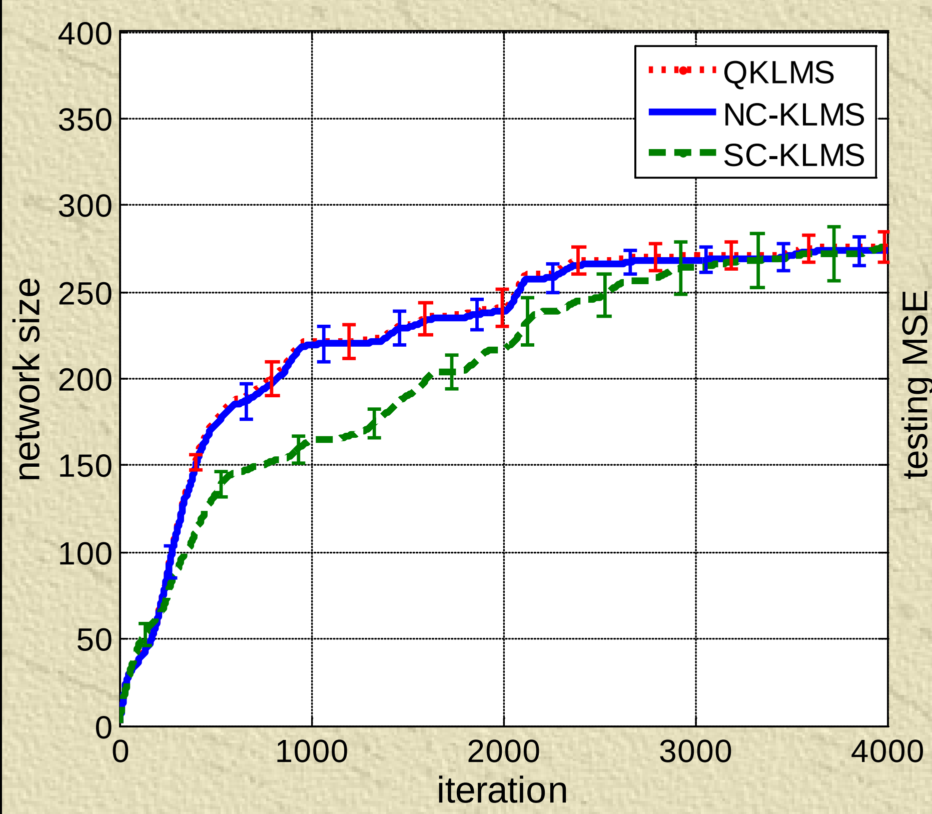
- Short Term Lorenz Time Series Prediction

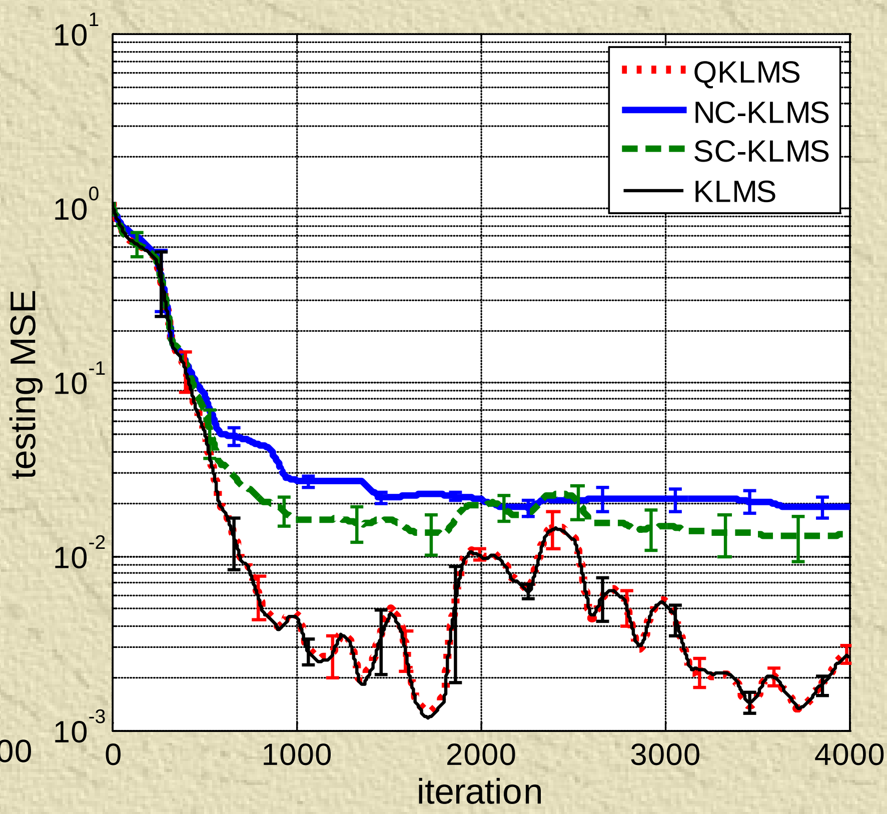




Quantized Kernel Least Mean Square

- Short Term Lorenz Time Series Prediction





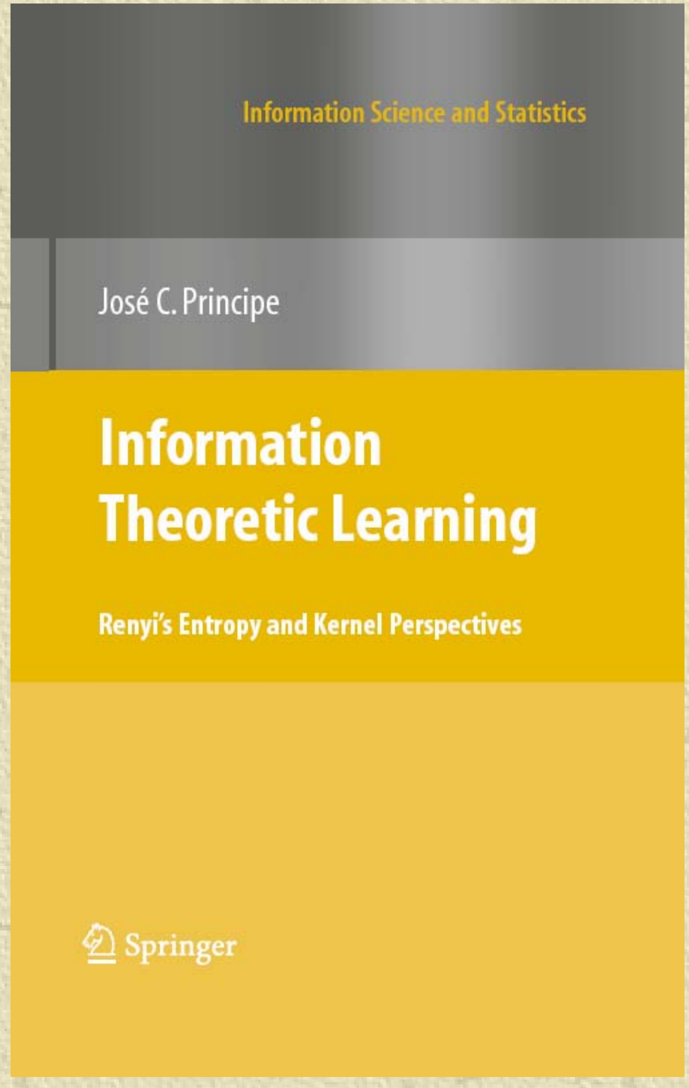
Redefinition of On-line Kernel Learning

- Notice how problem constraints affected the form of the learning algorithms.
- On-line Learning: A process by which the *free parameters* and the *topology* of a ‘learning system’ are *adapted* through a process of stimulation by the environment in which the system is embedded.
- Error-correction learning + memory-based learning
  - What an interesting (biological plausible?) combination.

Impacts on Machine Learning

- KAPA algorithms can be very useful in large scale learning problems.
- Just sample randomly the data from the data base and apply on-line learning algorithms
- There is an extra optimization error associated with these methods, but they can be easily fit to the machine contraints (memory, FLOPS) or the processing time constraints (best solution in x seconds).

Information Theoretic Learning (ITL)


Information Science and Statistics

José C. Principe

Information Theoretic Learning

Renyi's Entropy and Kernel Perspectives

Springer

This class of algorithms can be extended to ITL cost functions and also beyond Regression (classification, Clustering, ICA, etc). See

IEEE
SP MAGAZINE, Nov 2006

Or ITL resource
www.cnel.ufl.edu