The Kernel Least Mean Squares Algorithm

Nikolaos Mitsakos (MathMits@yahoo.gr)

The Kernel Least-Mean-Square Algorithm (W.Liu,P.Pokharel,J.Principle)

Applications of Functional Analysis in Machine Learning - Univ. of Athens 2012
(Chapter 3,N.Mitsakos,P.Bouboulis)

May 11, 2014

What Does the Kernel Trick Do?

Given an algorithm which uses inner products in it's calculations, we can construct an alternative algorithm, by replacing each of the inner products with a positive definite kernel function.

What Does the Kernel Trick Do?

Given an algorithm which uses inner products in it's calculations, we can construct an alternative algorithm, by replacing each of the inner products with a positive definite kernel function.

So ... What is a Positive Definite Kernel Function?

What Is a Kernel Function?

Given a set X, a 2-variable function $K : X \times X \longrightarrow \mathbb{C}$ is called **positive definite (kernel) function** ($K \geq 0$) provided that for each $n \in \mathbb{N}$ and for every choice of n distinct points $\{x_1, ..., x_n\} \subseteq X$ the Gram matrix of K regarding $\{x_1, ..., x_n\}$ is positive definite.

Gram Matrix:

The elements of the **Gram Matrix** (or **kernel Matrix**) of K regarding $\{x_1, ..., x_n\}$ are given by the relation:

$$(K(x_i, x_j))_{i,j} = K(x_i, x_j) \;\text{ for }\; i, j = 1, ..., n \tag{1}$$

The Gram Matrix is a **Hermitian Matrix** i.e. a matrix equal to it's Conjugate Transpose.

Such a matrix being **Positive Definite** means that $\lambda \geq 0$ for each and every one of it's eigenvalues λ.

How Does the Kernel Trick Do It? (in short ...)

Consider a set X and a positive definite (kernel) function $K : X \times X \longrightarrow \mathbb{R}$. The RKHS theory ensures:

- the existence of a corresponding (Reproducing Kernel) Hilbert Space $\mathscr{H}$, which is a vector subspace of $\mathscr{F}(X, \mathbb{R})$ (Moore's Theorem).
- the existence of a representation $\Phi : X \longrightarrow \mathscr{H} : \Phi(x) = k_x$ (**feature representation**) which maps each element of X to an element of $\mathscr{H}$ ($k_x \in \mathscr{H}$ is called the **reproducing kernel function for the point** x).

so that :

$$\langle \Phi(x), \Phi(y) \rangle_{\mathscr{H}} = \langle k_x, k_y \rangle_{\mathscr{H}} = k_y(x) = K(x, y)$$

Thus:

- Through the feature map, the kernel trick succeeds in transforming a **non-linear problem** within the set X into a **linear problem** inside the "better" space $\mathscr{H}$.
- We may, then, solve the linear problem in $\mathscr{H}$, which usually is a relatively easy task, while by returning the result in space X we obtain the final, non-linear, solution to our original problem.

Examples of Kernel functions.

- The most widely used kernel functions include the **Gaussian kernel**:

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-a\|\mathbf{x}_i - \mathbf{x}_j\|^2}$$

as well as the **polynomial kernel**:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + 1)^p$$

But there are plenty of other choices (e.g. linear kernel, exponential kernel, Laplacian kernel etc.)

Examples of Algorithms capable of operating with kernels:

- Support Vector Machines (SVM's)
- Gaussian processes
- Fisher's linear discriminant analysis (LDA)
- Principal Components Analysis (PCA)
- Adaptive filters (Least Mean Squares Algorithm) e.t.c

In This Presentation We Focus On:

- Description of Learning Problems.
- The Least Mean Squares (LMS) algorithm used to address Learning Problems.
- Application of the Kernel Trick - The Kernel LMS algorithm.
- Techniques for Sparsifying the Solution:
  - Platt's Novelty Criterion.
  - Coherence Based Sparsification strategy.
  - Surprise Criterion.
  - Quantization Technique.

The corresponding algorithms for each case are presented in the text: Applications of Functional Analysis in Machine Learning.

MOTIVATION:

Suppose we wish to discover the mechanism of a function

$$F : X \subset \mathbb{R}^M \longrightarrow \mathbb{R} \text{ (\textbf{true filter})}$$

having at our disposal just a sequence of example inputs-outputs

$$\{(\mathbf{x}_1, d_1), (\mathbf{x}_2, d_2), \ldots, (\mathbf{x}_n, d_n), \ldots\}$$

(where $\mathbf{x}_n \in X \subset \mathbb{R}^M$ and $d_n \in \mathbb{R}$ for every $n \in \mathbb{N}$).

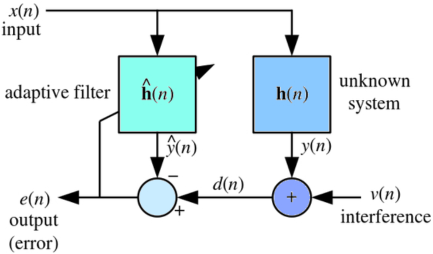

Figure: *Adaptive Filter*

MOTIVATION:

Suppose we wish to discover the mechanism of a function

$$F : X \subset \mathbb{R}^M \longrightarrow \mathbb{R} \text{ (\textbf{true filter})}$$

having at our disposal just a sequence of example inputs-outputs

$$\{(\mathbf{x}_1, d_1), (\mathbf{x}_2, d_2), \ldots, (\mathbf{x}_n, d_n), \ldots\}$$

(where $\mathbf{x}_n \in X \subset \mathbb{R}^M$ and $d_n \in \mathbb{R}$ for every $n \in \mathbb{N}$).

Objective of a typical **Adaptive Learning algorithm:** to determine, based on the given "training" data, the proper input-output relation, $f_{\mathbf{w}}$, member of a parametric class of functions $H = \{f_{\mathbf{w}} : X \longrightarrow \mathbb{R},\ \mathbf{w} \in \mathbb{R}^{\nu}\}$, so as to minimize the value of a predefined loss function $L(\mathbf{w})$.
$L(\mathbf{w})$ calculates the error between the actual result d_n and the estimation $f_{\mathbf{w}}(\mathbf{x}_n)$, at every step n.

Settings for the LMS algorithm:

- **hypothesis space**: the class of linear functions

$$H_1 = \{f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}, \mathbf{w} \in \mathbb{R}^M\}$$

- **loss function**: the Mean Sqare Error (MSE), defined as

$$L(\mathbf{w}) \equiv E[|d_n - f_{\mathbf{w}}(\mathbf{x})|^2] = E[|d_n - \mathbf{w}^T\mathbf{x}|^2]$$

Notation: $e_n = d_n - \mathbf{w}_{n-1}^T\mathbf{x}_n$
We call this the **a priori error** (at each step n).

The TARGET: based on a given set of **training data** $\{(\mathbf{x}_1, d_1), (\mathbf{x}_2, d_2), \dots, (\mathbf{x}_n, d_n), \dots\}$, determine the proper input-output relation $f_{\mathbf{w}}$, so as to minimize the value of the loss function $L(\mathbf{w})$.

Stochastic Gradient Descent method:
at each instance time $n = 1, 2, \ldots, N$ the gradient of the mean square error

$$-\nabla L(w) = 2E[(d_n - \mathbf{w}_{n-1}^T \mathbf{x}_n)(\mathbf{x}_n)] = 2E[e_n \mathbf{x}_n]$$

approximated by it's value at every time instance n

$$E[e_n \mathbf{x}_n] \approx e_n \mathbf{x}_n$$

leads to the **step update (or weight-update) equation**, which, towards the direction of reduction, takes the form:

$$\mathbf{w}_n = \mathbf{w}_{n-1} + \mu e_n \mathbf{x}_n$$

Note: parameter μ expresses the size of the "learning step" towards the direction of the descent.

In more detail, the algorithm's steps evolve in the following manner:

Initialization: $\mathbf{w}_0 = \mathbf{0}$

Step 1: $(\mathbf{x}_1, d_1)$ arrives

Step 2: $f(\mathbf{x}_1) \equiv \mathbf{w}_0^T \mathbf{x}_1 = 0$

Step 3: $e_1 = d_1 - f(\mathbf{x}_1) = d_1$

Step 4: $\mathbf{w}_1 = \mathbf{w}_0 + \mu e_1 \mathbf{x}_1 = \mu e_1 \mathbf{x}_1$

Step 5: $(\mathbf{x}_2, d_2)$ arrives

Step 6: $f(\mathbf{x}_2) \equiv \mathbf{w}_1^T \mathbf{x}_2$

Step 7: $e_2 = d_2 - f(\mathbf{x}_2)$

Step 8: $\mathbf{w}_2 = \mathbf{w}_1 + \mu e_2 \mathbf{x}_2$

Step 9: $(\mathbf{x}_3, d_3)$ arrives

$\vdots$

The Least-Mean Square Code:

- $\mathbf{w} = \mathbf{0}$
- for $i = 1$ to N (e.g. $N = 5000$)

  $f \equiv \mathbf{w}^T \mathbf{x}_i$

  $e = d_i - f$ (a priori error)

  $\mathbf{w} = \mathbf{w} + \mu e \mathbf{x}_i$
- end for

Variation: generated by replacing the last equation of the aforementioned iterative process with

$$\mathbf{w} = \mathbf{w} + \frac{\mu e}{\|\mathbf{x}_i\|^2}\mathbf{x}_i$$

called **Normalized LMS**. It's optimal learning rate has been proved to be obtained when $\mu = 1$.

Note that:

- After a training of n steps has taken place, each **weight $\mathbf{w}_n$** is expressed as the linear combination of all the previous and the last input data, all of them weighted by their corresponding **a priori errors**.
- The input-output procedure of this particular training system can be expressed exclusively in terms of inner products:

$$f(\mathbf{x}_{n+1}) = \mathbf{w}_n^T \mathbf{x}_{n+1} = \mu \sum_{k=1}^{n} e_k \mathbf{x}_k^T \mathbf{x}_{n+1}$$

where

$$e_n = d_n - \mu \sum_{k=1}^{n-1} e_k \mathbf{x}_k^T \mathbf{x}_n \qquad \textit{(text pg. 34)}$$

Conclusion: LMS can be easily extended to kernel LMS algorithm.

Settings for the Kernel LMS algorithm:

- new **hypothesis space**: the space of linear functionals $H_2 = \{ T_{\mathbf{w}} : \mathscr{H} \longrightarrow \mathbb{R}, T_{\mathbf{w}}(\phi(\mathbf{x})) = \langle \mathbf{w}, \phi(\mathbf{x}) \rangle_{\mathscr{H}}, \mathbf{w} \in \mathscr{H} \}$
- new **sequence of examples**: $\{(\phi(\mathbf{x}_1), d_1), \ldots, (\phi(\mathbf{x}_n), d_n)\}$
- determine a function

$$f(\mathbf{x}_n) \equiv T_{\mathbf{w}}(\phi(\mathbf{x}_n)) = < \mathbf{w}, \phi(\mathbf{x}_n) >_{\mathscr{H}} \quad , \mathbf{w} \in \mathscr{H}$$

  so as to minimize the **loss function**:

$$L(w) \equiv E[|d_n - f(\mathbf{x}_n)|^2] = E[|d_n - \langle \mathbf{w}, \phi(\mathbf{x}_n) \rangle_{\mathscr{H}}|^2]$$

- once more:

$$e_n = d_n - f(\mathbf{x}_n)$$

We calculate the Frechet derivative:

$$\nabla L(\mathbf{w}) = -2E[e_n \phi(\mathbf{x}_n)]$$

which again (according to LMS rational...) we approximate by it's value for each time instance n

$$\nabla L(\mathbf{w}) = -2e_n \phi(\mathbf{x}_n)$$

eventually getting, towards the direction of minimization

$$\mathbf{w}_n = \mathbf{w}_{n-1} + \mu e_n \phi(\mathbf{x}_n) \tag{2}$$

Algorithm Steps:

$$\mathbf{w}_0 = 0$$

$$\mathbf{w}_1 = \mu e_1 \phi(\mathbf{x}_1)$$

$$\mathbf{w}_2 = \mu e_1 \phi(\mathbf{x}_1) + \mu e_2 \phi(\mathbf{x}_2)$$

$$\vdots$$

$$\mathbf{w}_n = \mu \sum_{k=1}^{n} e_k \phi(\mathbf{x}_k)$$

So, at each time instance n we get:

$$\begin{aligned} f(\mathbf{x}_n) = T_{\mathbf{w}_{n-1}}(\phi(\mathbf{x}_n)) &= < \mathbf{w}_{n-1}, \phi(\mathbf{x}_n) >_{\mathscr{H}} \\ &= < \mu \sum_{k=1}^{n-1} e_k \phi(\mathbf{x}_k), \phi(\mathbf{x}_n) >_{\mathscr{H}} \\ &= \mu \sum_{k=1}^{n-1} e_k < \phi(\mathbf{x}_k), \phi(\mathbf{x}_n) >_{\mathscr{H}} \\ &= \mu \sum_{k=1}^{n-1} e_k K(\mathbf{x}_k, \mathbf{x}_n) \end{aligned}$$

The Kernel Least-Mean Square Code:

- **Inputs**: the data $(\mathbf{x}_n, y_n)$ and their number N
- **Output**: the expansion $\mathbf{w} = \sum_{k=1}^{N} \alpha_k K(\cdot, \mathbf{u}_k)$, where $\alpha_k = \mu e_k$
- **Initialization**:
  $f^0 = 0$, n: the learning step, μ: the parameter μ of the learning step
  Define: vector $\alpha = 0$, array $D = \{\}$ and the parameters of the *kernel* function.
- **for** $n = 1 \dots N$ **do**

  if $n == 1$ **then**

  $f_n = 0$

  else

  Calculate the filter output $f_n = \sum_{k=1}^{M} \alpha_k K(\mathbf{u}_k, \mathbf{x}_n)$

  end if

  Calculate the error: $e_n = d_n - f_n$

  $\alpha_n = \mu e_n$

  Register the new center $\mathbf{u}_n = \mathbf{x}_n$ at the center's list, i.e.
  $D = \{D, \mathbf{u}_n\}$, $\alpha^T = \{\alpha^T, \alpha_n\}$
- **end for**

Notes on Kernel LMS algorithm:

- After N steps of the algorithm, the input-output relation is

$$\mathbf{w}_n = \mu \sum_{k=1}^{n} e_k \phi(\mathbf{x}_k)$$

$$f(\mathbf{x}_n) = \mu \sum_{k=1}^{n-1} e_k K(\mathbf{x}_k, \mathbf{x}_n) \qquad \textit{(see text pg. 37)}$$

- We can, again, use a normalised version:

$$\mathbf{w}_n = \mathbf{w}_{n-1} + \frac{\mu e_n}{K(\mathbf{x}_n, \mathbf{x}_n)} \phi(\mathbf{x}_n)$$

getting the **normalized KLMS (NKLMS)**.(replacing the step $a_n = \mu e_n$ with $a_n = \frac{\mu e_n}{\kappa}$, where $\kappa = K(\mathbf{x}_n, \mathbf{x}_n)$ would have already been calculated at some earlier step).

Disadvantage of the KLMS algorithm: The number of points $\mathbf{x}_n$ that get involved to the estimation of the result (**Dictionary**) increases continually. This leads to:

- constant increase in memory demands
- constant increase in computational power demand

for as long as the algorithm evolves.

Solution: Discover methods that will limit the expansion's size, by

- forming the dictionary, to some extend, during the first stages of the algorithm, adding plenty of new points (**centers**) and increasing it's range
- subsequently allow new points to be added as centres only when they satisfy certain criteria.

Generally, sparsification can be achieved by importing in Kernel LMS algorithm the following procedure:

$\vdots$

Calculate the error: $e_n = d_n - f_n$

$\alpha_n = \mu e_n$

Sparsification Rules Check

if Sparsification Rules are Satisfied **then**

$M = M + 1$

Register the new centre $\mathbf{u}_M = \mathbf{x}_n$ at the centres list

$D = \{D, \mathbf{u}_M\}$, $\alpha^T = \{\alpha^T, \alpha_n\}$

end if

$\vdots$

Go to text (pg.40): See Platt's Novelty Criterion & Quantization.

Platt's novelty criterion: for every pair $(\mathbf{x}_n, d_n)$ that arrives:

- Initially, the distance of the new point $\mathbf{x}_n$ from the dictionary D_{n-1} is calculated
$$dist = \min_{\mathbf{u}_k \in D_{n-1}} \{\|\mathbf{x}_n - \mathbf{u}_k\|\}$$
- If $dist < \delta_1$ (a predefined lower limit), i.e. the vector under consideration is "very" close to one of the vectors already in the dictionary: the new vector is not registered at the dictionary (so $D_n = D_{n-1}$).
- Else the error $e_n = d_n - f_n$ is calculated. If $|e_n| \leq \delta_2$ (predefined limit): the new point, still, doesn't get registered at the dictionary, (once more $D_n = D_{n-1}$).
- Only if $|e_n| \geq \delta_2$ then: $\mathbf{x}_n$ is registered at D_{n-1} so the dictionary is then shaped as $D_n = D_{n-1} \cup \{\mathbf{x}_n\}$.

Of course, every time we register a new point at the dictionary D we should not neglect to register the corresponding coefficient $a_n = \mu e_n$ at the coefficients list α.

Main disadvantage: such methods they preserve, indefinitely and unchanged, the old information (in the form of a_i that constitute α), thus not being able to cope with changes that may effect the channel. (should be considered more as on-line than as adaptive filtering algorithms).

An alternative approach: imposing sparsity on the solution of KLMS, preserving also the ability to adjust to channel's changes (**quantization** of the training data inside the input space).

The Quantized Kernel Least-Mean Square (QKLMS):
Each new data $\mathbf{x}_n$ arrives successively and the algorithm decides whether this is a new center or a redundant point. Specifically:

- If the distance of $\mathbf{x}_n$ from the dictionary D_n, as shaped until that certain time instance, is greater or equal than the quantizing size δ (i.e. $\mathbf{x}_n$ cannot be "quantized" to one of the points already in D_{n-1}): $\mathbf{x}_n$ is classified as a new center and it gets registered at the dictionary $(D_n = \{D_{n-1}, \mathbf{x}_n\})$.
- Otherwise, $\mathbf{x}_n$ is recognized as "redundant" point and the algorithm does not unnecessarily burden the size of the dictionary by registering it as a new center. However it takes advantage of that information by updating the coefficient of the center which is closest to this particular point (say $\mathbf{u}_l \in D_n$)(i.e. $a_l = a_l + \mu e_n$) .

In order to test the performance of KLMS algorithm we consider a typical **non-linear channel equalization** task. The non-linear channel consists of a linear filter

$$t_n = 0.8 \cdot y_n + 0.7 \cdot y_{n-1}$$

and a memoryless non-linearity

$$q_n = t_n + 0.8 \cdot t_n^2 + 0.7 \cdot t_n^3$$

Then, the signal gets effected by additive white Gaussian noise being finally observed as x_n. Noise level has been set equal to 15 dB.

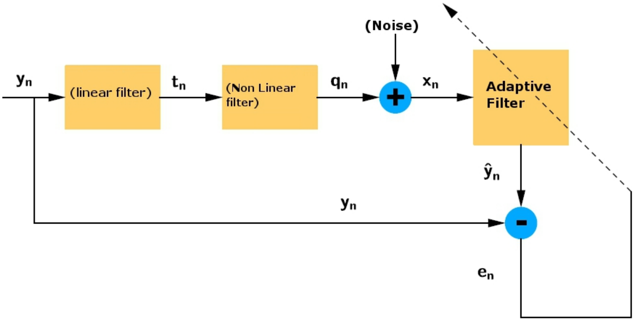

Figure: *Equalization Task*

- The Channel Equalization Task aims at designing an inverse filter which acts upon the filter's output, x_n, thus producing the original input signal as close as possible.
- We execute the algorithm KLMS for the set of examples

$$((x_n, x_{n-1}, \ldots, x_{n-k+1}), y_{n-D})$$

  where $k > 0$ is the "equalizer's length" and D the "equalizer's time delay" (present at almost any equalization set up).
- In other words, the equalizer's result at each time instance n corresponds to the estimation of y_{n-D}.

Details about the test:

- We used **50** sets of **5000** input signal samples each (Gaussian random variable with zero mean and unit variance) comparing the performance of standard **LMS** with that of **KLMS**, applying two different sparsification strategies.
- Regarding the two versions of KLMS:
  - **Gaussian kernel function** was used in both variants.
  - NKLMS(nov.crit.): **Platt's Novelty Criterion** was adopted as the solution's sparsification strategy
  - QNKLMS: the technique of **data quantization** was used.
- We consider all algorithms in their **normalized version**.
- The step update parameter was set for optimum results (in terms of the steady-state error rate). Time delay was also configured for optimum results.

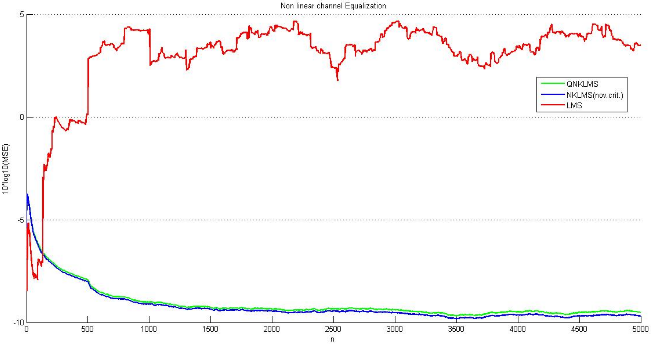

Figure: *The Learning curves for normalized LMS and two different versions of the KLMS algorithms. For KLMS, the Gaussian kernel function ($\sigma = 5$) was used in both cases. For NKLMS(nov.crit.) variant Platt's Novelty Criterion was adopted as the solution's sparsification technique ($\delta_1 = 0.04$, $\delta_2 = 0.04$), while in QNKLMS quantization of the data undertakes the sparsification labor (with quantization size $\delta = 0.8$).*

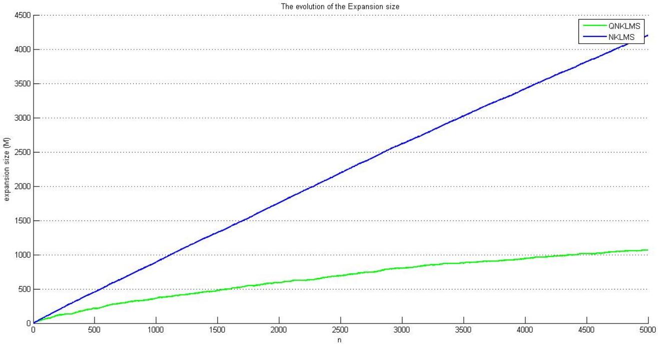

Figure: *The evolution of the expansion of the solution (i.e. the number of terms that appear in the expansion of the solution) applying the two different sparsification methods: Platt's novelty criterion ($\delta_1 = 0.04$, $\delta_2 = 0.04$) in NKLMS(nov.crit.), quantization of the data (with quantization size $\delta = 0.8$) in QNKLMS.*

Conclusions:

- The **superiority of KLMS** is obvious, which was of no surprise as LMS is incapable of handling non-linearities.
- The **economy achieved by quantization**, without any actual cost in the efficiency of the algorithm, is remarkable.

- **Chaotic Time Series Prediction:** short-term prediction of the terms of the chaotic Mackey-Glass time-series. (see paper)
- **Real Data Prediction (in progress):**
  - Apollo 14 Active Seismic Experiment: prediction of the actual, chaotic, data recorded during the “thumber Apollo 14 Active Seismic Experiment (ASE).
  - Economy Indices: prediction of the National Stock Market Index of Greece.

Thank You !

Enjoy Summer While it Lasts!