INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

The Equi-Correlation Network: a New Kernelized-LARS with Automatic Kernel Parameters Tuning

Loth Manuel and Preux Philippe

N° 6794 — version 1.2

initial version Nov. 2008 — revised version Janvier 2009

Thème COG

ISSN 0249-6399 ISRN INRIA/RR--6794--FR+ENG

The Equi-Correlation Network: a New Kernelized-LARS with Automatic Kernel Parameters Tuning

Loth Manuel and Preux Philippe*

Thème COG — Systèmes cognitifs
Projet SequeL

Rapport de recherche n° 6794 — version 1.2 — initial version Nov. 2008 — revised version Janvier 2009 — 17 pages

Abstract:

Machine learning heavily relies on the ability to learn/approximate real functions. State variables, the perceptions, internal states, *etc*, of an agent are often represented as real numbers; grounded on them, the agent has to predict something, or act in some way. In this view, this outcome is a nonlinear function of the inputs. It is thus a very common task to fit a nonlinear function to observations, namely solving a regression problem. Among other approaches, the LARS is very appealing, for its nice theoretical properties, and actual efficiency to compute the whole l_1 regularization path of a supervised learning problem, along with the sparsity. In this paper, we consider the kernelized version of the LARS. In this setting, kernel functions generally have some parameters that have to be tuned. In this paper, we propose a new algorithm, the Equi-Correlation Network (ECON), which originality is that while computing the regularization path, ECON automatically tunes kernel hyper-parameters; thus, this opens the way to working with infinitely many kernel functions, from which, the most interesting are selected. Interestingly, our algorithm is still computationaly efficient, and provide state-of-the-art results on standard benchmarks, while lessening the hand-tuning burden.

Key-words: supervised learning, non linear function approximation, non parametric function approximation, kernel method, LARS, l_1 regularization

* Both authors are with INRIA Lille - Nord Europe , University of Lille , LIFL, France

Unité de recherche INRIA Futurs
Parc Club Orsay Université, ZAC des Vignes,
4, rue Jacques Monod, 91893 ORSAY Cedex (France)
Téléphone : +33 1 72 92 59 00 — Télécopie : +33 1 60 19 66 08

Le réseau équi-corrélé : un nouvel algorithme LARS noyauté avec un réglage automatique des paramètres des noyaux

Résumé : Un ingrédient important de l'apprentissage automatique est la capacité à apprendre et représenter une fonction réelle. Variables d'états, perceptions, états internes, *etc.*, d'un agent sont souvent représentées par un nombre réel ; avec ces données, l'agent doit prédire quelque chose, ou agir. Ainsi, cette prédiction est une fonction non linéaire des variables d'entrées. Il s'agit donc d'ajuster une fonction non linéaire à des observations, donc d'effectuer une régression. Parmi d'autres approches, l'algorithme LARS possède de nombreuses qualités, depuis ces propriétés formelles à son efficacité pour calculer le chemin de régularisation l_1 complet en apprentissage supervisé (classification, ou régression) et la parcimonie des solutions obtenues. Dans ce rapport, on considère une version noyautée, ou plutôt "*featurizée*", du LARS. Dans ce cadre, les noyaux ont généralement des (hyper-)paramètres qui doivent être réglés. Nous proposons un nouvel algorithme, le réseau équi-corrélé (ECON pour *Equi-Correlated Network*) qui, en calculant le chemin de régularisation, règle au mieux ces hyper-paramètres ; cela ouvre la porte à la possibilité de travailler avec une infinité de noyaux potentiels parmi lesquels seuls les plus adéquats sont sélectionnés. Notons que ECON demeure efficace en terme de temps de calcul et espace mémoire, et fournit des résultats exprimentaux au niveau de l'état de l'art, tout en diminuant le travail de paramétrage "à la main".

Mots-clés : apprentissage supervisé, approximation de fonctions non linéaires, approximation de fonctions non paramétrique, méthode à noyau, LARS, régularisation l_1

1 Introduction

The design of autonomous agents that are able to act on their environment, and to adapt to its changes, is a central issue in artificial intelligence, since its early days. To reach this goal, reinforcement learning provides a very appealing framework, in which an agent learns an optimal behavior by interacting with its environment. A central feature of such reinforcement learners is their ability to learn, and represent a real function, hence perform a regression task. So, even if the regression problem is not the full solution to the reinforcement learning problem, it is indeed a key, and basic, component of such learning agents. More generally, in machine learning, regression and classification are very common tasks to solve. While focusing on regression here, the algorithm we propose may be directly used for supervised classification.

Let us formalize the problem of regression. In this problem, an agent has to represent, or approximate a real function defined on some domain $\mathcal{D}$, given a set of n examples $(x_i, y_i) \in \mathcal{D} \times \mathbb{R}$. It is then supposed that there exists some deterministic function y, and the y_i are noisy realization of y. The agent has to learn an estimator $\hat{y} : \mathcal{D} \rightarrow \mathbb{R}$ so as to minimize the difference between $\hat{y}$ and y. There are innumerable ways to derive such a $\hat{y}$ (see [5] for an excellent survey).

One general approach is to look for an estimator which is a linear combination of K basis functions $\{g_k : \mathcal{D} \rightarrow \mathbb{R}\}_k$ and we search for $\hat{y} \equiv \sum_{k=1}^{k=K} w_k g_k$. This is very general, and encompasses multi-layer perceptrons with linear output, RBF networks, support vector-machines and (most) other kernel methods, ...

The set of basis functions may be either fixed *a priori*, thus does not rely on the examples (parametric approach), or may evolve, and adapt to the examples (non parametric approach), such as in Platt's Resource-Allocating Network [10], in Locally Weighted Projection Regression [15], in GGAP-RBF [6], or Grafting [9]. Either parametric, or not, the form of the estimator $\hat{y}$ is the same. However, in parametric regression, we look for the best w_k given the set of K basis functions g_k, whereas in non parametric regression, we look at the w_k, the g_k, and K altogether. In this case, it is customary to look for the sparsest solutions, that is, $\hat{y}$ in which the number of terms is as little as possible.

To find the best parameters w_k, the l_2 norm is very often used as the objective function to minimize: given a set of examples $(\mathcal{X}, \mathcal{Y}) \equiv \{(x_i, y_i)\}$, we define:

$$l_2 = \sum_{i=1}^{i=n} (\hat{y}(x_i) - y_i)^2 \quad .$$

To obtain a sparse solution, it is usual to use an l_1 regularization defined as:

$$l_1(\sum_{k=1}^{k=K} w_k g_k) \equiv \sum_{k=1}^{k=K} |w_k| \quad .$$

By combining both terms, we obtain the objective function:

$$\zeta \equiv \sum_{i=1}^{i=n} (\hat{y}(x_i) - y_i)^2 + \lambda \sum_{k=1}^{k=K} |w_k| \quad ,$$

with λ a regularization constant.

Using the l_1 norm to obtain a sparse solution is actually a heuristic; ideally, we would use an l_0 norm, if the problem would not be untractable in this case. There are

many empirical evidences that the l_1 sparsifies. This heuristic flavor of l_1 keeps the door open to a non strict obedience to the l_1 measure to reach, even higher, sparsity (see for instance [13]).

Minimizing ζ is known as the LASSO problem [14]. Finding the optimal λ is yet an other problem to be solved. Instead of choosing arbitrarily, and *a priori* the value of λ and solve the LASSO with this value, we may dream of solving this minimization problem for all λ's. Actually this dream is a reality: the LARS algorithm [1] is a remarkably efficient way of computing all possible LASSO solutions. The LARS computes the l_1 regularization path, that is, basically all the solutions of the LASSO problem, for all values of λ ranging from 0 to $+\infty$. Though dealing with infinity, the LARS is a practical algorithm, that runs in a finite amount of time, and is actually very efficient: computing the regularization path turns out to be only a little more expensive that computing the solution of the LASSO for a single value of λ.

Initially proposed with the g_k basis functions being the attributes of the data, the algorithm has been extended to arbitrary finite sets of basis functions, such as kernel functions [3, 4]. As basis functions, Gaussian kernels are very popular, but many other kernels may be used. Indeed, it is known that the solution of the LASSO problem may be written $\hat{y} \equiv \sum_k w_k g_k$ under suitable conditions [8]. Hence, basis functions generally have (hyper-)parameters that have to be tuned, that is, $g(\boldsymbol{x})$ is really a $g(\boldsymbol{\theta}, \boldsymbol{x})$. For the moment, a kernel with two different values of parameters are considered as different (*i.e.*, $g(\boldsymbol{\theta_1},) \neq g(\boldsymbol{\theta_2},)$ when $\boldsymbol{\theta_1} \neq \boldsymbol{\theta_2}$). In this paper, we extend the kernelized-LARS algorithm to handle the problem of tuning the hyper-parameters automatically. So, instead of providing the (infinite) set of basis function to the kernelized-LARS and let it choose the best ones, we merely provide one basis function, which is thus a function (of $\boldsymbol{\theta}$). During the resolution of the LASSO, our algorithm then uses the kernel with the adequate parameters, in each term of $\hat{y}$ it appears in. For some reasons that will be clarified in this paper, we name our algorithm the "Equi-Correlated Network", ECON for short.

In the sequel of this paper, we will first present the idea of l_2, l_1 minimization (the LASSO problem), and the LARS algorithm that computes the regularization path. Based on that, we present the ECON algorithm, accompanied with details on practical implementation issues. Then, we provide some experimental results, before we conclude.

2 Background

This section serves as introducing the necessary background on the LASSO problem, and the LARS algorithm.

2.1 Notations

In the following, we use the following notations:

- vectors are written in bold font, *e.g.*, $\boldsymbol{\theta}$, $\mathbf{x}$,
- the i^{th} scalar component of a vector is written in regular font with a subscript, *e.g.*, θ_i, x_i,
- the i^{th} component of a list of vectors is written in bold font with a subscript, *e.g.*, $\boldsymbol{\theta}_i$, $\mathbf{x}_i$,

- matrices are written in capital bold font such as $\mathbf{X}$, $\mathbf{X}^T$ denotes its transpose, $\mathbf{x^i}$ its i^{th} line, and $\mathbf{x_i}$ its i^{th} column.

2.2 l_2-l_1 path: the LARS algorithm

Algorithm 1: LARS algorithm

Input: vector $\mathbf{y} = (y_1, ...y_n)$, matrix $\mathbf{X}$ of predictors
Output: a sequence of linear models using an increasing number of attributes
Normalize $\mathbf{X}$ so that each line has mean 0 and standard deviation 1
$\lambda \leftarrow$ very large positive value
$\mathbf{r} \leftarrow \mathbf{y}$ `// residual`
$\mathbf{w} \leftarrow ()$ `// solution to the LASSO`
$\mathcal{A} \leftarrow \{\}$ `// set of selected attributes`
$\tilde{\mathbf{X}} \leftarrow []$ `// submatrix of` $\mathbf{X}$ `with selected attributes`
$\mathbf{s} \leftarrow ()$ `// vector of the signs of correlations of selected attributes`
while *not (some stopping criterion or* $\lambda = 0$*)* **do**
  $\mathbf{\Delta w} \leftarrow \left(\tilde{\mathbf{X}}^\mathsf{T}\tilde{\mathbf{X}}\right)^{-1}\mathbf{s}$
  $\mathbf{\Delta r} \leftarrow \mathbf{\Delta w}^\mathsf{T}\tilde{\mathbf{X}}$
  $\Delta\lambda \leftarrow$ lowest positive among
    1. λ
    2. $-\frac{w_j}{\Delta w_j}, j \in 1..K$
    3. $\frac{\lambda - \langle \mathbf{x}^i, \mathbf{r}\rangle}{1 - \langle \mathbf{x}^i, \mathbf{\Delta r}\rangle}, i \in 1..P, i \notin \mathcal{A}$
    4. $\frac{\lambda + \langle \mathbf{x}^i, \mathbf{r}\rangle}{1 + \langle \mathbf{x}^i, \mathbf{\Delta r}\rangle}, i \in 1..P, i \notin \mathcal{A}$
  $\mathbf{w} \leftarrow \mathbf{w} + \Delta\lambda\mathbf{\Delta w}$
  $\mathbf{r} \leftarrow \mathbf{r} - \Delta\lambda\mathbf{\Delta r}$
  $\lambda \leftarrow \lambda - \Delta\lambda$
  switch $\Delta\lambda$ *from* **do**
    case (2)
      remove j-th element from $\mathbf{w}, \tilde{\mathbf{X}}, \mathbf{s}, \mathcal{A}$
      $K \leftarrow K - 1$
    case (3)
      append $\mathbf{x}^i$ to $\tilde{\mathbf{X}}$, 0 to $\mathbf{w}$, $+1$ to $\mathbf{s}$, i to $\mathcal{A}$
      $K \leftarrow K + 1$
    case (4)
      append $\mathbf{x}^i$ to $\tilde{\mathbf{X}}$, 0 to $\mathbf{w}$, -1 to $\mathbf{s}$, i to $\mathcal{A}$
      $K \leftarrow K + 1$

Let $\mathbf{X}$ be a $P \times n$ matrix representing n sampled elements from $\mathcal{D} \equiv \mathbb{R}^P$. $\mathbf{x^i}$ contains the value of the i^{th} attribute for all n elements, and $\mathbf{x_i}$ contains the value of the attributes of x_i.

Let us consider an estimator $\hat{y} \equiv \mathbf{X}^T\mathbf{w}, \mathbf{w} \in \mathbb{R}^P$. The Least Absolute Shrinkage and Selection Operator (LASSO) consists in minimizing the squared l_2 norm of the residual subject to a constraint on the l_1 norm of $\mathbf{w}$:

$$\begin{aligned} \text{minimize } \|\mathbf{y} - \mathbf{X}^\mathsf{T}\mathbf{w}\|_2^2 &\equiv \sum_{i=1}^{n}(y_i - \langle \mathbf{x_i}, \mathbf{w}\rangle)^2 \\ \text{s.t.} \quad \|\mathbf{w}\|_1 &= \sum_{j=1}^{P}|w_j| < c \ , \end{aligned}$$

which is equivalent to

$$\text{minimize } \|\mathbf{y} - \mathbf{X}^\mathsf{T}\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_1 \tag{1}$$

The union of a convex loss function and a linear regularization term has the property that the greater λ, the sparser the solution (the more components of $\mathbf{w}$ are zero), hence the *Selection* property of the LASSO.

Efron *et al.*, have shown that the whole set of solutions to the LASSO (that is $\mathbf{w}(\lambda), \forall\ \lambda \in [0, +\infty)$) can be efficiently computed in an iterative fashion by the Least Angle Regression (LARS)[1], providing the l_1 regularization path.

In a few words, the LARS is based on the fact that this set of solutions $\mathbf{w}(\lambda)$ is continuous and piecewise linear w.r.t. λ, and characterized by the equicorrelation of all selected attributes:

for all $\lambda \in [0, \infty)$, let $\mathbf{w}$ be the solution of eq. (1), we have:

$$\forall i \in 1..P, \begin{cases} w_i \neq 0 & \Rightarrow & |\langle \mathbf{x}^i,\ \mathbf{y} - \mathbf{X}^\mathsf{T}\mathbf{w}\rangle| = \lambda \\ w_i = 0 & \Rightarrow & |\langle \mathbf{x}^i,\ \mathbf{y} - \mathbf{X}^\mathsf{T}\mathbf{w}\rangle| \leq \lambda \end{cases} \tag{2}$$

This implies that from any point of the path, the evolution of $\mathbf{w}(\lambda)$ as λ decreases is linear as long as no new attribute enters the solution and none leaves it.

For $\lambda = 0$, the solution of the LASSO problem turns out to be the least-square solution: weights are not penalized, so that all attributes may be used in $\hat{y}$. For $\lambda = +\infty$, in order to have a finite value of the objective function, all weights should be set to 0: no attribute is selected. The idea of the LARS is actually to start with $\lambda = +\infty$, hence all weights set to 0, and no attribute in $\hat{y}$ (only a bias: $\hat{y} = w_0$). At that point, the LARS is selecting the attribute which is most correlated with the residual ($\mathbf{r}$ in Algo. 1); this attribute being found, it enters the "active" set ($\mathcal{A}$), and its weight is set so that the new residual is equi-correlated with this weighted attribute, and the next attribute to enter the active set. Each time an attribute enters the active set corresponds to a certain value of λ (attributes belonging to the active set have a non zero weight, thus are all equi-correlated with the residual, since Eq. (2) holds)). In some cases, an active attribute may also leave the active set. When P terms are involved in $\hat{y}$, the algorithm may stop. But, the LARS can also be stopped as soon as a certain proportion of the attributes are used in $\hat{y}$, providing a certain accuracy, *e.g.*, measured on a test set. So, at each time, the estimator is of the form $\hat{y}(x) \equiv \sum_{a\in\mathcal{A}} w_a a(x)$, where a denotes an attribute, $a(x)$ the value of the attribute a of the data x, and $\mathcal{A}$ the active set.

We can not describe with much more details the algorithm as well as its properties, and refer the interested reader to [1]. The resulting procedure is sketched as Algorithm 1. This algorithm is computationally efficient, since its cost is quadratic in the size of the active set, which is upper bounded by the number of attributes P, and linear in P.

2.3 Featurising the LARS algorithm

For the moment, data have been represented by their attributes. This representation being quite arbitrary, we can actually represent data in many other way. A principled way to achieve this is to introduce a kernel. Let us note $g : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$ such a kernel function. A good intuition of a kernel is a way to measure the dissimilarity between two data. Such a function may have, and generally has, some (hyper)-parameters $\boldsymbol{\theta}$. Setting these parameters to some value, we may represent a data by $(g(\boldsymbol{\theta}, \mathbf{x_1}), ...g(\boldsymbol{\theta}, \mathbf{x_n}))$. We may also use various values of the parameters, various g functions, and we end-up with data being shattered in a space with a much higher dimensionality than the original one; we denote this dimension by M. Then, each data is represented by such M features, instead of the $P \ll M$ original attributes. There is absolutely no problem to use this representation instead of the original one, and use it in the LARS algorithm. This has already been investigated as the "Kernel basis pursuit" (see [4]). The hyper-parametrizations are chosen based on some expertize, or using a heuristic. This clearly do not guarantee that the parametrization is optimal.

The problem is that M may be quite large, so that the $M \times M$ matrix $\mathbf{X}$ may become huge. In particular, it is difficult to set the hyper-parameters *a priori*, so that we end-up having to consider kernels with different parameter settings as different kernels. This becomes cumbersome.

However, instead of that, we can consider one kernel function parameterized by its hyper-parameters, and let the algorithm find their best values. We would therefore restrict considerably the number of attributes per data to consider, hence the size of the matrix $\mathbf{X}$, having to deal with a minimization problem instead. This is precisely what we propose in this paper. Based on the kernelized-LARS, the next section details this point.

3 The Equi-Correlated Network algorithm

We now want to be able to perform kernel hyper-parameters automatic tuning. In short, with regards to Algorithm 1, the matrix of predictors $\mathbf{X}$ can no longer be constructed explicitly, since it would contain an (non denumerable) infinite number of rows, and columns. In this section, we describe our algorithm, ECON. Then, we discuss some issues related to the approximate minimization being done in ECON.

3.1 ECON

The core of the LARS algorithm consists in computing, at each step, the minimum positive value of a function on a finite support, that is, steps 3 and 4 in the while loop in Algorithm 1. We have to replace:

$$\min_{i \in 1..P} \left(\frac{\lambda - \langle \mathbf{x}^i, \mathbf{r} \rangle}{1 - \langle \mathbf{x}^i, \boldsymbol{\Delta}\mathbf{r} \rangle} \right)^+ \text{ and } \min_{i \in 1..P} \left(\frac{\lambda + \langle \mathbf{x}^i, \mathbf{r} \rangle}{1 + \langle \mathbf{x}^i, \boldsymbol{\Delta}\mathbf{r} \rangle} \right)^+$$

by

$$\min_{\boldsymbol{\theta}} \xi_+(\boldsymbol{\theta}) \equiv \left(\frac{\lambda + \langle \boldsymbol{\phi}(\boldsymbol{\theta}), \mathbf{r} \rangle}{1 + \langle \boldsymbol{\phi}(\boldsymbol{\theta}), \boldsymbol{\Delta}\mathbf{r} \rangle} \right)^+$$

and

$$\min_{\boldsymbol{\theta}} \xi_-(\boldsymbol{\theta}) \equiv \left(\frac{\lambda - \langle \boldsymbol{\phi}(\boldsymbol{\theta}), \mathbf{r} \rangle}{1 - \langle \boldsymbol{\phi}(\boldsymbol{\theta}), \boldsymbol{\Delta}\mathbf{r} \rangle} \right)^+$$

where $\boldsymbol{\phi}(\boldsymbol{\theta}) = \big(g(\boldsymbol{\theta}, \mathbf{x_1}), \ldots, g(\boldsymbol{\theta}, \mathbf{x_n})\big)^\mathsf{T}$, and

$$(x)^+ \equiv \begin{cases} x \text{ if } x \geq 0 \\ +\infty \text{ if } x < 0 \end{cases}$$

If g is continuous and differentiable, ξ_+ and ξ_- are continuous and differentiable everywhere except at rare unfeasible points.

The minimization can actually be conducted over the function $\xi(\boldsymbol{\theta}) = \min\big(\xi_+(\boldsymbol{\theta}), \xi_-(\boldsymbol{\theta})\big)$, which is also continuous and looses differentiability only at the frontiers where $\arg\min\big(\xi_+(\boldsymbol{\theta}), \xi_-(\boldsymbol{\theta})\big)$ changes.

Thus, the main task becomes to minimize, at each step, a relatively smooth function of $\mathbb{R}^l$, l being the number of hyper-parameter of *one* unit of the network, that is, $l = |\boldsymbol{\theta}|$. This comes in striking contrast to the usual minimization task for sigmoidal networks that is done in the space of *all* weights of the network. This minimization problem is discussed in section 4.2. Having no closed-form solution, we have to contend ourselves either with a local minimum, or an approximate value of a global minimum. We investigate this issue in the next section.

3.2 The risks of approximate minimization and some workarounds

When applying the LARS algorithm on a finite and computationaly reasonable number of attributes/features, one can perform an exact minimization of the step $\Delta\lambda$, thus computing the exact path of solutions to the LASSO and benefiting safely from its nice properties.

As ECON performs an approximate minimization over $\mathbb{R}^l$, some features may be missed and attain a higher correlation than the one of active features. Two distinct cases should be considered: either the selected feature is in the immediate neighbourhood of the minimizer, which has been missed only by the limited precision of the minimization algorithm, or it is in the neighbourhood of a local but not global minimum.

In the first case, the missed features can generally safely be considered as being equivalent to the one selected. Although they have and may keep a correlation higher than λ, this correlation will stay in the neighbourhood of λ, as the correlation is a continuous function of $\boldsymbol{\theta}$. An illustration of the harmless nature of such misses is the fact that RBF networks are succesfully used with predefined features of which the centers form a regular grid over $\mathcal{D}$, and the bandwidth is also common, and set *a priori*. In our algorithm, the missed features can be considered as if they had been deliberately left out from the start, and still represent by far less than the gaps in between the grid and list of bandwidth in such algorithms.

The second case, where a local but not global minimum is found, is more critical. A specific feature is missed and probably will not be recovered in subsequent steps. Indeed, the principle of selecting features by searching for the first one that becomes equicorrelated as λ decreases, precludes from finding a feature already more correlated than active ones. It is generally specific and not represented by a similar active feature, and its correlation should decrease slower than λ or even increase, and the value of ξ for this feature will be negative, meaning it *was* equicorrelated at a previous point in the regularization path. We propose a workaround that has yet no strong theoretical guarantees against possible side effects, but has proven its efficiency in experiments.

The first idea is to relax the positiveness condition about $\xi(\boldsymbol{\theta})$, applying this constraint only to its denominator, which implies that at a given step on the path, we also look for features with an absolute correlation greater than λ, *i.e.*, missed in previous steps. The relaxed constraint separates the two causes for negativeness of ξ: an absolute correlation greater than λ, or the fact that it cannot reach λ along the current direction of weight change. The second idea is that when such a feature is found, instead of trying to ride the regularization path back to the point where it was missed, the feature is incorporated at the present point (λ does not change), and, to keep up with the soundness of the algorithm, this feature is considered having been penalized from the start, so that the right point to include it when solving the weighted LASSO *is* the present point. By penalization and weighted LASSO, we mean the following:

$$\text{minimize } \sum_{i=1}^{n}(y_i - \langle \mathbf{x_i}, \mathbf{w} \rangle)^2 + \lambda \sum_{j=1}^{M} p_j |w_j|$$

where p_j's are penalization factors assigned to each weight beforehand. This implies that the path of solutions is now characterized by the correlation of all active features being equal in absolute value to λ *times their penalization*. Thus, when a feature is forgotten and caught back at a later point, we set this point to be the legitimate one by assigning to the feature a penalization coefficient equal to its absolute correlation divided by λ.

4 Practical implementation

In this section, we deepen some implementation related issues about ECON.

4.1 The bias

It is useful to add a bias term w_0 in the general model $\hat{y}$. This can be seen as a weight associated to the particular feature constantly equal to 1. Unlike regular features, it cannot be normalized, as its standard deviation is zero. But its particularity and uniqueness allows it to have a dedicated treatment as: we can use the notion of penalization again and decide it will be almost not penalized — no penalization at all would fail the computations. The algorithm starts by setting λ to a very large value, by far larger than the correlation of any regular feature, and arbitrarily decide that this extreme point of the regularization path is the one to include the bias, by setting its penalization to $1/\lambda$. In the following of the algorithm, the deactivation/reactivation of this feature need not be reconsidered.

4.2 Optimization algorithm

As an optimization algorithm to apply at each step, DiRect [7] appears to be a good choice for several reasons. DiRect minimizes a function over a bounded support by recursively dividing this support into smaller hyperrectangles (boxes). The choice of the box to split is based on both the value of the function at its center and the size of the box.

The first appealing property is that it can limit the search to a given granularity—by setting a minimal size for the boxes—which is sufficient for our needs: we do not seek a precise minimum, but rather to ensure that the region of the global minimum is found,

and DiRect has proven to be good at the global search level but rather slow for local refinement.

The second reason is that it is not gradient-based, although it needs and exploits smoothness of the function to minimize. Despite the fact that ξ inherits the differentiability property of g almost everywhere, its gradient shows noticeable discontinuities at non-differentiable points. ξ is continuous and regular nonetheless, except at unfeasible points, which can be handled by DiRect, by systematically dividing rectangles of which the center is unfeasible to ξ.

An objection to the use of DiRect could be the constraint to restrict the search to a bounded support. This is not an issue for RBF, as the hyper-parameters consist in the coordinates of the center and the bandwidth parameters. The first ones can be naturally bounded by the bounds of the training points, and the latter ones can be bounded by the width of the training set.

4.3 Sigmoidal networks

When applying the algorithm to feed-forward sigmoidal neural networks, the hyper-parameters gathered in $\boldsymbol{\theta}$ are the weights between the inputs and a unit, and the bias of that unit; thus $l = P + 1$. g is set to the sigmoid function

$$g(\boldsymbol{\theta}, \mathbf{x}) = \frac{1}{1 + e^{\langle \boldsymbol{\theta}_{1..P}, \mathbf{x} \rangle + \theta_{P+1}}}$$

As stated in the previous subsection, $\boldsymbol{\theta}$ can be constrained in about $[-100, 100]^l$ after normalization of inputs. In our experiments, we used $[-146.41, 146.41]$ with a non uniform granularity: possible values for a parameter were set to $\{\left(\frac{\alpha}{10}\right)^2 \mid \alpha \in -121, -120, \ldots + 120, +121\}$, which makes the granularity thinner for small values, to which the output of the sigmoid is more sensible.

The critical issue is regarding conditionning (computation precision?) and lies in the normalization of a unit's output. This means dividing it by the standard deviation of the feature's values on the training set. For some values of $\boldsymbol{\theta}$, the standard deviation is almost zero, or even considered zero at a machine level. A workaround is to add a small value to the variance, but it appeared that the choice of this value can lead to a delicate trade-off between computational unstability and bias.

4.4 ECON-RBF networks

We name "Equi-Correlation RBF networks" (ECON-RBF) the case in which Gaussian kernels are used in ECON. ECON-RBF offer far more expressiveness than classical fixed RBF networks. In the latter, the units are usually set in advance, with a single and common bandwidth, and centers forming a grid in $\mathcal{D}$. ECON-RBF allow not only to choose on the fly the centers in the whole domain (or actually among a dense grid), but also to select for each unit specific bandwidth(s), common to all dimensions or not, or a whole correlation matrix:

$$g = e^{-\frac{1}{2}(\mathbf{x}-\mathbf{c})^\mathsf{T} \boldsymbol{\Sigma} (\mathbf{x}-\mathbf{c})}$$

$$\text{with } \mathbf{c} = (\theta_1, \ldots, \theta_P)^\mathsf{T}$$

$$\text{and } \mathbf{\Sigma} = \begin{pmatrix} \ddots & & \mathbf{0} \\ & \theta_{P+1} & \\ \mathbf{0} & & \ddots \end{pmatrix}$$

$$\text{or } \begin{pmatrix} \theta_{P+1} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \theta_{P+P} \end{pmatrix}$$

$$\text{or } \begin{pmatrix} \theta_{P+1} & \dots & \theta_{P+P} \\ \vdots & \ddots & \vdots \\ \theta_{P+P} & \dots & \theta_{P+P(P+1)/2} \end{pmatrix}$$

Once again, setting distinct penalization coefficients on the features can be beneficial. The straight, unweighted, penalization of $\|\mathbf{w}\|_1$, regularizes the model by limiting the number of active features. When using radial features, another interesting way of regularization is to favor large bandwidths or, at least, penalize or avoid small bandwidths that could fit the noise. If features are normalized, all of them are treated equally, and such overfitting features can be selected sooner than desired in the regularization path. By specifying a penalization factor related to the width of a feature, both regularization schemes operate, and the successive models include more features, which tend to have a smaller support.

A nice setting for the penalization is the inverse of the standard deviation of the feature's outputs. It is strongly related to the feature's width, and its implementation simply consists in not normalizing the features' standard deviations.

4.5 Stopping criterion

The algorithm constructs a sequence of models that goes from an infinitely-regularized one ($\mathbf{w} = \mathbf{0}$) to a non-regularized one (the least squares solution when it exists, or a solution with residual $\mathbf{0}$). Good models naturally lie in between, at some points that remain to be determined. Identifying these points is an open issue, and no general and automated criterion has made a consensus yet. We are currently working on possibly interesting ways of identifying a transition between fitting and overfitting, but meanwhile, in the experiments exposed below, we used a simple yet satisfying procedure: we split the sample set into a training set and a validation set, compute a sequence of models until the number of selected features is larger than one can expect to be needed, and select the one on which the residual over the validation set is the smallest.

4.6 Incremental regression

One last remark about the fact that ECON, likewise the original LARS, may very easily be turned into an incremental algorithm. Indeed, new examples may be added after an estimator has been computed. This estimator is then updated with the newly available examples. Even though we have not experienced this possibility yet, it should be possible to deal with non stationary y.

5 Experiments

In this section, we provide some experimental results to discuss ECON. We have investigated three variants of ECON-RBF: a fixed bandwidth for all kernels (only the center of the Gaussian is tuned); individually tuned bandwidth for each kernel, using the same same variance in each direction (diagonal covariance matrix); individually tuned bandwidth for each kernel, and in each dimension (covariance matrix is still diagonal though). The latter version provides the best results, for a computational costs that is not much significantly higher than the other two variants, so we only report the results obtained with this variant.

5.1 Synthetic data sets

5.1.1 The noisy sinc function

The Equi-Correlation Network algorithm was run with Gaussian RBFs on noisy samples of the sinc function, in order to visualize the functions that it minimizes, and illustrate the benefit of letting bandwidths of each Gaussian be automatically tuned. Fig. 1 represents the approximation obtained when the number of features in $\hat{y}$ has reached 9, as well as each of these 9 features. First, one can notice that the approximation $\hat{y}$ is very close to the function to approximate; second, we also notice that ECON actually uses kernels with different hyper-parameters. To exemplify the optimization of ξ, Fig. 2 shows the function that the algorithm has to minimize at the third iterations.

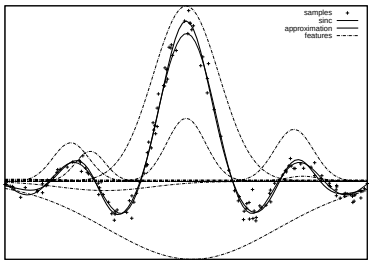

Figure 1: Approximation of noisy sinc after 12 steps, with 9 features. The dashed lines show these features, scaled by their weight in the network ($\hat{y}$ is represented by the bold line). Notice how the negative parts are approximated by means of a substraction of a large feature from a medium one.

5.1.2 The $\cos(\exp \omega x)$ function

The $\cos(\exp \omega x)$ is a toy problem in which we try to learn the function $f(x) \equiv \cos e^{\omega x} + \eta$, with $x \in [0, 1]$, and η a noise with variance $\sigma_\eta = 0.15$. We set ω to 4 so that the function wiggles in the domain (see Fig. 3). This problem is used in [4]; it illustrates the fact that an algorithm is able, or not, to select useful parametrizations of the kernel. We stick to Guigue *et al.* experimental setting: the training set is made of 400 points drawn uniformly at random in $[0, 1]$, and the test set is made of 1000

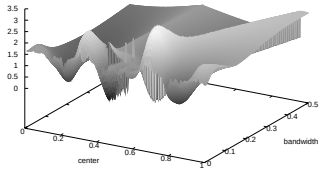

Figure 2: Plot of the function to minimize at third step of sinc approximation: the z-axis represents the amount λ should decrease for the Gaussian with center x and bandwidth σ to appear in the LASSO solution. The peaks correspond to the active features in the current $\hat{y}$, for which this function is unfeasible $(0/0)$.

points. In the training set, noise is added, whereas no noise is added in the test set. Fig. 3 shows very clearly that the fit is very good.

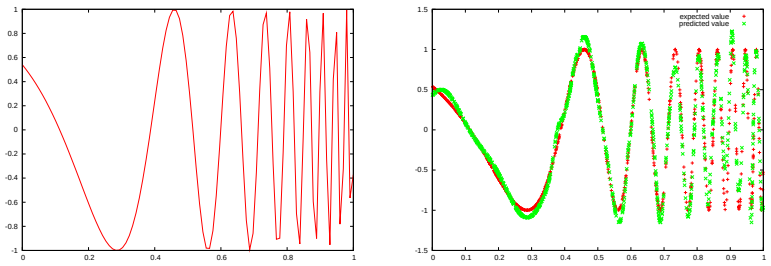

Figure 3: On the left, the cos-exp function to be learned. On the right, red points are the value to learn, while green points are the predictions. We see that the fit is quite very good.

5.1.3 Friedman's functions

Friedman's benchmark function were introduced in [2] and used quite widely. There are 3 such functions: F1 has $P = 10$ attributes, the domain being $\mathcal{D} = [0, 1]^{10}$. The function is noisy, and 5 of the attributes are actually useless. F2 and F3 are defined on some subset of $\mathbb{R}^4$ and are also noisy. For each of these problems, we generate training sets made of 240 examples, and the same test set made of 1000 data[1].

For these problems, we compare our results with those obtained by [12], on a LASSO algorithm (see table 1). The latter compares his own results with Support Vector Machine and Relevance Vector Machines. We measure the mean-squared error on the data-set:

[1]To generate the data, we use the implementation available in R, in the `mlbench` package, with the standard setting for noise.

Table 1: We compare the performance of ECON with those published in [12], concerning the Support Vector Machine, the Relevance Vector Machine, and a LASSO-based algorithm proposed in this paper. For each algorithm, we provide the accuracy measured on a test set of 1000 data / K (aka, the number of support vectors) in $\hat{y}$, averaged over 100 runs for ECON. Both the average, and the median are provided. See the text for more discussion.

FRIEDMAN'S FUNCTION	SVM	RVM	[12]	ECON	
	AVE. MSE	AVE. MSE	AVE. MSE	AVE. MSE	MEDIAN MSE
F1	$2.92/116.2$	$2.80/59.4$	$2.84/73.5$	$1.99/76$	$1.97/71$
F2	$4140/110.3$	$3505/6.9$	$3808/14.2$	$4845/54$	$4696/55$
F3	$0.0202/106.5$	$0.0164/11.5$	$0.0192/16.4$	$0.0115/53$	$0.0113/53$

$$\text{MSE} = \frac{\sum_{i=1}^{n} \left(y_i - \hat{y}(\mathbf{x_i})\right)^2}{n}$$

where $\bar{y} = \sum_{i=1}^{n} y_i/n$.

As ECON computes the whole regularization path, we compare the results obtained, in the same experimental settings, by ECON and the one proposed by [12] who obtains a solution with a certain number of terms in $\hat{y}$ (K).

On F1, ECON by far obtains the best results among the 4 algorithms. Only 17 terms provide an accuracy better than 2.8, the best accuracy mentioned for the other 3 algorithms. On F2, the results are less good, but are still rather good. To stick to the results published in [12], the figures in table 1 are averages; however, if we consider the best accuracy, or better, the histogram of accuracies, it is skewed towards better accuracies. On F3, we again obtain the best accuracy. It is also very significant to note that the number of kernels saturates at some point: on both F2, and F3, the mean number of kernels involved in $\hat{y}$ is 54, with a standard deviation of 12. This saturation in the number of terms, while the test error keeps on decreasing, is a sign ECON does not overfit, or to the least, do not overfit a lot.

5.2 Real-world regression problems

To further the comparison, we use the well-known Boston housing dataset used by [4], as well as two datasets that are used in [12], namely abalone, and house-price-8L datasets[2].

5.2.1 Boston housing

This dataset is made of 502 data, each made of 13 attributes. We use the same experimental setting as [4] with which we compare our results: 20 % of the dataset as a test-set, the other 80 % being used for training. On 50 runs, we clearly outperform their results when using a single kernel, with an average accuracy on the test-set of 10.91, standard deviation of 3.56 (the distribution is very skewed, with more than half runs in the range 6-10, see Fig. 4). We note that on average, 232 basis functions are used (ranges from 100 to 400), that is about 25% less than Guigue *et al.*

[2]For boston, we use the dataset from package `mlbench`, in R; for the other two, we use the datasets available at `http://www.cs.toronto.edu/~delve/delve.html`.

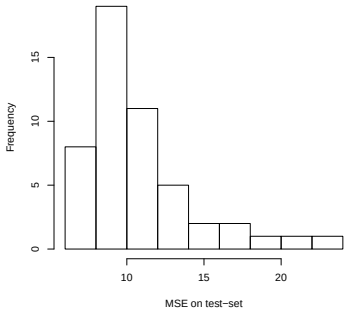

Figure 4: Histogram of the best MSE measured on a test-set, obtained on 50 different runs. We see that the distribution is far from normal, so that the average, and standard deviation mentionned in the text only give a very poor image of the reality.

Table 2: We compare the performance of ECON with those published in [12], concerning the Support Vector Machine, the Relevance Vector Machine, and a LASSO-based algorithm proposed in this paper. For Abalone, we provide the accuracy measured on a test set of 1000 data / the number of terms (aka, the number of support vectors) in $\hat{y}$, averaged on 100 runs for ECON. For house-price-8L, the training set is successively made of 1000, 2000, and 4000 data, the test set of 4000 data. See the text for more discussion.

DATASET	SVM	RVM	[12]	ECON
ABALONE	$4.37/972$	$4.35/12$	$4.35/19$	$4.31/71$, $4.30/100$
HOUSE-PRICE-8L, 1K	$1.062/597$	$1.099/33$	$1.075/61$	$0.57/20$
HOUSE-PRICE-8L, 2K	$1.022/1307$	$1.048/36$	$1.054/63$	$0.41/38$
HOUSE-PRICE-8L, 4K	$1.012/2592$	NA	$1.024/69$	$0.40/40$

5.2.2 Abalone and house-price-8L

In both datasets, data are made of 8 attributes; Abalone holds 4177 data, while house-price-8L has 22784 data. The results are provided in table 2. We see that ECON performs comparably to other methods on Abalone, much better than other methods on house-price-8L. Indeed, the accuracy of the test error is much better than the one published in [12], and the number of kernels is much smaller. As for Friedman's functions, regarding the distribution of accuracies, the same remarks are true here again: for instance on Abalone, the best accuracy with 71 kernels is 4; on house-price-8L, we obtain much better performance, with often sparser expansions.

6 Conclusion and perspectives

In this paper, we considered the regression problem, for which we have proposed an algorithm, the Equi-Correlated Network, inspired by the (kernelized-)LARS. ECON automatically tunes the hyper-parameters of the kernel functions. The resulting algorithm can be seen as a non parametric one-hidden layer perceptron, that is, a perceptron

in which the hidden layer does not contain a fixed amount of units, but grows according to the complexity of the problem to solve. Furthermore, building on the ability of the LARS to compute efficiently the whole l_1 regularization path of the LASSO, ECON does not provide one solution, but the whole family of possible solution, according to the regularization parameter. This is a very interesting ability for practical purpose, and the fact that the algorithm is very efficient provides even more interest to ECON. Furthermore, sticking to this idea of closeness to MLPs, using a non parametric approach in which the hidden layer is not fixed, this let the estimator adapts to the complexity of the problem, to get the best compromise between under- and over-fitting, something an MLP with a fixed architecture can not achieve.

There remains some work to do to fine tune ECON itself. The impact of the quality of the minimizations still has to be studied, both theoreticaly and experimentaly; some conditionning issues need to be clearly identified and solved; a good stopping criterion remains to be defined. It will also be important to investigate how this algorithm can be extended to more general loss and regularization functions, following the work of [11]. Finally, as pointed out in the introduction, ECON may be directly applied to classification problem, and this should be investigated, at least for an experimental assessment.

Nevertheless, first experiments exhibits state-of-the-art performances on two real-world regression problems. These results were obtained without the need to set hyperparameters from domain knowledge, cross-validations, or high expertise in the algorithm. The key reason lies in the fact that although the class of models has a high expressivity, the optimization task in the space of all parameters is performed by a sequence of optimization in *one* feature's parameters.

As we are mostly interested in control optimization and sequential learning problems, further researches will focus on how this algorithm can be extended to online learning, moving targets, and specific ways to embed it in reinforcement learning algorithms.

References

[1] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least-angle regression. *Annals of statistics*, 32(2):407–499, 2004.

[2] J. Friedman. Multivariate adaptive regression splines. *Annals of Statistics*, 19(1):1–82, 1991.

[3] V. Guigue. *Méthodes à noyaux pour la représentation et la discrimination de signaux non-stationnaires*. PhD thesis, Institut National des Sciences Appliquées de Rouen, 2005.

[4] V. Guigue, A. Rakatomamonjy, and S. Canu. Kernel basis pursuit. In *Proc. ECML*, volume 3720, pages 146–157. Springer, LNAI, 2005.

[5] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning — Data mining, inference and prediction*. Statistics. Springer, 2001.

[6] G-B. Huang, P. Saratchandran, and N. Sundararajan. A generalized growing and pruning RBF (GGAP-RBF) neural netowrk for function approximation. *IEEE Transactions on Neural Networks*, 16(1):57–67, January 2005.

[7] D.R. Jones, C.D. Perttunen, and B.E. Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Aplications*, 79(1):157–181, October 1993.

[8] G. Kimeldorf and G. Wahba. Some results on tchebycheffian spline functions. *J. Mathematical Analysis and Applications*, 33(1):82–95, 1971.

[9] S. Perkins, K. Lacker, and J. Theiler. Grafting: Fast, incremental feature selection by gradient descent in function space. *Journal of Machine Learning Research*, 3:1333–1356, 2003.

[10] J. Platt. A resource-allocating network for function interpolation. *Neural Computation*, 3(2):213–225, 1991.

[11] S. Rosset and J. Zhu. Piecewise linear regularized solution paths. *Annals of Statistics*, 35(3):1012–1030, 2007.

[12] V. Roth. Generalizd lasso. *IEEE Trans. on Neural Networks*, 15(1):16–28, January 2004.

[13] Sh. Shalev-Shwartz and N. Srebro. Low l_1 norm and guarantees on sparsifiability, July 2008. Sparse Optimization and Variable Selection, Workshop, ICML/COLT/UAI.

[14] R. Tibshirani. Regression shrinkage and selection via the lasso. *J. Royal Statistics*, 58(1):267–288, 1996.

[15] S. Vijayakumar, A. D'Souza, and Stefan Schaal. Incremental online learning in high dimensions. *Neural Computation*, 17(12):2602–2632, 2002.

Unité de recherche INRIA Futurs
Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399