A Telescopic Binary Learning Machine for Training Neural Networks

Mauro Brunato, *Member, IEEE*, and Roberto Battiti, *Fellow, IEEE*

***Abstract*— This paper proposes a new algorithm based on multiscale stochastic local search with binary representation for training neural networks [binary learning machine (BLM)]. We study the effects of neighborhood evaluation strategies, the effect of the number of bits per weight and that of the maximum weight range used for mapping binary strings to real values. Following this preliminary investigation, we propose a telescopic multiscale version of local search, where the number of bits is increased in an adaptive manner, leading to a faster search and to local minima of better quality. An analysis related to adapting the number of bits in a dynamic way is presented. The control on the number of bits, which happens in a natural manner in the proposed method, is effective to increase the generalization performance. The learning dynamics are discussed and validated on a highly nonlinear artificial problem and on real-world tasks in many application domains; BLM is finally applied to a problem requiring either feedforward or recurrent architectures for feedback control.**

***Index Terms*— Incremental local search, neural networks, stochastic local search (SLS).**

I. Introduction

MACHINE learning (ML) and optimization share a long common history. ML implies optimizing the performance of the trained system measured on the examples, with possible early stopping or additional regularizing terms to avoid overtraining and increase generalization.

Starting from backpropagation (BP), most techniques for training neural networks use continuous optimization schemes based on partial derivatives, like gradient descent and variations thereof, while support vector machines use quadratic optimization. Recent proposals to simplify the optimization task by creating a randomized first layer and limiting optimization to the last-layer weights are the extreme learning machine (ELM) [1] and reservoir computing [2].

Different methods consider combinatorial optimization (CO) techniques without derivatives, such as simulated annealing (SA) and some flavors of evolutionary algorithms [3] and genetic algorithms (GAs) (see [4] for a comparative analysis). In most CO techniques, weights are considered as binary strings, and the operators acting during optimization change bits, either individually (like with mutation in GA) or more globally (like with crossover operators in GA) [5], [6]. Methods for the optimization of functions of real variables with no derivatives are an additional option, for example, versions of SA [7]. Intelligent schemes based on adaptive diversification strategies by prohibiting selected moves in the neighborhood are considered in [8] and [9]. Such methods were proposed for neural network training to overcome the problem of early convergence of gradient-based schemes to local minima (see, for example, proposals of tabu search [10], [11], particle swarm optimization [12], and continuous versions of GAs [13]).

A strong motivation for considering techniques with no derivatives is when derivatives are not available, for example, in threshold networks [14], because the employed transfer functions are discontinuous. The binary learning machine (BLM) algorithm has been introduced in this context [15].

This paper considers neural networks with smooth input–output transfer functions and proposes a telescopic BLM that combines stochastic local search (SLS) optimization with the discrete representation of network parameters (weights). The technique is suitable both for feedforward and recurrent networks, and can also be applied to the control of dynamic systems with feedback.

This paper is organized as follows. In Section II, we introduce the SLS techniques that will be used for the optimization of network parameters. In Section III, we describe the specific building blocks and algorithmic choices that enable the BLM algorithm. In Section IV, we use a highly nonlinear benchmark task to develop insight and drive the choice of specific algorithms and parameters. In Section V, we test the technique on a set of classic real-world benchmark problems and compare it with the state-of-the-art derivative-based and derivative-free methods. In Section VI, we apply the network to a well-known control problem and test the suitability of BLM for training recurrent networks.

II. Local Search on Highly Structured Fitness Surfaces

Local search is a basic building block of most heuristic search methods for CO. It consists of starting from an initial tentative solution and trying to improve it through repeated small changes. At each repetition, the current solution is perturbed, and the function to be optimized is tested. The change is kept if the new solution is better, otherwise another change is tried. The function $f(X)$ to be optimized

Manuscript received April 30, 2015; accepted February 26, 2016. This work was supported by the Università degli Studi di Trento. The work of R. Battiti was supported by the Russian Science Foundation through the Project entitled Global Optimization, Supercomputing Computations, and Applications under Grant 15-11-30022.

M. Brunato is with the Dipartimento di Ingegneria e Scienza dell'Informazione, University of Trento, 38123 Trento, Italy (e-mail: mauro.brunato@unitn.it).

R. Battiti is with the Dipartimento di Ingegneria e Scienza dell'Informazione, University of Trento, 38123 Trento, Italy, and also with the Lobachevsky State University of Nizhny Novgorod, 603950 Nizhny Novgorod, Russia (e-mail: roberto.battiti@unitn.it).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2016.2537300

is called fitness function, goodness function, or objective function.

In a way, gradient-based techniques, such as BP, also consider the effect of small local changes. In BP, the effect of the tentative change is approximated by the local linear model: the scalar product between gradient and displacement. When no analytic derivatives are available, the derivatives can be approximated by finite differences. One is, therefore, dealing with techniques based on gradual changes evaluated through their effect on the error function, but considering individual bits leads to a radical simplification, and to a natural multiscale structure because of the binary representation (change of the ith bit leads to a change proportional to 2^i).

Let us define the notations. $\mathcal{X}$ is the search space, $f(X)$ is the function to be optimized, and $X^{(t)}$ is the current solution at iteration (time) t. $N(X^{(t)})$ is the neighborhood of point $X^{(t)}$, obtained by applying a set of basic moves $\mu_0, \ldots, \mu_M$ to the current configuration

$$N(X^{(t)}) = \{X \in \mathcal{X} : X = \mu_i(X^{(t)}), i = 0, \ldots, M\}.$$

If the search space is given by binary strings with a given length L, then $\mathcal{X} = \{0, 1\}^L$, and the moves can be those changing (or complementing or flipping) the individual bits; therefore, M is equal to the string length L.

Local search starts from an admissible configuration $X^{(0)}$ and builds a trajectory $X^{(0)}, \ldots, X^{(t+1)}$. The successor of the current point is a point in the neighborhood with a lower value of the function f to be minimized

$$Y \leftarrow \text{Improving-Neighbor}(N(X^{(t)}))$$
$$X^{(t+1)} = \begin{cases} Y, & \text{if } f(Y) < f(X^{(t)}) \\ X^{(t)}, & \text{otherwise (search stops).} \end{cases}$$

Function Improving-Neighbor returns an improving element in the neighborhood. In a simple case, this is the element with the lowest f value, but other possibilities exist, for example, the first-improving neighbor encountered. If no neighbor has a better f value, i.e., if the configuration is a local minimizer, the search stops.

Local search is surprisingly effective, because most CO problems have a very rich internal structure relating the configuration X and the f value. A neighborhood is suitable for local search if it reflects the problem structure. Stochastic elements can be immediately added to local search to make it more robust, for example, a random sampling of the neighborhood can be adopted instead of a trivial exhaustive coverage.

In many relevant problems, local minima tend to be clustered; furthermore, good local minima tend to be closer to other good minima. Let us define, as attraction basin associated with a local optimum, the set of points X that is mapped to the given local optimum by the local search trajectory. One may think about a smooth f surface in a continuous environment, with basins of attraction, which tend to have a nested structure, where smaller valleys are nested within larger ones. This motivates more complex methods, such as variable neighborhood search and iterated local search [16]. This structural property is also called big valley property (or massif central).

In SA, for instance, the big valley property motivates gradual reduction (cooling schedule) [17] of the temperature parameter T controlling move acceptance, so that the initial sampling is regulated by the overall large-scale structure of the problem (fine details are initially ironed out by the high T), while lower values for T at the end make the search more sensitive to fine-scale details.

Multiscale methods that jointly solve related versions of the same problems at different scales have been proposed in different areas, e.g., image analysis [18], modeling and computation, and neural networks for optimization [19]. The application in [19] is for multiscale optimization (inexact graph matching). A cubic neural network energy function is considered, a smaller coarser version of the problem is derived, and one alternates between the relaxation steps for the fine-scale and coarse-scale problems.

In this paper, we consider two simple options: no diversification (where the search stops upon reaching a local optimum), and repeated local search, where the search restarts from a random configuration whenever a local optimum is attained.

In the following, the term run will refer to a sequence of moves that ends to a local minimum of the search space, while a search can refer to a sequence of runs and restarts.

In the following, we illustrate the main building blocks leading to a much faster and more effective multiscale implementation of local search for neural networks.

III. Binary Learning Machine Algorithm

A brute-force implementation of local search consists of evaluating all possible changes of the individual bits representing the weights. For each bit change, all training examples are evaluated to calculate the difference in the training error. This implementation is out of question for networks with more than a few neurons and weights, leading to enormous CPU times. This can explain why basic local search has not been considered as a competitive approach for training up to now.

In this Section, we design an algorithm, called telescopic BLM, which uses a smart realization of SLS, coupled with an efficient network representation. The term BLM underlines the fact that it is based on the binary representation of weights and changes acting on the individual bits.

Telescopic BLM is based on the following:

1) gray coding to map from binary strings to weights;
2) incremental neighborhood evaluation (also called delta evaluation);
3) smart sampling of the neighborhood (stochastic first improving and best improving);
4) multiscale (telescopic) search, gradually increasing the number of bits in a dynamic and adaptive manner.

In this section, we illustrate the many choices to obtaining a representation of the neural network that can be efficiently optimized by an SLS algorithm.

A. Weight Representation

The appropriate choice of representation is critical to ensuring that the changes of individual bits lead to an effective sampling of the neighborhood.

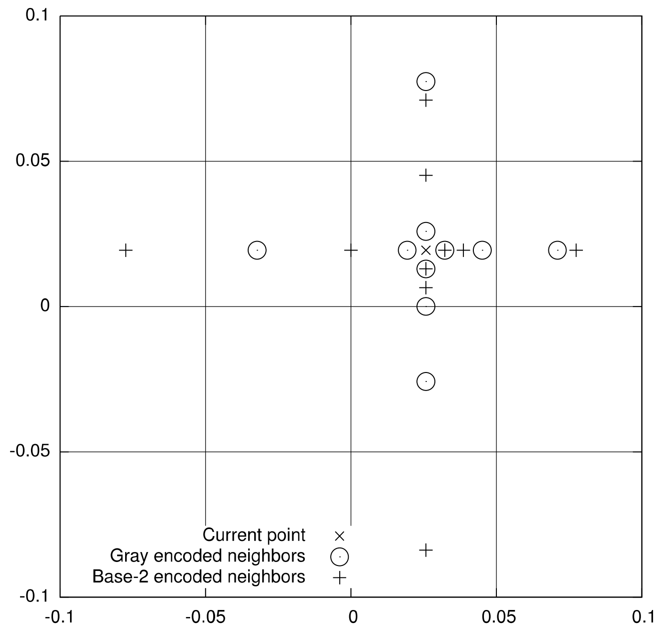

Fig. 1. Neighbors of the current point with Gray coding versus standard base-2 coding.

1) Number of Bits and Discretization: We represent weights and biases as the integral multiples of a discretization value $\epsilon > 0$, with an n-bit two's-complement integer as multiplier. Options $n = 2, 4, 8, 12, 16, 24$ have been investigated. In all tests, a maximum representable weight value $w_{\max}$ is given (this limitation in a weight size acts as an intrinsic regularization mechanism to prevent overtraining), and ϵ is set, so that the range of representable weights includes the interval $[-w_{\max}, w_{\max}]$. As the two's-complement representation is asymmetric, with n bits actually representing the set $\{-2^{n-1}, \dots, 2^{n-1} - 1\}$, we set

$$\epsilon = \frac{w_{\max}}{2^{n-1} - 1} \tag{1}$$

so that the maximum value $w_{\max}$ can be represented; the minimum representable weight will actually be $-w_{\max} - \epsilon$. Since we start from $n = 2$ b, one can always represent at least one positive and two negative weight values.

2) Gray Coding: In order to improve the correspondence between the bit-flip local moves of the BLM algorithm and the natural topology of the network's weight space, we use Gray encoding of the n-bit integer multiplier. Gray encoding has the property that, starting from code n, the nearby codes for $n-1$ and $n+1$ are obtained by changing a single bit (i.e., they have a Hamming distance of one with respect to the code of n). This choice allows the generic value $h\epsilon$, h being an n-bit integer, to reach its neighboring values $(h+1)\epsilon$ and $(h-1)\epsilon$ by a single move, while in the conventional binary coding, one of the two neighbors could be removed by as much as n moves, namely when moving from 0 to -1 (represented as $00\cdots0$ and $11\cdots1$ in two's-complement standard binary). Given a weight w, by changing one of the n bits in the encoding (and by repeating the operation for all possible bits), one obtains n weights in the neighborhood. As shown in Fig. 1, for the cited property of the Gray code, the neighborhood always contains the nearest weights on the discretized grid, plus a cloud of points at growing distances in weight space.

B. Memory-Based Incremental Neighborhood Evaluation

Let N_{layers} be the number of network layers (hidden and outputs); the input layer is layer 0, while layer N_{layers} is the output. For $l = 0, \dots, N_{\text{layers}}$, let n_l be the number of neurons in layer l. For $j = 1, \dots, n_l$, let o^l_j be the output of neuron j in layer l. Thus, o^0_j is the value of the jth input. Let $f^l(x)$ be the transfer function of neurons at layer l, and w^l_{ij} be the weight connecting neuron i at layer $l-1$ to neuron j at layer l. Starting from the input values, the feedforward computation consists of iterating the following operation for $l = 1, \dots, N_{\text{layers}}$:

$$o^l_j = f^l\left(s^l_j\right), \quad \text{where} \quad s^l_j = b^l_j + \sum_{i=1}^{n_{l-1}} w^l_{ij} o^{l-1}_i. \tag{2}$$

The simple bit-flip neighborhood scheme chosen for this investigation requires a large number of steps, each consisting of the modification of a single weight. A very significant speedup can be obtained by storing the s^l_j values for all neurons and all samples. In order to obtain the network output when weight w^L_{IJ} is replaced by a new value W, the following algorithm provides a faster incremental evaluation.

1) Recompute the output of neuron J in layer L
$$o'^L_J = f^L\left(s^L_J + \left(W - w^L_{IJ}\right) o^{L-1}_I\right).$$
If $L = N_{\text{layers}}$, the Jth network output is o'^L_J, the others are unchanged. Stop.
2) Let the output variation be
$$\Delta o^L_J = o'^L_J - o^L_J.$$
3) Recompute all contributions of output o^L_J to the neurons of the next layer, for $k = 1, \dots, n_{L+1}$
$$s'^{L+1}_k = s^{L+1}_k + w^{L+1}_{Jk} \Delta o^L_J$$
$$o'^{L+1}_k = f^{L+1}\left(s'^{L+1}_k\right).$$
4) If there are other layers, apply (2) for $l = L+2, \dots, N_{\text{layers}}$.

Obvious modifications apply for bias values.

Consider the case shown in Fig. 2, where $N_{\text{layers}} = 2$ (single hidden layer). Upon the modification of an input weight, the update procedure requires $O(1)$ operations per sample to update the output of the only affected hidden neuron, and $O(n_2)$ operations to recompute the contributions of that hidden neuron on all outputs. Upon the modification of an output weight, the whole incremental procedure requires $O(1)$ calculations for recomputing the affected output neuron. On average, since the network contains $O(n_0 n_1)$ input weights and $O(n_1 n_2)$ output weights, an incremental computation requires $O(n_0 n_2 / (n_0 + n_2))$ operations.

For comparison, the number of operations required by the complete feedforward procedure is roughly proportional to the number of weights in the network, i.e., $O(n_1(n_0 + n_2))$. An incremental search step on the largest benchmark considered is faster by about two orders of magnitude with respect to the complete evaluation.

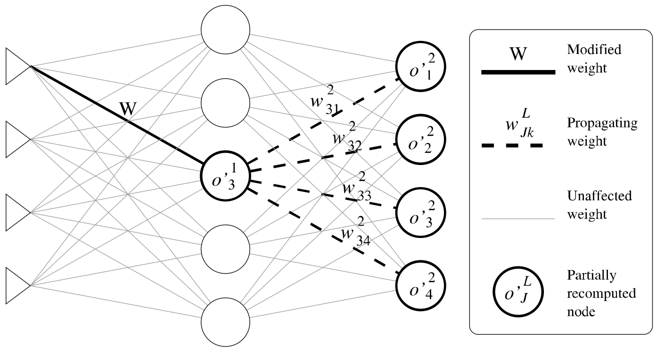

Fig. 2. Affected neurons in an incremental evaluation step with a single changed weight in the input layer.

C. Neighborhood Exploration

An important parameter for the implementation of a local search heuristic is the fraction of neighborhood that the algorithm explores before moving, and two extreme cases are as follows.

1) *Best Move:* All feasible moves are checked, and the one leading to the best improvement in the target function is selected. Ties may be broken randomly or by considering the secondary objectives.
2) *First-Improving Move:* Feasible moves are explored in the random order, and the first move that leads to an improvement of the target objective is selected.

Preliminary tests of the BLM algorithm suggest that the best move strategy causes a serious slowdown of each iteration and, while having a steeper initial improvement, it can lead to more overtraining. Therefore, in the remainder of this paper, we shall only consider the first-move strategy.

Since the search starts from a random configuration, at the beginning, most moves are improving, and the algorithm only needs to scan a small subset before finding one. As the search proceeds, the average number of neighbors that must be evaluated increases. The fraction, however, does not become very large until the local minimum is close, and remains in the 10% range for most of the run.

On the benchmark problems considered, the combination of incremental evaluation and first-improving strategy leads to an approximate speedup in the order of 100 times with respect to the unoptimized baseline implementation of the algorithm.

D. Multiscale Network Representation (Telescopic BLM)

Given the potentially large number of weights in a fully connected feedforward neural network, its discrete n-bit representation can be large, leading to slow improvements. The telescopic variant of BLM starts with a coarser representation, where only the most significant n' bits per weight (e.g., $n' = 2$) are allowed to change. This allows for a quick, large-scale exploration of the search space. Periodically, the number n' of flippable bits is increased until the full-fledged n-bit search takes place.

As the search proceeds, therefore, the neighborhood structure of the problem becomes richer and richer; moreover, the most significant bits are always allowed to flip. Therefore, the search algorithm is never restrained in small regions as the search proceeds (as is often the case with BP and learning-rate-decay mechanisms).

The decision to increase the number n' of unlocked bits can be linked in an adaptive manner to the current search results. The simplest criterion is to increase n' whenever a local minimum of the loss function is reached for the current neighborhood structure. In this case, an additional bit is unfrozen and available for tests and possible changes.

E. Telescopic Threshold Adjustment

Instead of waiting until a local minimum is found, one can increase the number of bits in the weight representation of the BLM algorithm whenever the number of improving neighbors falls below a certain threshold (e.g., expressed as a fraction of the total number of possible moves in the neighborhood).

In a first-improvement scenario, where we do not test all possible moves, we cannot directly determine the desired number. Therefore, we need to efficiently estimate it.

As shown in the Appendix,[1] given N possible moves of which only $k < N$ are improving, the expected number of moves to be tested before finding an improving one is given by the following recurrence equation:

$$E_{k,N} = \begin{cases} 0, & \text{if } N = k \\ \alpha_{k,N} + \dfrac{N+k}{N} E_{k,N-1}, & \text{otherwise} \end{cases} \quad (3)$$

where $\alpha_{k,N}$ is determined by another recurrence

$$\alpha_{k,N} = \begin{cases} 0, & \text{if } N = k \\ \dfrac{N-k}{N}\left(\dfrac{k}{N-1} + \alpha_{k,N-1}\right), & \text{otherwise.} \end{cases} \quad (4)$$

To estimate the number of improving moves k, we proceed at estimating the current value of E_{kn} as follows. Let c_i be the number of moves that have been sequentially tested at step i of the local search procedure before finding an improving one. If we assume that the number of improving moves does not change dramatically between iterations, then a moving average of the c_i values is an estimator of $E_{k_i,N}$, where k_i is the (unknown) total number of improving moves at time i

$$E_{k_i,N} \approx \mu_i \quad (5)$$

where $\mu_0 = 0$, $\mu_i = \eta\mu_{i-1} + (1-\eta)c_i$ is an exponential moving average with decay factor η.

Suppose that we want to increase the number of bits in the search space whenever the number of improving moves k_i falls below a given threshold ϕN (given as a fraction of the total number of moves N, with $0 \leq \phi \leq 1$); since $E_{k,N}$ is a decreasing function of k, then the condition $k_i < \phi N$ is equivalent to

$$E_{k_i,N} \geq E_{\lfloor \phi N \rfloor, N}$$

finally, by considering the above approximation (5), the telescopic criterion becomes

$$\mu_i \geq E_{\lfloor \phi N \rfloor, N}.$$

[1]See http://intelligent-optimization.org/papers/tnnls2016a.pdf.

In order to use this criterion, we need to recompute the threshold $E_{\lfloor \phi N \rfloor, N}$ by using (3) and (4) whenever the number N of moves changes (i.e., when the number of bits n' in the weights representation is increased), and maintain a mobile average of failed attempts per local move with a suitable decay factor η.

Observe that the initial value for the moving average, $\mu_0 = 0$, corresponds to the optimistic assumption that all moves are improving. This choice mitigates the impact of high c_i values happening by chance at the beginning of the search. The moving average is reset to zero every time the number of bits is increased.

F. Weight Initialization

The single-bit-flip move structure presented above is not directly compatible with all-zero weight initialization. As soon as the network has at least one hidden layer, in fact, at least two nonzero weights, one in the hidden layer and one in the output layer, are necessary to have nonconstant output.

1) Full Random Initialization: The simplest initialization strategy is to initialize all weights within their variation range, using a uniform distribution. This is equivalent to initializing every bit of the binary representation to 0 or 1 with equal probability.

2) Bounded Random Initialization: Starting from small random weight values is often advisable. Small weights lead to better generalization, and this is particularly important at the start of training, where weights should not be strongly polarized. Therefore, setting a small initialization range $[-w_{\text{init}}, w_{\text{init}}]$ is a valid choice. Since weights are discretized, it is important that the maximum initial weight is at least as large as the discretization step: $w_{\text{init}} \geq \epsilon$. Initial weights are then rounded to the closest discretized value.

3) Initialization in Telescopic BLM: Starting from small weights before running telescopic BLM requires a few considerations. Telescopic BLM initially operates on the most significant part of the weight representation, leaving the least significant bits untouched, thereby preventing any weight to return to zero until the final phase when it operates on all bits.

Fig. 3 shows the case where two $n = 5$-bit weights are initialized with small values; for simplicity, the discretization step (1) has been set to $\epsilon = 1$. If the initial step only involves the $n' = 2$ most significant bits, then the grid of actually accessible weight values is offset with respect to the origin, and weights are not allowed to vanish or, in some cases, even become smaller until later phases.

In some cases, the small offset acts as a beneficial noise source, forcing small random contributions and encouraging the use of spontaneous features, in the spirit of reservoir learning. Otherwise, it is possible to alleviate this problem by choosing random initial weights that are multiple of $\eta = 2^{n-n'}\epsilon$ (clearly, $w_{\text{init}} > \eta$, therefore initial weights are not going to be as small), so that all least significant bits are forced to zero.

In this paper, for lack of space, we concentrate mostly on the latest two options.

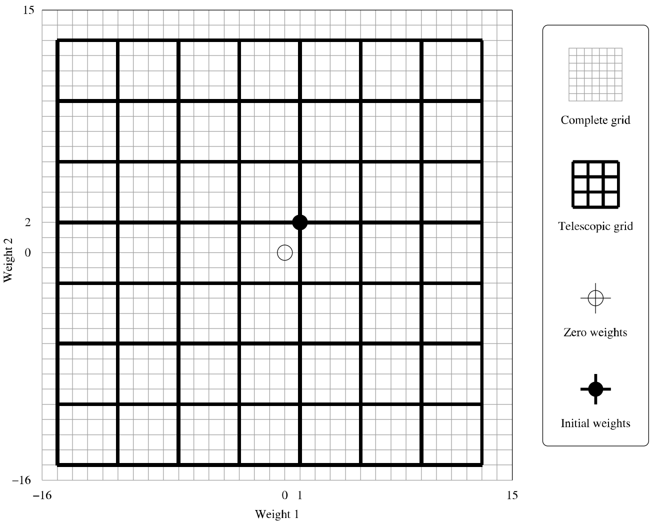

Fig. 3. Two-weight network, $n = 5$ b/weight, initialized with small values (black dot). If only the $n' = 2$ most significant bits are changed, the accessible weight set (thick grid) does not include $(0, 0)$.

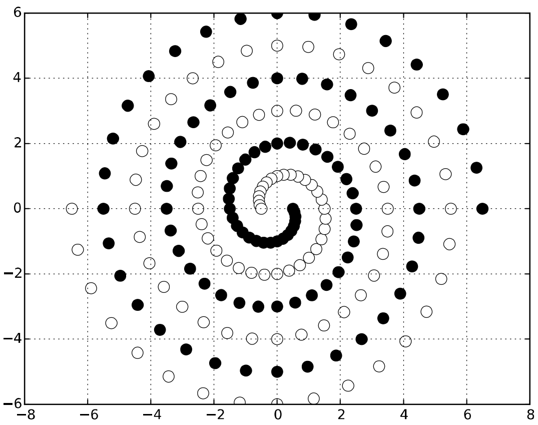

Fig. 4. Two-spirals problem.

IV. Tests on the Two-Spirals Problem

The purpose of the experiments in this section is to test the effect of the BLM algorithm on a 2-D benchmark task, so that the results can be visualized to develop some intuition before the second series of quantitative tests in Sections V and VI.

A highly nonlinear classification benchmark that is particularly difficult for gradient descent optimization is the two-spirals problem developed in [20]. The data set is taken from the CMU neural networks benchmarks. It consists of 193 training instances and 193 test instances located on a 2-D surface. They belong to one of two spirals, as shown in Fig. 4.

It is possible to solve this problem with one or two hidden layers, but architectures with two hidden layers need less connections and can learn faster. The architecture we consider here is $2 - 20 - 20 - 1$ with bias. The hidden units consist of symmetric sigmoids (hidden layers) and the usual $0 - 1$ sigmoid for the output unit. All points are used for training, and the final mapping is shown with a contour plot for a visual representation of its generalization abilities.

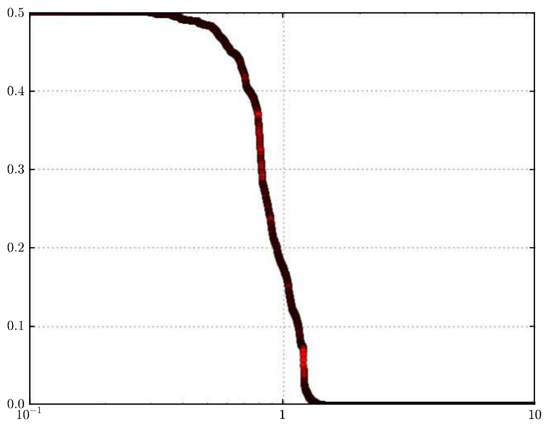

Fig. 5. Two-spirals problem with OSS: rms error as a function of CPU time. Log time axis.

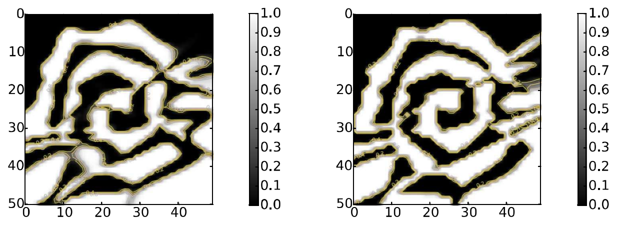

Fig. 6. Two-spirals problem with OSS. Intermediate result (2000 iterations, rms 0.02) and final map obtained.

Simple BP tends to remain trapped in local minima or lead to excessive CPU times. The one-step secant (OSS) technique [21], [22] based on the approximations of the second derivatives from gradients is much more effective and is among the state-of-the-art methods for training this kind of highly nonlinear systems. The rms error of OSS as a function of CPU time is shown in Fig. 5. It can be shown that a large number of OSS iterations (about 2000) are spent in the initial phase without significantly improving the rms error; then, the proper path in weight space is identified and the error rapidly goes to zero.

The final results obtained with OSS are shown in Fig. 6. Some signs of overtraining are clearly visible as stripes in the map. Early stopping does not help. Let us focus on the final map for a fair comparison with those obtained by BLM at the local minimum.

Results for telescopic BLM are in Fig. 7 The architecture is the same as that used for OSS. The specific parameters of BLM are: an initial weight range of 0.001, initial number of bits 2, 12 b, and a weight range of 6.

It can be observed that the percentage of the neighborhood considered at each step is below 1% for most steps, with rare jumps up to 5%–10% (these rare jumps are visible in the plot which shows all 726 702 steps executed in 1000 s). The fraction of the neighborhood explored by the first improve tends to grow only in the final part of search for each number of bits, when the local minimum is close and more and more neighbors need to be examined before finding an improving move. After the local minimum is reached, a new bit is added to each weight, so that the fraction drops again to very

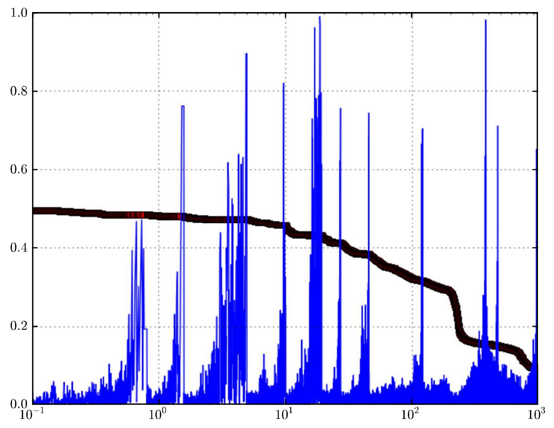

Fig. 7. Two-spirals problem with telescopic BLM, 20-20 architecture, training rms error as a function of CPU time. One curve (top—red) shows the rms error evolution, and the second curve (bottom—blue) shows the percentage of the neighborhood considered at each step.

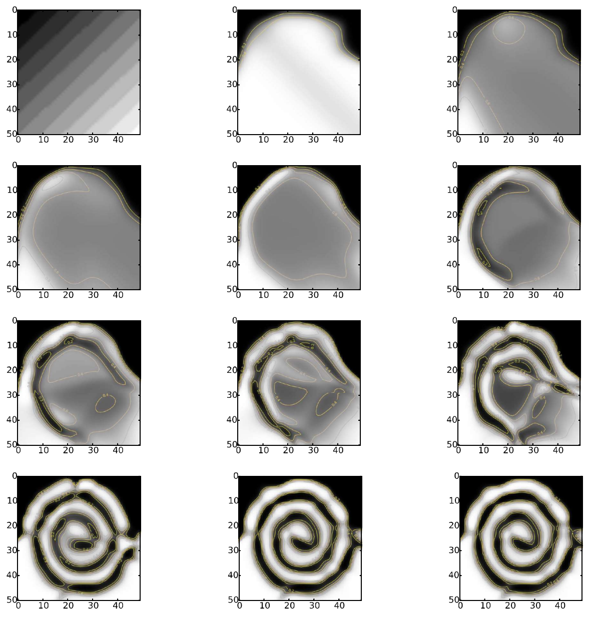

Fig. 8. Two-spirals problem with telescopic BLM, 20-20 architecture. The capability of the map to represent the spiral increases when the number of bits per weight increases (from 1, top left, to 12, bottom right).

low values. Each macroscopic spike, therefore, corresponds to the moment when a local minimum is reached and the number of bits is increased.

The various local minima correspond to the mappings of Fig. 8. The telescopic multiscale algorithm works by first modeling the overall large-scale circular structure and the south positive and north negative area. Then, finer and finer internal details are fixed. The spiral maps are already well formed with 10 b, while the two additional bits add some fine-tuning.

The final mapping obtained (Fig. 9) is very smooth, without the signs of overtraining that are evident in the mapping for OSS. This result is surprising given that it is obtained with discretized weights represented with 12 b.

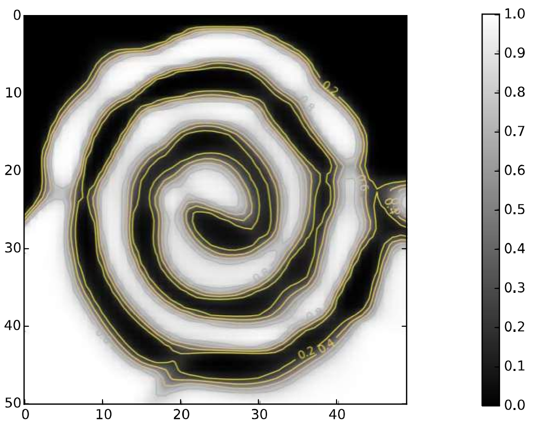

Fig. 9. Two-spirals problem with telescopic BLM, 20-20 architecture, final mapping obtained.

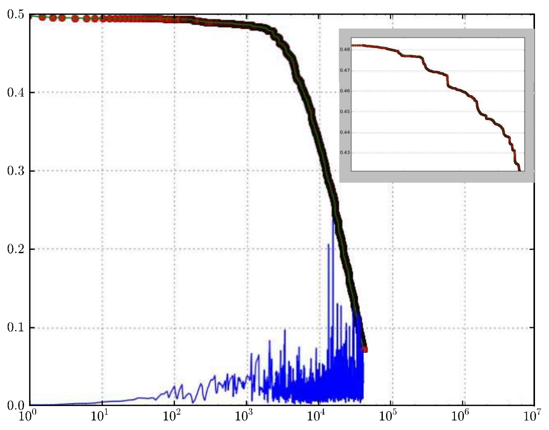

Fig. 10. Two-spirals problem with sparsity enforcing, evolution of training. Circles: rms error as a function of CPU time. Line: fraction of explored neighborhood. Inset: zoomed-in view of the initial improvement; the steplike behavior is due to new neurons becoming active (new nonzero weights).

Because BLM acts by local search changing individual bits, it is simple to consider the variations of the algorithms. For example, sparsity can be encouraged, by aiming at networks in which a large number of weights are fixed to zero (and, therefore, can be eliminated from the final realization).

To encourage sparsity, in the following test, only 50% of the weights are initialized with values different from zero, and the analysis of potential moves in the neighborhood at each step is changed as follows. First, all bits in nonzero weights are tested for a possible change. Only of no such bit is identified, and the bits in zero weights are considered. The philosophy is “Let the current neurons fully express their potential before adding additional neurons (with nonzero weights).”

The evolution of this sparsity-enforcing training is shown in Fig. 10. It can be observed that the final distribution of weights is indeed sparse [Fig. 11 (left)], with a peak at zero and two peaks at the smallest and largest possible values. The final mapping is smoother than the one obtained without enforcing sparsity [Fig. 11 (right)].

While we have no space in this paper to investigate the sparsity issues for all benchmarks, the above preliminary results are encouraging and we plan to extend the analysis in the future work.

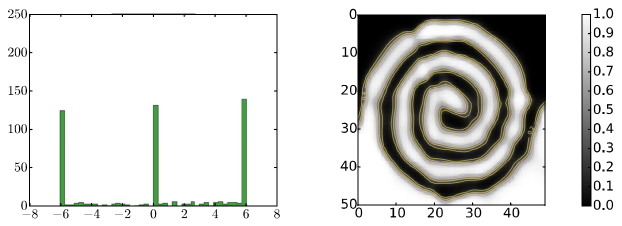

Fig. 11. Two-spirals problem with sparsity enforcing, final distribution of weights (left) and final mapping obtained (right).

TABLE I

SPECIFICATION OF THE SIX BENCHMARK SETS USED IN COMPARISONS

Dataset name	Training samples	Test samples	Attributes	Outputs	
house16H	15948	6836	16	1	
yeast	1038	446	8	10	(classif.)
abalone	2924	1253	$7+1$	1	
comp_act	5406	2786	12	1	
housing	333	173	13	1	
pole_tlc	9900	5100	48	1	

V. FEEDFORWARD NETWORK BENCHMARKS

In the following, we collect the experimental results on the widely used benchmark real-life regression and classification problems listed in Table I. The data sets have been chosen among the larger ones presented in [23].[2]

The house16H data set contains housing information that relates demographics and housing market state of a region with the median price of houses.

The yeast data set consists of a classification problem. The task is to determine the localization site of protein in gram-negative bacteria and eukaryotic cells [24]. Output is coded with a unary representation (1 for the correct class and 0 for the wrong classes). In addition to the traditional rms error error, also the cross-entropy (CE) error is considered in this classification case.

The abalone data set contains data on specimens of the abalone mollusk; seven columns report physical measures, one column is nominal (male/female/infant), and for our purposes, it has been transformed into three ± 1-valued inputs, so that a total of ten inputs were used. The output column is the age of the specimen (number of accretion rings in the shell).

The comp_act data set is a collection of computer activity measures that require to predict one continuous value (fraction of time spent by a CPU in the user mode) given 12 continuous attributes that measure various OS operations.

The housing data set relates 13 features of different areas of a city (Boston) to the median house price in that area.

The pole_tlc data set is a widespread data set referring to a telecommunications industry problem [25] with 48 continuous variables and one continuous output.

All data set inputs have been normalized by an affine mapping of the training set inputs onto the $[-1, 1]$ range and

[2]Also available at http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html.

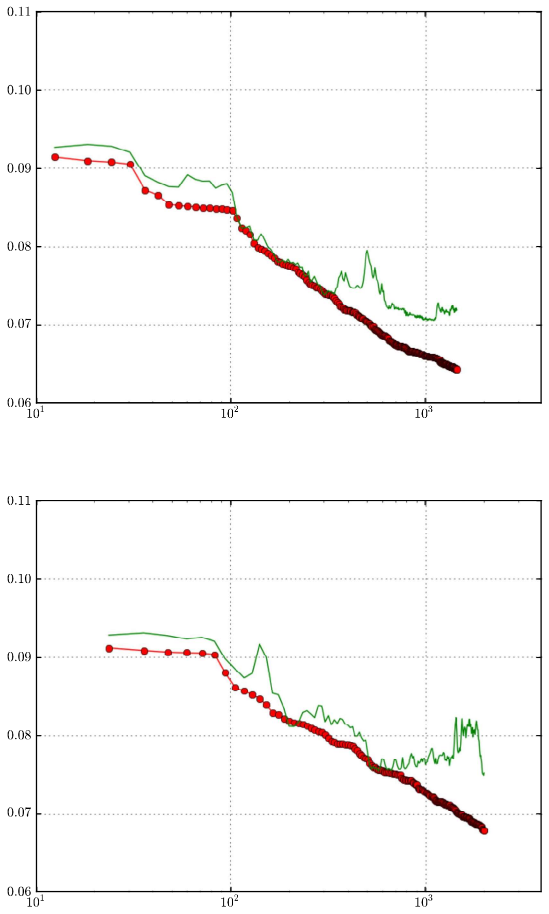

Fig. 12. OSS for `house16H`. Training and validation data as a function of CPU time. 100 hidden units (top) and 200 hidden (bottom). Average rms error on training set (circles) and on test set (simple line).

by an affine mapping of the outputs onto the [0, 1] range. The affine normalization coefficients obtained on the training set have also been applied to the test set values. Two thirds of the examples are used for training, and the remaining are used for validation. Given that we did not repeat the experiments to determine free parameters, a separate test set was not deemed necessary (the terms validation and the test are used with the same meaning in the following description).

A. `House` Benchmark

Before analyzing the distribution of results over multiple runs, it is of interest to observe two individual runs of OSS and BLM for 100 and 200 hidden units. The transfer functions are symmetric sigmoid (hidden units) and linear (output units). Weights are initialized randomly in the range (−0.01,0.01). For BLM, the number of bits is 12, and the weight range is 8. BLM regularization weight is 1. The runs last 4000 s.

In Fig. 12, one observes a qualitative difference. While the error on the training set always decreases (as expected), the generalization error on the validation set has a noisy and irregular behavior for OSS, with the clear signs of overtraining at the end of the run. The evolution of the validation error for BLM is much smoother, and no sign of overtraining is present. We conjecture that this behavior is caused by the more

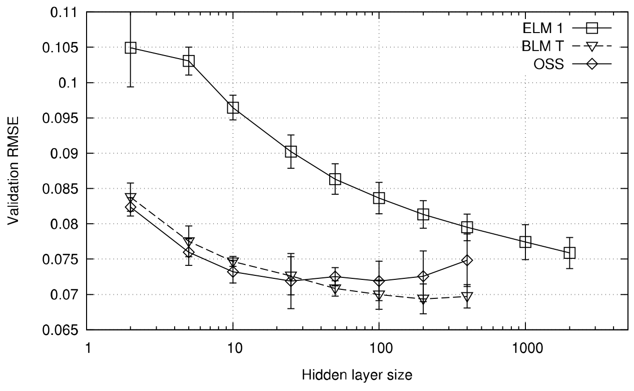

Fig. 13. `house16H` benchmark. Validation results as a function of hidden layer size.

stochastic (less greedy) choice of each weight change in BLM with respect to OSS. The best validation results are generally better for BLM (with the improvements of ~1%–3%).

The results of five runs for each configuration, with different numbers of hidden neurons, are shown in Fig. 13 with error bars (estimated error on the average). The test results of BLM are better than those of OSS, in particular for large networks, a result that confirms less overtraining for BLM. For 400 hidden units, results are 7% better for BLM. ELM needs a very large number of hidden units to obtain competitive results, but the error for 4000 hidden nodes (0.0758) is still 10% worse than the best BLM results.

In order to understand if the difference in validation is related to the final size of weights, the final distribution of weights is shown in Fig. 14. The histogram shows that the better generalization for BLM cannot be explained in a simple manner by a smaller average weight magnitude, which actually tends to be larger for BLM. There is evidence that the search trajectory in the weight space of BLM explores parts that are not explored by the more greedy OSS based on gradient descent.

B. `Yeast` Benchmark

The `yeast` benchmark task is a classification problem. With our unary output encoding (1 for the correct class, 0 otherwise), at least two error measures are of interest, the rms error and the CE error. The second better reflects the classification nature of the task, but the first is also reported for uniformity with the other benchmarks considered in this paper.

The parameters of the runs are symmetric sigmoid (hidden layer) and standard 0-1 sigmoid (output layer). The initial weight range is ± 0.001; two values (8 and 12) are tested for the weight range in BLM. The runs last 500 s. The rms error validation results (Fig. 15) as a function of the number of hidden nodes show superior results by BLM. OSS obtains close results for small numbers of hidden units but shows a larger standard deviation. ELM needs a very large hidden layer to reach result of interest. For a quantitative comparison, the best average validation results are of 0.231 for BLM, of 0.237 for OSS, and of 0.238 for ELM.

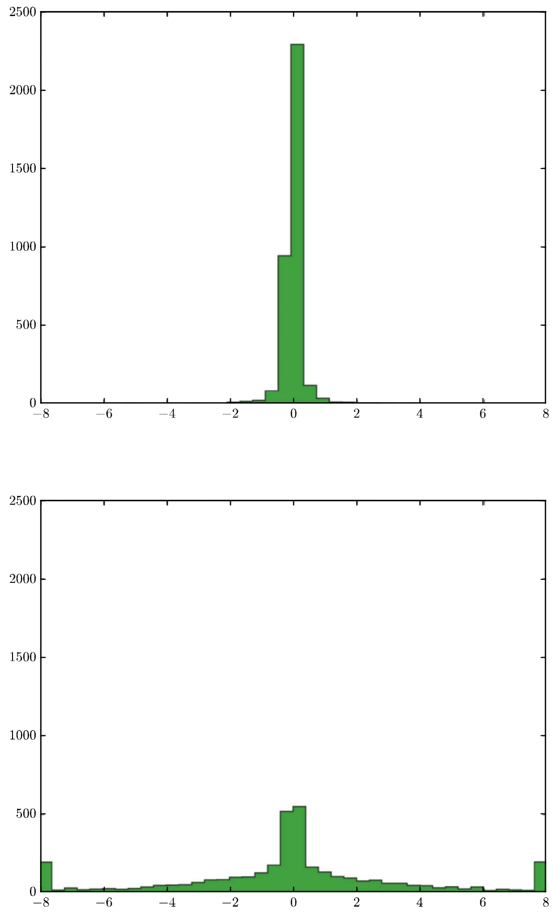

Fig. 14. `House` benchmark. Distribution of weight values (200 hidden units). Top: OSS. Bottom: BLM.

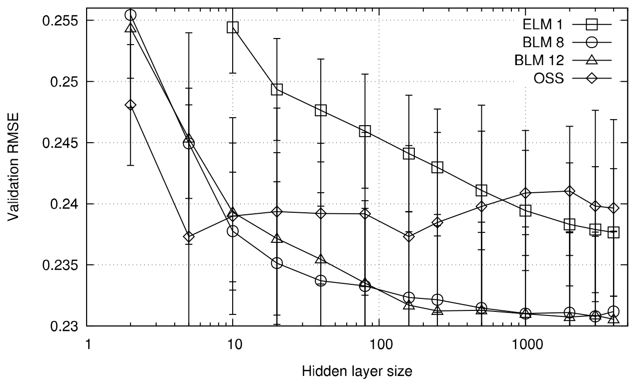

Fig. 15. `yeast` benchmark (rms error). Validation results as a function of hidden layer size. BLM with maximum weight range 8 (BLM 8) and 12 (BLM 12), ELM, and OSS.

The two plots for the different weight ranges (8 and 12) for BLM show that the detailed choice of this parameter is not critical (a value between 6 and 12 usually corresponds to a plateau of best results).

The results for the CE error (Fig. 16) show similar validation results for OSS and BLM, and inferior results for ELM. In an attempt to improve BLM results, different values of the regularization parameter (from 0.1 to 100) have been tested. Two plots for values 1 and 10 are shown. The parameter does

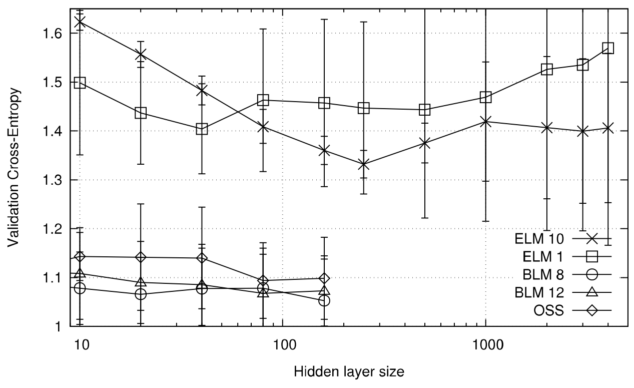

Fig. 16. `yeast` benchmark (CE error). Validation results as a function of hidden layer size. BLM with maximum weight range 8 (BLM 8) and 12 (BLM 12), ELM with regularization parameter 1 (ELM 1) and with regularization parameter 10 (ELM 10), and OSS.

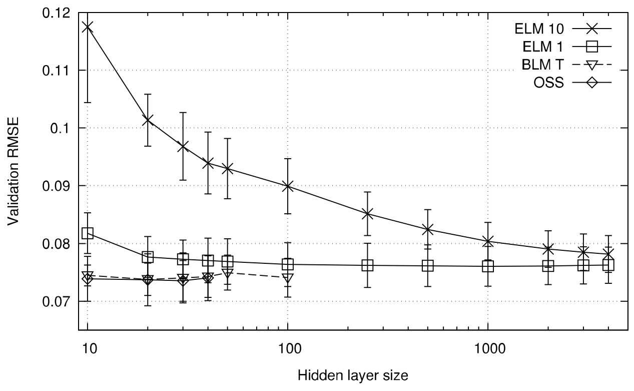

Fig. 17. `abalone` benchmark (rms error). Validation results as a function of hidden layer size. BLM with maximum weight range 8, telescopic BLM, ELM with regularization parameter 1 (ELM 1) and with regularization parameter 10 (ELM 10), and OSS.

influence the results, but they remain in any case far from those of BLM and OSS.

C. `Abalone` Benchmark

The parameters of the runs are: symmetric sigmoid (hidden layer) and linear (output layer). The initial weight range is ± 0.001; two values (1 and 10) are tested for the regularization parameter of ELM. For BLM, both the telescopic and the normal version are compared. The runs last 500 s.

The average rms error on validation for different numbers of neurons in the hidden layer is shown in Fig. 17. One can observe that the OSS and BLM results are similar. The results of normal or telescopic BLM are almost indistinguishable. As usual, ELM needs a fat hidden layer to reach competitive results, but results with 4000 hidden units are still larger than those of OSS/BLM.

D. `Comp_Act`, `Housing`, and `Pole_Tlc` Data Sets

The results in Fig. 18 were obtained for the three remaining data sets by comparing ELM with unit regularization factor, telescopic BLM, and OSS; each point is the average of 12 runs. Again, for small hidden layer sizes, telescopic BLM and OSS perform better than ELM; however, as the number of hidden

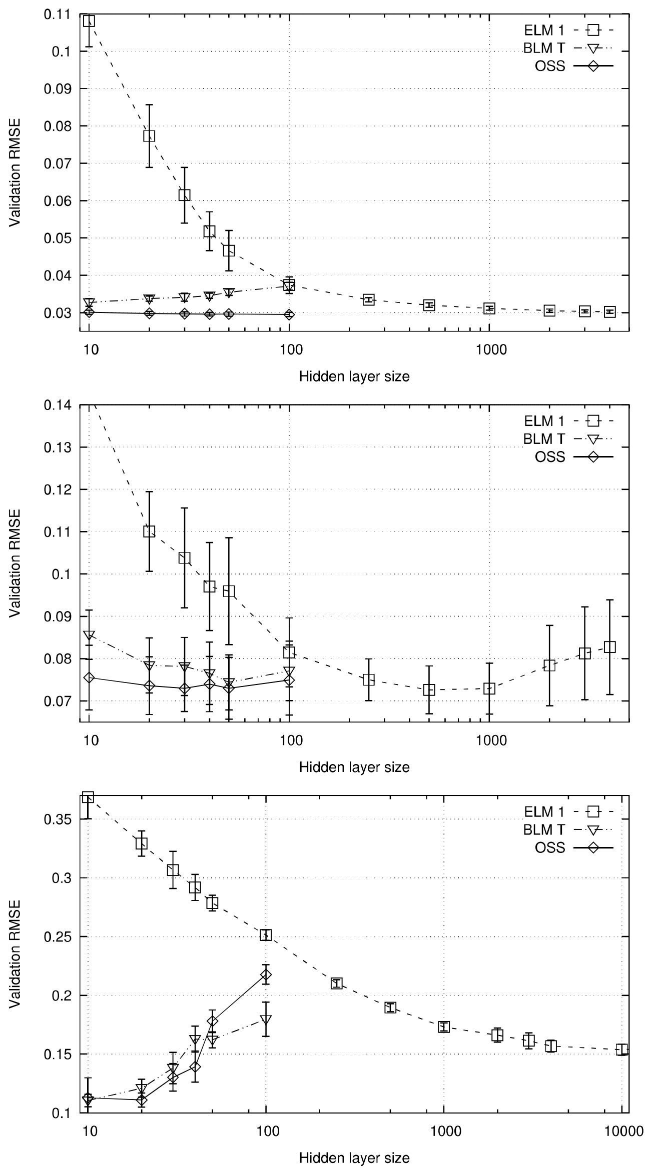

Fig. 18. Benchmark results (rms error) for the remaining data sets comp_act (top), housing (middle), and pole_tlc (bottom). Validation results are reported as a function of hidden layer size.

nodes hits the practical limit for iterated methods, ELM keeps improving and, in some cases, obtains comparable or better results.

The effect of the hidden layer size on performance is particularly visible in the pole_tlc tests, whose relevant number of inputs (48) causes the number of weights to grow too large to be manageable as soon as the hidden layer is larger than a few tens of nodes. While the ELM algorithm significantly improves its performance when a large number of hidden nodes are allowed, it does not attain the same accuracy as in the other cases.

E. Error and CPU Time Comparison

Table II shows a full comparison between the different tested algorithms on the data sets, reporting the average error and execution time corresponding to the optimal hidden layer size (the minima of all curves from Figs. 13–18). Average rms error figures are reported with a confidence interval (given as three times the sample error of the average), computed over 12 runs. For the yeast data set, the results of CE optimizations are also reported. The execution time of each BLM and OSS run was capped at 4000 s.

With most data sets, the different rms errors are comparable within confidence intervals, the main exception being the yeast data set with the CE error, where BLM and OSS outperform ELM, possibly due to the different objective function being optimized, which prevents the straightforward use of the pseudoinverse matrix. The corresponding results for the rms error are not very significant due to the problem's nature, with an optimal number of hidden neurons that varies from 5 (OSS) to 4000 (BLM and ELM) with similar results.

As expected, the optimal hidden layer size for ELM is larger due to the randomness of the weights, leading to networks with a much larger number of parameters.

Times to minima are almost always longer (often by orders of magnitude) for BLM with respect to ELM, because of ELM's noniterative nature, and its execution time is dominated by the pseudoinverse computation. In some larger data sets such as house and pole_tlc, ELM time becomes comparable with that of iterative algorithms, and OSS actually outperforms both ELM and BLM.

F. Comparison With Genetic Algorithms

To compare BLM with GAs, we consider the ANNA ELEONORA technique proposed in [5]. The algorithm uses an efficient variable-length description of the network, encourages sparseness by using a deactivation flag for each node, and introduces, alongside mutation and crossover, a custom operator called G-simplex that takes three candidate solutions x_1, x_2, and x_3 in order of fitness and generates a new one by bitwise majority between x_1, x_2, and $\neg x_3$.

After reproducing the results in [5] to check that our implementation follows the original description, we verified that the ANNA ELEONORA algorithm does not achieve good results on the tested benchmarks (abalone and house), where the rms error remains higher by 25%. The best validation result for abalone among various parameter combinations is around 0.095 against 0.075 when using the other algorithms.

The main problem is related to the little robustness of GAs when applied to problems with continuous outputs, where the output neurons are required not to saturate, but to provide a continuous response. This, together with the difficulties of tuning the large number of parameters a typical GA relies upon, discourages the use of GAs with respect to more state-of-the-art algorithms, such as ELM and OSS.

VI. Tests on a Control Problem: The Inverted Pendulum

In this section, we describe our investigation on the use of BLM-trained networks for applications in control problems, with the network acting as a controller component of a feedback loop, as shown in Fig. 19 (left).

TABLE II

AVERAGE BEST RESULTS FOR DATA SET BENCHMARKS. FOR EACH DATA SET AND ALGORITHM, THE NUMBER OF HIDDEN NODES WITH THE LOWEST AVERAGE ERROR IS REPORTED, TOGETHER WITH THE CORRESPONDING AVERAGE EXECUTION TIME AND ERROR

Dataset	Telescopic BLM			ELM ($\lambda = 1$)			OSS		
	Hidden	Time (s)	Error	Hidden	Time (s)	Error	Hidden	Time (s)	Error
`abalone`	20	431	.0738±.003	2000	4.2	.0761±.003	30	110	.0735±.004
`comp_act`	10	1450	.0327±.001	4000	57	.0302±.0004	100	4000	.0295±.0006
`house`	200	4000	.0693±.002	2000	2146	.0759±.002	25	2120	.0719±.004
`housing`	50	428	.0744±.007	500	.12	.0726±.006	50	210	.0726±.009
`pole_tlc`	10	3085	.109 ±.01	10000	718	.154 ±.005	10	97	.105 ±.005
`yeast` (RMSE)	4000	505	.231 ±.007	4000	14	.238 ±.005	5	148	.238 ±.01
`yeast` (CE)	160	498	1.06 ±.09	250	.04	1.332 ±.03	80	121	1.09 ±.08

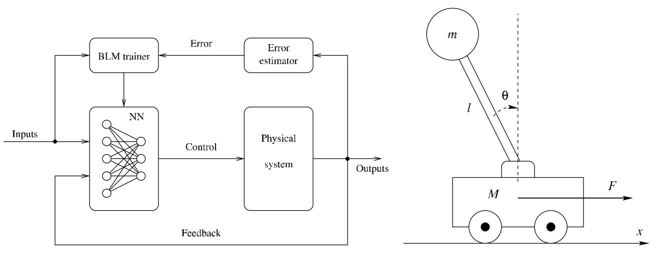

Fig. 19. BLM-trained neural network as a controller (left) and the physical system that we want to control (right).

In the scheme, we can exploit the main advantage of BLM as a derivative-free training algorithm. The controlled physical system can be treated as a black box, with the only obvious addition of a system-specific error evaluation function (top right) whose numeric output measures the correctness of the system's behavior.

In our case study, we simulate the system shown in Fig. 19 (right): an inverted pendulum mounted on a cart on which a control force is applied. The cart has mass $M = 1$ kg and is constrained to move along the x-direction, while the pendulum has length $l = 1$ m and mass $m = 1$ kg applied on the end. Other masses are negligible. The control variable is a force applied to the cart along its moving direction.

The dynamics of the system are described by the following second-order, nonlinear system of differential equations:

$$\ddot{x} = \frac{F - m \sin\theta (l\dot{\theta}^2 - g\cos\theta)}{M + m\sin^2\theta}$$
$$\ddot{\theta} = \frac{\ddot{x}\cos\theta + g\sin\theta}{l}.$$

The correct behavior of the system is to keep the pendulum in the upright position $\theta = 0$, with the cart at a specified coordinate $x = 0$. If we simulate the system in the time interval $[0, T]$, then the error is given by a combination of the mean squares of the two position variables x, θ

$$\mathrm{Err}(x, \theta) = \frac{1}{T - t_{\min}} \int_{t_{\min}}^{T} (\theta^2(t) + \lambda x^2(t))\, dt \qquad (6)$$

where λ is a suitable weighting factor (in our experiments, $\lambda = 0.01$ m^{-2}), while $t_{\min} \geq 0$ defines a transient portion of the time interval that does not contribute to the error.

In the described test, we use a feedforward neural network with a five-neuron hidden layer (with tanh as transfer function) and a single output neuron that provides the control force.

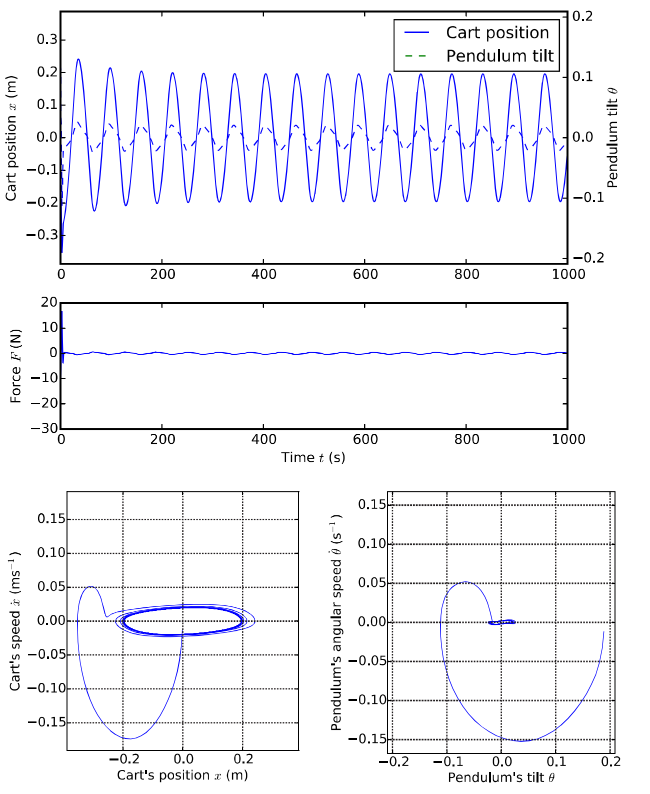

Fig. 20. Inverted pendulum with position and speed feedback $(x, \theta, \dot{x}, \dot{\theta})$, feedforward MLP without recurrence, five hidden units, and optimization on 2 b/weight. 2 b are sufficient to keep the pendulum from falling, although control is coarse and both the pendulum and the cart keep moving back and forth along a limit cycle, as shown by the two phase-space diagrams in the bottom.

The network has four inputs: the two position variables x, θ and their time derivatives $\dot{x}, \dot{\theta}$. We use 16-b weights, with 10 as maximum value, initialized to small random values in the $[-10^{-2}, 10^{-2}]$ interval.

For training, we generate 50 pendulum simulations, each running for a simulated time $T = 100$ s. We use first-order (Euler) integration with time increment $\Delta t = 10^{-2}$ s. The control force is kept constant for ten consecutive steps, after which the system's status is fed to the neural net, and the new output is used for the subsequent ten iterations. Therefore, the net is queried 1000 times per simulation. The first ten queries, corresponding to $t_{\min} = 1$ s of simulated time, do not account for the error. Every 100 iterations, a different set of 50 simulations is used to validate the training, and the weights corresponding to the best validation are saved.

Fig. 20 shows the behavior of a network with five hidden units trained with only 2 b/weight on a sample test instance; to make sure that the stability is not temporary, the test simulation runs for 1000 simulated seconds, i.e., tenfold the actual training and validation period. The top chart of Fig. 20 shows the behavior of the two positional variables x and θ during the simulation. The network's control, however coarse due to the few representable weight values, is sufficiently precise to prevent the pendulum from falling, although the cart tends to drift back and forth (remember that we gave a relatively small value to the penalty λ associated with the cart position x). The middle chart in Fig. 2 shows the corresponding applied force. After a transient high force to correct the unbalanced initial conditions of the system, much less force needs to be applied to keep the system upright. The ensuing periodicity is apparent in phase space, where the pairs of coordinates $(x, \dot{x})$ and $(\theta, \dot{\theta})$ both converge to a limit cycle. The mean error of the network as defined in (6), computed on 50 random initial conditions, is $\text{Err} = 2.34 \times 10^{-2}$. If a finer representation is chosen for weights (e.g., 16 b), then the system comes to a relatively smooth halt in a short time, with error $\text{Err} = 8.9 \times 10^{-3}$. See the Appendix[3] for further details.

VII. Conclusion

The objective of this paper was to revisit simple SLS as building block for complex ML tasks. The BLM algorithm is very simple from the point of view of discrete dynamical systems in weight space; complexity is delegated to the design of efficient methods and data structures to enormously reduce CPU times.

The results were counterintuitive. Our telescopic BLM algorithm not only reproduced results by more complex methods, but actually surpassed them by a statistically significant gap in some cases. In addition, the performance produced by extreme learning was improved, usually with much smaller networks (with less hidden units), although ELM is much faster.

The experimental results indicate that a simple method like BLM based on SLS and an adaptive setting of the number of bits per weight is fully competitive even with more complex approaches based on derivatives.

The speedup obtained by supporting data structures, incremental evaluations, a small and adaptive number of bits per weight, and stochastic first-improvement neighborhood explorations is of about two orders of magnitude for the benchmark problems considered, and it increases with the network dimension. The CPU training time is acceptable if the application area does not require very fast online training.

BLM results motivate a wider application for more complex ML schemes. We encourage researchers to consider special-purpose hardware implementations, or realizations on GPU accelerators. Some recent examples of special-purpose hardware are the architecture for probabilistic computing in [26] and the hybrid processing hardware in [27]. Other areas, in which Boolean variables are used, can also be considered for the extensions of this approach (see, for example, Boolean factor analysis [28]). Ensembling [29] and feature selection [30] are other promising avenues, as well as the theoretical analysis of the dynamics in weight space produced by telescopic BLM.

In digital computers, bits dominate and real numbers are only simulated. Working directly with bits can lead to surprising and effective results.

[3]http://intelligent-optimization.org/papers/tnnls2016a.pdf.

References

[1] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, nos. 1–3, pp. 489–501, 2006.

[2] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Comput. Sci. Rev.*, vol. 3, no. 3, pp. 127–149, 2009.

[3] J. Tian, M. Li, F. Chen, and N. Feng, "Learning subspace-based RBFNN using coevolutionary algorithm for complex classification tasks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 1, pp. 47–61, Jan. 2016.

[4] R. S. Sexton, R. E. Dorsey, and J. D. Johnson, "Optimization of neural networks: A comparative analysis of the genetic algorithm and simulated annealing," *Eur. J. Oper. Res.*, vol. 114, no. 3, pp. 589–601, 1999.

[5] V. Maniezzo, "Genetic evolution of the topology and weight distribution of neural networks," *IEEE Trans. Neural Netw.*, vol. 5, no. 1, pp. 39–53, Jan. 1994.

[6] F. H. F. Leung, H. K. Lam, S. H. Ling, and P. K. S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," *IEEE Trans. Neural Netw.*, vol. 14, no. 1, pp. 79–88, Jan. 2003.

[7] A. Corana, M. Marchesi, C. Martini, and S. Ridella, "Minimizing multimodal functions of continuous variables with the 'simulated annealing' algorithm," *ACM Trans. Math. Softw.*, vol. 13, no. 3, pp. 262–280, 1987.

[8] R. Battiti and G. Tecchiolli, "Training neural nets with the reactive tabu search," *IEEE Trans. Neural Netw.*, vol. 6, no. 5, pp. 1185–1200, Sep. 1995.

[9] R. Battiti and M. Brunato, *The LION Way: Machine Learning Plus Intelligent Optimization*. Trento, Italy: LIONlab, Univ. Trento, 2014.

[10] Y. He, Y. Qiu, G. Liu, and K. Lei, "Optimizing weights of neural network using an adaptive tabu search approach," in *Proc. 2nd Int. Conf. Adv. Neural Netw.*, 2005, pp. 672–676. [Online]. Available: http://dx.doi.org/10.1007/11427391_107.

[11] R. S. Sexton, B. Alidaee, R. E. Dorsey, and J. D. Johnson, "Global optimization for artificial neural networks: A tabu search application," *Eur. J. Oper. Res.*, vol. 106, nos. 2–3, pp. 570–584, 1998.

[12] B. A. Garro, H. Sossa, and R. A. Vazquez, "Design of artificial neural networks using a modified particle swarm optimization algorithm," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jun. 2009, pp. 938–945.

[13] J. N. D. Gupta and R. S. Sexton, "Comparing backpropagation with a genetic algorithm for neural network training," *Omega*, vol. 27, no. 6, pp. 679–684, 1999.

[14] P. L. Barlett and T. Downs, "Using random weights to train multilayer networks of hard-limiting units," *IEEE Trans. Neural Netw.*, vol. 3, no. 2, pp. 202–210, Mar. 1992.

[15] M. Brunato and R. Battiti, "Stochastic local search for direct training of threshold networks," in *Proc. Int. Joint Conf. Neural Netw.*, 2015, pp. 1–8.

[16] R. Battiti, M. Brunato, and F. Mascia, *Reactive Search and Intelligent Optimization* (Operations Research/Computer Science Interfaces Series), vol. 45. New York, NY, USA: Springer-Verlag, 2009.

[17] P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications*. Norwell, MA, USA: Kluwer, 1987.

[18] J.-L. Starck, F. D. Murtagh, and A. Bijaoui, *Image Processing and Data Analysis: The Multiscale Approach*. Cambridge, U.K.: Cambridge Univ. Press, 1998.

[19] E. Mjolsness, C. D. Garrett, and W. L. Miranker, "Multiscale optimization in neural nets," *IEEE Trans. Neural Netw.*, vol. 2, no. 2, pp. 263–274, Mar. 1991.

[20] K. J. Lang and M. J. Witbrock, "Learning to tell two spirals apart," in *Proc. Connectionist Models Summer School*, 1988, pp. 52–59.

[21] R. Battiti, "First- and second-order methods for learning: Between steepest descent and Newton's method," *Neural Comput.*, vol. 4, no. 2, pp. 141–166, Mar. 1992.

[22] R. Battiti and G. Tecchiolli, "Learning with first, second, and no derivatives: A case study in high energy physics," *Neurocomputing*, vol. 6, no. 2, pp. 181–206, 1994.

[23] G.-B. Huang, Q.-Y. Zhu, K. Z. Mao, C.-K. Siew, P. Saratchandran, and N. Sundararajan, "Can threshold networks be trained directly?" *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 53, no. 3, pp. 187–191, Mar. 2006.

[24] K. Nakai and M. Kanehisa, "A knowledge base for predicting protein localization sites in eukaryotic cells," *Genomics*, vol. 14, no. 4, pp. 897–911, 1992.

[25] S. M. Weiss and N. Indurkhya, "Rule-based machine learning methods for functional prediction," *J. Artif. Intell. Res.*, vol. 3, pp. 383–403, Dec. 1995.

[26] V. Canals, A. Morro, A. Oliver, M. L. Alomar, and J. L. Rossello, "A new stochastic computing methodology for efficient neural network implementation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 3, pp. 551–564, Mar. 2016.

[27] C. Kyrkou, C.-S. Bouganis, T. Theocharides, and M. M. Polycarpou, "Embedded hardware-efficient real-time classification with cascade support vector machines," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 1, pp. 99–112, Jan. 2016.

[28] A. A. Frolov, D. Húsek, and P. Y. Polyakov, "Comparison of seven methods for Boolean factor analysis and their evaluation by information gain," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 3, pp. 538–550, Mar. 2016.

[29] H. Chen, H. Chen, X. Nian, and P. Liu, "Ensembling extreme learning machines," in *Advances in Neural Networks*. Heidelberg, Germany: Springer, 2007, pp. 1069–1076.

[30] R. Battiti, "Using mutual information for selecting features in supervised neural net learning," *IEEE Trans. Neural Netw.*, vol. 5, no. 4, pp. 537–550, Jul. 1994.

Mauro Brunato (M'99) received the Laurea degree in mathematics from the University of Padua, Padua, Italy, and the Ph.D. degree from the University of Trento, Trento, Italy, in 1999.

He is currently an Assistant Professor with the University of Trento. He has co-authored books, book chapters, and articles about learning and intelligent optimization and holds courses on the topic. His current research interests include machine learning and optimization, in particular, neural networks, feature generation and selection, combinatorial and continuous optimization, and data analysis and visualization.

Roberto Battiti (F'09) received the Laurea degree in physics from the University of Trento, Trento, Italy, and the Ph.D. degree from the California Institute of Technology, Pasadena, CA, USA, in 1990.

He is currently a Full Professor of Computer Science and the Director of the Machine Learning and Intelligent Optimization Laboratory with the University of Trento. He is intrigued by issues at the boundary between optimization and machine learning. He is the creator of Reactive Search Optimization, advocating the integration of machine learning techniques into search heuristics for solving complex optimization problems. He has authored over 100 scientific publications, including the recent book entitled *The LION Way*, which led to hundreds of applications in widely different fields.