Automatic synthesis of sparse neural networks using l_1 regularization

Manuel Loth[1,2,3] and Philippe Preux[1,2,3]

[1] Team-Project SequeL, INRIA Lille Nord-Europe
[2] LIFL (UMR CNRS)
[3] Université de Lille, France
{manuel.loth,philippe.preux}@inria.fr

Preliminary note

This is a paper that has been rejected by the program committee of ICANN'2009. Despite that, we find it interesting to make it available. We solemnly declare that **nothing is wrong, fake, ...** in this paper. May be it is badly written, very unpedagogical, maybe it is even useless to the scientific community, but anyway, I take the responsibility to put it as-is, on the web.

The interested reader may find more information on my website:
http://www.grappa.univ-lille3.fr/ ppreux/pmwiki/index.php?n=Review.Re

Philippe Preux

Automatic synthesis of sparse neural networks using l_1 regularization

Manuel Loth[1,2,3] and Philippe Preux[1,2,3]

[1] Team-Project SequeL, INRIA Lille Nord-Europe
[2] LIFL (UMR CNRS)
[3] Université de Lille, France
{manuel.loth,philippe.preux}@inria.fr

Abstract. We consider the LARS algorithm to solve the regression problem. This algorithm may be seen as iteratively building a one hidden layer perceptron. Along iterations, the hidden layer grows, and shrinks, according to the data, to provide an optimal balance between the compactness of the network, and its accuracy. In most cases, the hidden units have parameters that should be set beforehand, and *a priori*. In this paper, we propose ECON, a new LARS-based algorithm that automatically handles the selection of an optimal parametrization of the hidden units.

We consider supervised learning, in particular regression problems. Given a set of N examples $(\mathbf{x_i}, y_i)_{i\in\{1,\dots,N\}}, \mathbf{x_i} \in \mathcal{D} \subset \mathbb{R}^P, y_i \in \mathbb{R}$, we want to investigate algorithms to build 1 hidden layer perceptrons (1HLP) to accurately predict the label $y(\mathbf{x})$ of any data $\mathbf{x} \in \mathcal{D}$. By building, we mean find a set of hidden units, along with their hyper-parameters, and the weights on the connexion between units. We want the 1HLP to be as small (or sparse) as possible, that is, we want the hidden layer to contain the smallest number of units. Actually, we wish that for a given number of hidden units, we get the best possible accuracy. From a user point of view, we want an algorithm which is efficient, and easy to use: we wish to avoid, as much as possible, the burden of hand-tuning parameters and hyper-parameters, and any assumption that would break its application in a sequential setting. Our choice of a 1HLP architecture is motivated by the fact that these are easy to interpret for a human being; indeed, once build on a dataset, the neural network (NN) has extracted an information about the structure of the dataset, and encoded it into the hidden layer; so, it is highly interesting to be able to interpret this information, and a 1HLP with a linear output is probably the simplest structure that we can understand: in this case, the NN outputs $\hat{y} \equiv \sum_{k=1}^{k=K} w_k \phi_k$ where the w_k's are the weights between each of the K hidden units and the output unit, and $\phi_k : \mathbf{x} \mapsto \phi_k(\mathbf{x}) \in \mathbb{R}$ is the output of the k^{th} hidden unit. The building algorithm acts as a feature extractor which are encoded in the hidden units, so that we will also use the term "feature" to name each ϕ_k; the magnitude of its weight $|w_k|$ then provides an information about the importance of this feature to predict the label of a data. So, when we say we look forward small 1HLP, that means we look for a 1HLP with K as small as possible. To finish with our motivations in this work, we are also

interested in being able to work on sequential data, or to the least, on sequences of batches of data. In this case, the distribution of data in $\mathcal{D}$ may evolve along time, without notice.

The design of training algorithms to accomplish this growth is not obvious in particular to avoid overfitting. To avoid overfitting, a general approach consists in using a regularization term, such as $l_2(1HLP) \equiv l_2(\mathbf{w}_{1HLP}) \equiv \sum_k w_k^2$, the l_2 norm of its weight vector. Using an l_1 norm instead of the l_2-norm has been recently recognized to provide very sparse solutions, avoiding many terms with very small weights in the expansion [1]. It is defined by: $l_1(1HLP) \equiv l_1(\mathbf{w}_{1HLP}) \equiv \sum_k |w_k|$. The LARS is an algorithm that builds such 1HLPs in a principled way [2]; the LARS was generalized (kernelized) under the name of "kernel basis pursuit" [3]. Since the background algorithm is really the same, we will use the term LARS to designate both. As the basis of our work, the LARS will be detailed below; let us just mention here that it provides an exact solution to the problem we wish to solve, it is efficient, and we have many experimental evidences that it does provide very sparse, yet accurate, solutions. Assuming the features are multivariate Gaussians (RBF), the hyper-parameters of the features have to be set: the center and the covariance matrix in general, that is $p \equiv P + P^2$ real numbers to set for this kind of features. To grow a sparse network, these parameters should be carefully set. This is precisely the problem we tackle in this paper which is not currently handled by the LARS. To this end, we introduce a new algorithm named ECON which grows such an 1HLP, providing a sparse expansion, including automatic hyper-parameter tuning of the features.

The rest of this paper is organized as follows. Section 1 briefly reviews the relevant litterature. In section 2, we introduce the necessary material on the LARS algorithm. In section 3, we present the extension of the LARS to ECON. Sec. 4 provides experimental results that show state of the art performance on regression problems for ECON. Finally, section 5 concludes, and draws future lines of work.

1 Growing networks for supervised learning

In this section, we review algorithms that have been proposed to grow a neural network in a classification, or regression setting. Typically, these algorithms begin with a mere input layer, and an output. Then, as data are acquired, hidden units are added to the network, and also removed for some of them.

Cascade-correlation networks (CCN) [4] are very well-known such growing networks; they are efficient, but their cascaded architecture makes them very difficult to interpret. Resource-Allocating Network (RAN) [5] grows according to the "novelty" of data. RAN has been improved later. However, both CCN and RAN can only grow, and never shrink; both may become large. This inability to shrink is a severe shortcoming. GGAP-RBF [6] is a follow-up to these algorithms, designed to work in a sequential setting; however, though left unmentioned in this paper, information about the distribution of data are necessary for GGAP-

RBF which are not easy to obtain in a really sequential setting furthermore, it is unclear how GGAP-RBF may deal with data which distribution evolves in time. These works are closely related to support vector machines (SVM) in their incremental version, which builds such a linear expansion of weighted support vectors [7]. So, the result of an SVM has exactly the same structure as a 1HLP, the support vectors being the hidden units. SVM are appealing in that they produce a globally optimal solution; despite tremendous achievements in supervised learning, and a conceptual renewal in the way to consider these problems as a nicely formalized and solved optimization problem, the solution found is not necessary sparse, the computation cost may be important, the choice of the kernel is not obvious, not mentioning the selection of its hyper-parameters. More generally, kernel methods provide a general approach to supervised learning by building linear feature expansions, which often fits rather well a sequential setting. Let us also mention Locally Weighted Projected Regression (LWPR) [8] as an other approach to growing NNs.

In the next section, we concentrate ourselves on yet an other algorithm, namely the LARS, which is the basis of all our work.

2 The LARS algorithm

Given a regression problem, we want to build an accurate and sparse 1HLP. Along with K, we look for the best set of weights $w_{1,..,K}$, and the best set of units $\phi_{1,..K}$ altogether. We may consider a set of potential hidden units Φ among which we look for the best subset to make the hidden layer of a 1HLP. The objective is that this 1HLP minimizes a linear combination of its training error, and its complexity, or size, of the 1HLP:

$$\zeta(\text{1HLP}) = E_{train}(\text{1HLP}) + \lambda \; l_1(\text{1HLP}),$$

where $E_{train}(\text{1HLP}) \equiv \frac{1}{N} \sum_{i=1}^{i=N} (\hat{y}(\mathbf{x_i}) - y_i)^2$. Minimizing ζ is known as the LASSO problem. λ is the regularization constant: if large, the term measuring the complexity of the 1HLP will be given a large importance, thus the minimization will typically yield a sparse 1HLP which may exhibit a large error; to the opposite, if λ is small, the training error may be very small, but the 1HLP is likely to overfit, and the 1HLP may include a large number of units. The optimal value of λ being very difficult to set beforehand, the LARS algorithm computes all the NN for all values of $\lambda \in \mathbb{R}^+$; surprisingly, this algorithm is very efficient, and optimal.

The LARS draws from the homotopy idea: we wish to minimize $E_{train}(\text{1HLP})$ which is a non trivial task. To the opposite, we know how to minimize $l_1(\text{1HLP})$: simply take $K = 0$, that is, use no hidden unit! This solution corresponds to $\lambda = +\infty$. At this point, it is reasonable to include a bias unit which always outputs 1, connected to the output unit by its weight $w_0 \leftarrow \frac{1}{N} \sum_i y_i$. Starting from this solution, the LARS looks for a correction to apply to λ which corresponds to having a single hidden unit while minimizing ζ. To this end, all potential units are checked, and the best one enters the hidden layer, with a suitably adjusted

weight; this unit becomes "active" and enters the active set $\mathcal{A}$ made of all units used in the NN. The best one is the one that is most correlated to the residual error. A crucial point here is that the associated weight is not set so as to correct the error as much as this unit could do it, but so that the next feature to enter the 1HLP will be as much correlated with the residual error than this first unit. This fundamental point is quite tricky and we refer the interested reader to the seminal paper [2]. Then, the process iterates, by searching for the next best feature to enter the 1HLP, and the associated variation of the regularization constant λ. Though the algorithm computes a solution for a finite set of values of λ, the reader should keep in mind that all the values of $\lambda \in \mathbb{R}^+$ are legal; actually, between two successive iterations of LARS, the set of active features does not change, but their weights vary linearly, as a function of λ; so, knowing the value of the weights for two successive values of λ, it is easy to compute the weights for any value of λ in this interval, and, finally, at the completion of the LARS, we have them for any $\lambda \in \mathbb{R}^+$. LARS may iterate either until λ gets to 0, but this would yield non sparse solutions since at this point, the complexity of the 1HLP would not be penalized any longer, and all features would be used. In practice, one uses a stopping criterion, such as until the error measured on a test set reaches a minimum. Originally, in the LARS, Φ was restricted to the set of raw attributes of the data; later, kernel functions have been used to enrich this set of potential features in a principled way, and yielded the kernel basis pursuit algorithm [3]. Trickier, as the LARS iterates, some active feature may become inactive. So, the LARS has the very interesting property that it provides a principled algorithm to build a 1HLP that grows and shrinks to meet the complexity of the data. The computational complexity of the LARS is linear in the number of potentials features and quadratic in the number of active features which tends to remain small as will be seen in the experimental section. The LARS is sketched in Algorithm 1[1].

3 Our proposal: ECON

Since all potential units are tested to enter the 1HLP at each iteration of the LARS, needless to point out that if the number of potential features is large, each iteration will be costly (even if the complexity is linear in the number of potential units), and in any case, should remain finite. However, the features generally have some hyper-parameters, and we have to select a finite set of hyper-parametrizations *a priori* to apply the LARS. In pratice, the choice of this set is not easy. We want to go beyond this limitation, and be able to consider all possible hyper-parametrizations, which typically define a continuous space, for

[1] We use the following notations: i indexes examples, k indexes hidden units; $a \equiv b$ means "a is defined by the concept b", whereas $a \leftarrow b$ is the assignment operator of procedural programming language, that is: the value of the variable named a gets the current value of b; in this case, the value of a does not change whether the value of b changes later, or not; in the former case ($\equiv$), the value of a is always the same as the value of b. Boldfaces, as in $\mathbf{x}$ indicates vectors.

Algorithm 1 Sketch of the LARS algorithm.

Require: A regression problem defined by N examples $(\mathbf{x_i}, y_i), \mathbf{x_i} \in \mathcal{D} \subset \mathbb{R}^P, y_i \in \mathbb{R}$.
Require: A set of potential features Φ.
$\lambda \leftarrow +\infty$.
Set of active units: $\mathcal{A} \leftarrow \emptyset$.
Set of potential units: $\mathcal{P} \leftarrow \Phi \backslash \mathcal{A}$.
Compute the bias: $w_0 \leftarrow \frac{1}{N} \sum_i y_i$ and set $\phi_0 \equiv \mathbf{1}$ (identity function).
Number of active units: $K \equiv |\mathcal{A}|$.
Let $\hat{y} \equiv \sum_{k=0}^{k=K} w_k \phi_k$.
while stopping criterion not fullfilled **do**
  compute the residual $\mathbf{r}$ on the training set (*i.e.*, $r_i \leftarrow y_i - \hat{y}(\mathbf{x_i})$).
  Check whether the weight of an active unit has been nullified; if yes, remove it from $\mathcal{A}$, and put it back in $\mathcal{P}$;
  otherwise, select $\phi_{K+1} \equiv \phi^*$ the unit among $\mathcal{P}$ which is most correlated with $\mathbf{r}$.
  Compute its weight w_{K+1}, add ϕ_{K+1} to $\mathcal{A}$, remove it from $\mathcal{P}$.
  Update the weights of all active units, set $K++$, update λ.
end while

instance of dimension $P + P^2$ for multivariate Gaussian units. To that end, we propose an algorithm named ECON that precisely aims at selecting the best feature to enter the 1HLP, along with its best hyper-parametrization.

3.1 The principle of ECON

The key difference between ECON and the LARS lies in the function that is minimized in order to select the best unit to enter the 1HLP at each iteration, and how the minimization is performed. Otherwise, ECON retains the same principle of iterating over λ, starting from $+\infty$, and down to a certain value which corresponds to the stopping criterion used in the algorithm. In the LARS, the set of potential features Φ is finite. Let $\mathbf{r} \in \mathbb{R}^N$ the vector of residual, that is $r_i = \hat{y}(\mathbf{x_i}) - y_i$ for the current 1HLP. At each iteration, LARS selects the feature $\phi^* \in \Phi$ which maximizes the correlation between ϕ and $\mathbf{r}$. Since Φ is finite, this is merely done by enumerating Φ. Now, let us turn to features that have p hyper-parameters. We denote them as $\boldsymbol{\theta} \in \Theta \subset \mathbb{R}^p$, so that features are really $\phi_{\boldsymbol{\theta}}$, where the $\boldsymbol{\theta}$ is a free parameter which value has to be set. So, now, we really look for the optimal combination $(\phi^*, \boldsymbol{\theta}^*)$, and there is thus a non denumerable number of potential units to consider.

Technically, the LARS solves:

$$\min\{ \min_{i \in 1..N} \left(\frac{\lambda - \langle \mathbf{x_i}, \mathbf{r} \rangle}{1 - \langle \mathbf{x_i}, \boldsymbol{\Delta r} \rangle} \right)^+ , \min_{i \in 1..N} \left(\frac{\lambda + \langle \mathbf{x_i}, \mathbf{r} \rangle}{1 + \langle \mathbf{x_i}, \boldsymbol{\Delta r} \rangle} \right)^+ \}$$

where $< ., . >$ denotes the usual inner product, $\boldsymbol{\Delta r}$ is the latest correction to the residual, and:

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

whereas ECON solves: $\xi \equiv \min\{\min_{\boldsymbol{\theta}\in\Theta} \xi_+(\boldsymbol{\theta}), \min_{\boldsymbol{\theta}\in\Theta} \xi_-(\boldsymbol{\theta})\}$ where:

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

where $\boldsymbol{\phi_\theta} = \big(\phi_{\boldsymbol{\theta}}(\mathbf{x_1}), \ldots, \phi_{\boldsymbol{\theta}}(\mathbf{x_N})\big)^\mathsf{T}$. If ϕ is continuous and differentiable, ξ_+ and ξ_- are continuous and differentiable everywhere except at rare unfeasible points, and ξ is also continuous, and looses differentiability only at the frontiers where $\arg\min(\xi_+(\boldsymbol{\theta}), \xi_-(\boldsymbol{\theta}))$ changes, that is precisely at each iteration of ECON (which exactly corresponds to where λ is computed in the LARS). Thus, the main task becomes to minimize, at each step, a relatively smooth function of $\mathbb{R}^p$.

3.2 Solving the minimization problem

To exemplify things, let us assume without loss of generality, that the features are multivariate Gaussians, that is $\phi_{\boldsymbol{\theta}=(\boldsymbol{\mu},\boldsymbol{C})} \equiv e^{(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{C}(\boldsymbol{x}-\boldsymbol{\mu})}$ in which $\boldsymbol{\mu} \in \mathbb{R}^P$, and the covariance matrix $C \in \mathbb{R}^{P\times P}$. We have to find $\boldsymbol{\mu}^*$ and C^* that minimizes $< \phi_{\boldsymbol{\mu},\boldsymbol{C}}, \boldsymbol{r} >$. We do not know of any method to solve this minimization problem exactly, so we have to resort to a numerical minimization. Using a numerical minimization instead of an analytical solution of the minimization problem leads to sub-optimal solutions. This seems a strong weakness, so we would like to discuss this point.

First, we use DiRect [9], a global optimization algorithm which is guaranteed to converge asymptotically towards the global optimal of the function. DiRect keeps on iterating, splitting the domain of optimization in a smart way, so that the optimum will ultimately be found with probability 1. More practically, experiments show that DiRect can find good optima after a limited amount of iterations; furthermore, the quality of the optimum found by DiRect increases with the number of iterations it performs. So, from a practical point of view, we know that the more time we allow DiRect to run, the better will be the solution.

Now, we have to deal with sub-optimal solutions. Actually two cases may happen: DiRect returns a non optimal point which is close, *i.e.* in the same basin of attraction, as the optimum; or, DiRect returns a point which is not even in the basin of attraction of the optimum. In the first case, though we do not do it (yet), post-processing may be done by way of a gradient descent to get to the global optimum. In the second case, this means that ECON makes active a feature instead of an other one which is better. Then, the missed feature may be very well found by DiRect in a subsequent iteration of ECON; actually, the odds for this do increase as a feature is missed because the missed optimum will get more and more attractive (in terms of the energy landscape), so that it will quickly be caught; furthermore, the sub-optimal features that have been included instead of the best feature may well be removed in subsequent ECON iterations, since ECON naturally removes active features that are detected as no longer useful. Finally, with regards to usual NN with a fixed architecture

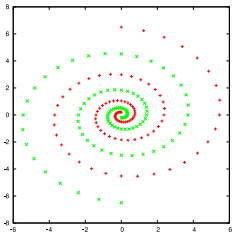
in which only weights are optimized, or growing networks in which the hyper-parameters are set *a priori*, or at random, we think that ECON goes beyond, and can not perform worse that these algorithms.

4 Experimental assessment

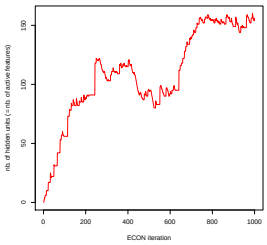
In this section, we use experiments for two things: first, demonstrate ECON on a toy problem, namely the 2 spirals problem; second, demonstrate ECON on larger regression problems. In all these experiments, we use ECON to fit multivariate Gaussians. The free hyper-parameters are the center of each Gaussian and a diagonal covariance matrix. Thus, if data are in dimension P, the number of parameters to be set is $p \equiv 2 \times P$.

4.1 Experiments on the 2 spirals problem

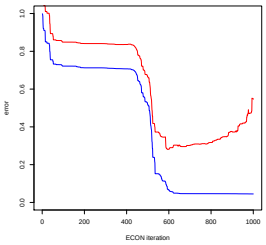
The 2 spirals dataset consists in 194 points located in the plane, along two spirals; each spiral corresponds to a class (see Fig. 1(a)). Without any preprocessing, the dataset is fed to ECON. The structure of the problem is quite complex: in the center, the two classes are closely intertwined, whereas in the outside regions, large patches of the domain are made of a single class. So, to obtain an accurate 1HLP, various bandwidths have to be used, small in the center, larger and larger as we move to outer regions.



(a) The 194 points of the dataset, color indicates the class.



(b) The evolution of the number of active features along ECON iteration.



(c) The learning curve: the evolution of the training error (`E_train` in blue) below the test error (`E_test` in red) along ECON iterations.

Fig. 1. The 2 spirals problem. See text for explanations.

For this experiment, we use 164 data for training (selected at random among the 194 data points), and we run 1500 iterations of ECON[2]. The plot of the evolution of the number of active features (see Fig. 1(b)) clearly exhibits that features are added and removed along ECON iteration. The number of active features saturates a little below 160; this conveys information about the complexity of the data set. We use the extra 30 data to measure a test error. On Fig. 1(c), we see that the test error reaches a minimum at the 594^{th} iteration; at this iteration, only 90 features are active; the leftmost plot in Fig. 2 shows the prediction obtained with this NN: we can see that the result is what we expect. Incidentally, we figure out that using the test error to monitor the stopping of ECON is a very good heuristics.

Then, we investigate further the ability of ECON to capture the complexity of the dataset. We create two other datasets based on the two spirals problem, one with 500 points, the other with 1000 points (half lying on each branch of the spirals). Clearly, those three datasets convey the same information, they have the same structural complexity. So the model being learnt should be approximately the same, and approximately of the same size. The leftmost part of Fig. 2 shows the prediction made by ECON after training of each of the three datasets, using, as previously, approximately 15 % of the data as a test set to determine a best 1HLP. We see two things: as we use more data, the prediction is getting more accurate; and as we get more data, the number of basis in the expansion saturates. A brief word about computational time: on this dataset, ECON performs 300 iterations on the 1000 points dataset in 100 seconds, on a Thinkpad X61s running Debian/sid, programs being compiled with `gcc-4.3`.

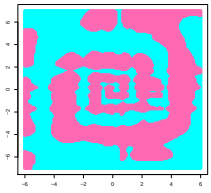
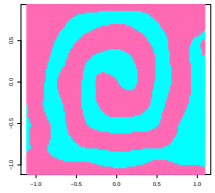
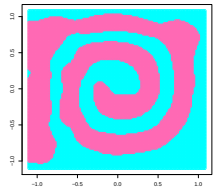
dataset	194 points	500 points	1000 points
prediction on $\mathcal{D}$			
number of hidden units	90	325	319
NMSE (test-set)	0.3	1.510^{-5}	1.510^{-4}

Fig. 2. The 2 spirals problem when the size of the dataset varies. The color indicates the class prediction made by the 1HLP providing the smallest test error. See text for more explanations.

[2] 1500 iterations is much too much iterations to obtain a satisfying accuracy. We run 1500 iterations to illustrate our purpose, and show how ECON works.

4.2 Experiments on regression problems

In this section, we experiment with ECON on larger, real world regression problems. We use the Boston housing dataset to compare our results with those of Guigue who introduced the kernel basis pursuit algorithm [3]. We also use the Abalone and house-price-8L datasets to compare with those published in [10] who compares an SVM, the Relevance Vector Machine, and his own Generalized LASSO algorithm. For Boston housing, 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`.

The dataset of Boston housing is made of 502 data, each made of 13 attributes; as [3], we use 20% of the dataset as a test-set, the other 80 % being used for training. Please note that we do not preprocess the data in any way. On 50 runs, ECON clearly outperforms their results when they use a single kernel, with an average accuracy on the test-set of 10.91, standard deviation of 3.56 (the distribution of the accuracy is actually very skewed, with more than half runs in the range 6-10). On average, 232 hidden units are used (ranges from 100 to 400), that is about 25% less than [3].

Regarding Abalone and house-price-8L, in both datasets the data are made of 8 attributes; Abalone holds 4177 data, while house-price-8L has 22784 data; so, these are much larger datasets than the toy spiral example, and Boston. 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 [10], and the number of hidden units is much smaller. The results are provided in table 1.

Table 1. We compare the performance of ECON with those published in [10], 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, or the number of hidden units for ECON) 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.

DATASET	SVM	RVM	[10]	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 Conclusion, discussion, and future work

We have presented ECON, a new algorithm which aims at building 1 hidden layer perceptrons for regression problems. The algorithm is an extension of the LARS

algorithm. With regards to this algorithm, the (important) novelty is that ECON optimizes the hyper-parameters of the features as they are added to the hidden layer of the neural net. This optimization is made by a global optimizer, since we have no analytical expression of the solution of this minimization problem. Based on l_1 regularization, ECON builds very compact networks. The natural ability to remove useless units makes ECON attractive for sequential learning, even for non stationary data, an issue that we have not yet seriously studied. Experiments show that ECON is able to achieve state of the art performance on regression problems. From the user point of view, one appealing feature of ECON is that it has basically no parameter to tune: just feed it with data, and ECON learns. Regarding its computational cost, compared to the traditional approach to find the "best" parametrization by testing different parametrizations and keeping the best, ECON is very competitive. A major feature of the LARS, inherited by ECON, is that it provides a whole set of networks, not a single one; actually, we get an infinite number of networks, one for each value of the constant of regularization. So, the user can select his/her best network *a posteriori*. Though we have not yet explored this point, the information provided in the build networks may probably be exploited to get a better understanding of the dataset since ECON acts as a feature extractor. On the technical side, we wish to study other global optimizers; we currently use our own implementation of DiRect, but we would like to compare it with other available implementations of DiRect, as well as with other optimizers. More fundamentally, we also foresee the possibility to develop the same kind of approach to synthesize multiple hidden layer perceptrons.

References

1. Ng, A.: Feature selection, l_1 vs. l_2 regularization, and rotational invariance. In: Proc. of the 21$^{\text{st}}$ ICML. (2004)
2. Efron, B., Hastie, T., Johnstone, I., Tibshirani, R.: Least-angle regression. Annals of statistics **32**(2) (2004) 407–499
3. Guigue, V., Rakatomamonjy, A., Canu, S.: Kernel basis pursuit. In: Proc. ECML. Volume 3720., Springer, LNAI (2005) 146–157
4. Fahlman, S.E., Lebière, C.: The cascade-correlation learning architecture. In: NIPS. Volume 2. (1990) 524–532
5. Platt, J.: A resource-allocation network for function interpolation. Neural Computation **3**(2) (1991) 213–225
6. Huang, G.B., Saratchandran, P., Sundararajan, N.: A generalized growing and pruning RBF (GGAP-RBF) neural netowrk for function approximation. IEEE Trans. on NN **16**(1) (2005) 57–67
7. Cauwenberghs, G., Poggio, T.: Incremental and decremental support vector machine learning. In: Proc. NIPS. (2000) 409–415
8. Vijayakumar, S., D'Souza, A., Schaal, S.: Incremental online learning in high dimensions. Neural Computation **17**(12) (2005) 2602–2634
9. Jones, D., Perttunen, C., Stuckman, B.: Lipschitzian optimization without the lipschitz constant. J'al of Opt. Theory and Applications **79**(1) (1993) 157–181
10. Roth, V.: Generalizd lasso. IEEE Trans. on NN **15**(1) (2004) 16–28