Stochastic Local Search for Direct Training of Threshold Networks

Mauro Brunato*, Roberto Battiti*†
*DISI - University of Trento, Italy
†Lobachevsky State University of Nizhny Novgorod, Russia
Email: {brunato,battiti}@disi.unitn.it

***Abstract*—This paper investigates Stochastic Local Search (SLS) algorithms for training neural networks with threshold activation functions. and proposes a novel technique, called Binary Learning Machine (BLM). BLM acts by changing individual bits in the binary representation of each weight and picking improving moves.**

While brute-force implementations of SLS lead to enormous CPU times, due to the limited extent of each move, the use of incremental neighborhood evaluation, first-improving strategies, and Gray encoding accelerates SLS by a huge factor, opening the way to affordable CPU times for a wide range of problems.

Comparisons with alternative methods demonstrate the effectiveness of the approach. In details, BLM outperforms state-of-the-art techniques either by achieving better generalization properties, or by allowing for more compact networks, suitable for the type of applications for which threshold neural networks were originally introduced.

I. Introduction

Starting from the seminal work related to backpropagation [1], [2], most techniques for training neural networks use continuous optimization schemes based on partial derivatives, like gradient descent and variations thereof. Each weight is considered as a real variable and the performance on training examples is optimized, possibly with early stopping or additional regularizing terms in the function to be optimized in order to increase generalization. Support Vector Machines (SVMs) share this approach based on continuous optimization, actually using quadratic optimization to take care of constraints [3]. More recently, extreme learning [4], reservoir computing [5] and the no-prop approach of [6] simplify the optimization task by creating a randomized first layer and limiting optimization to the last-layer weights.

A different research stream considers Combinatorial Optimization (CO) techniques without derivatives, like Simulated Annealing and Genetic Algorithms, see [7] for a comparative analysis. Complex memory-based Reactive Tabu Search is considered in [8]. 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 cross-over operators in GA). The main difference of the BLM approach w.r.t. these more sophisticated schemes like GA lies in simplicity of the building blocks. We deem important to push the limits of simple algorithmic schemes based on pure local search with state-of-the-art implementations, to see if they match the performance of more complex ones. For example, we deem important from a scientific and technological point of view to assess if the performance of GAs is caused by complex genetic operators or more by the presence of underlying local search methods, disguised under the genetic terminology.

Derivatives cannot be used directly for threshold networks [9], with transfer functions characterized by two values, depending on a threshold on the input (this function is discontinuous and therefore not always differentiable). Hard-limiting output functions are desirable for the following reasons. First, a unit with a threshold output function is much less costly to implement in digital hardware than a unit with the conventional logistic output function. The computation of activations in non-input units is considerably simpler, since it involves only additions and subtractions. Second, the relationship between the size of a network and the sample complexity of learning is better understood for units with threshold output functions [10]. In [11] the activation functions of hidden neurons are gradually transformed during training from pure analog units to pure threshold ones. The method in [12] adopts the "pseudo-gradient" (instead of the true gradient), using the gradient of a sigmoid function as a heuristic hint. Network weights as random variables with smooth distribution functions are considered in [9] so that probability of a hard-limiting unit taking one of its two possible values is a continuously differentiable function. The work in [13] shows that extreme learning can be used to train the neural networks with threshold functions directly instead of approximating them with sigmoid functions. In particular, faster training and better generalization results w.r.t. [14] are demonstrated. Let's note that other recent proposals consider units which are different from smooth sigmoidal ones. For example, state-of-the-art in deep learning uses variations of rectified linear units [15], cheap to compute and suitable for modifications of gradient-descent approaches (derivatives are missing at some specific points in the function inputs). Generalizations of the BLM approach presented for different kinds of units are possible, and will be considered in the continuation of this research.

The investigation in this paper considers simple *local search* techniques for directly training threshold network. With respect to Simulated Annealing and Genetic Algorithms, the focus in on the extreme simplicity of the considered local search algorithms. Better said, the investigation deals with pushing the limits of applicability of simple stochastic local search algorithms through a careful implementation and appropriate support data structures. Simplicity means lack of parameters

978-1-4799-1959-8/15/$31.00

to be tuned by the final user and better understanding of the contribution of the individual building blocks to the final results.

The remaining part of this paper is organized as follows. Section II introduces the state-of-the art methods for training threshold networks, describes the stochastic local search techniques that provide the foundation of the proposed BLM algorithm, and describes the benchmarks on which comparisons will be based. Section III describes the building blocks of the BLM algorithm and provides analysis and motivations for its design choices. In section IV, experimental results on BLM and other techniques are presented and discussed. Section V concludes this paper.

II. State of the art methods for training threshold networks and benchmark tasks

A. Threshold networks

The threshold transfer function

$$f(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

has null derivative for all $x \in \mathbb{R} \setminus \{0\}$, and has no derivative in $x = 0$. Therefore, classical gradient-descent based methods cannot be directly applied.

Early proposals for threshold network training consider the approximation of the threshold function with steeper and steeper versions of a more typical sigmoidal transfer function, for example by using a sigmoid function with a gain parameter λ:

$$g(x) = \frac{1}{1 + e^{-\lambda x}} \tag{2}$$

instead of the real threshold function. The gain parameter is gradually increased during the training until the slope of the sigmoid is sufficiently large to allow a transfer to a threshold network with the same architecture.

In [13], a direct method for training threshold networks is proposed. Based on the Extreme Learning paradigm [4], the ELM algorithm applies to a feed-forward neural network with a single hidden layer and operates in two phases. In the first step, input weights are initialized with uniform random values in a suitable range. Next, output weights are determined by solving a linear least-squares approximation on the training examples. In detail, if the network has n neurons in the hidden layer and the training set contains m examples, let $H \in \{0,1\}^{m \times n}$ be the $\{0,1\}$-valued matrix whose row i contains the activation state of the threshold neurons for training sample i. Consider the k-th output neuron, and let $\boldsymbol{w}_k = (w_{k1}, \ldots, w_{kn})$ be its input weights. Then the neuron's outputs for the m training samples is

$$\boldsymbol{y}_k = H\boldsymbol{w}_k$$

Therefore, the optimal weights $\hat{\boldsymbol{w}}_k$ can be found by computing the Moore-Penrose pseudo-inverse of H and applying it to the vector of training sample outputs $\hat{\boldsymbol{y}}_k$:

$$\hat{\boldsymbol{w}}_k = H^{\dagger}\hat{\boldsymbol{y}}_k,$$

where

$$H^{\dagger} = (H^T H)^{-1} H^T. \tag{3}$$

B. Stochastic local search

A basic problem-solving strategy consists of starting from an initial tentative solution and trying to improve it through repeated small changes. At each repetition the current configuration is slightly modified (*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 is called *fitness* function, *goodness* function, or *objective* function.

Local search based on perturbing a candidate solution is a first paradigmatic case where simple learning strategies can be applied. Let's define the notation. $\mathcal{X}$ is the search space, $X^{(t)}$ 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:

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

In our case, the search space is given by binary strings with a given length L: $\mathcal{X} = \{0,1\}^L$, and the moves are those changing (or complementing or *flipping*) the individual bits, and 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:

$$Y \leftarrow \text{IMPROVING-NEIGHBOR}(\ N(X^{(t)})\ ) \tag{4}$$

$$X^{(t+1)} = \begin{cases} Y & \text{if } f(Y) < f(X^{(t)}) \\ X^{(t)} & \text{otherwise (search stops).} \end{cases} \tag{5}$$

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. For robustness, neighbors are evaluated in a stochastic order until an improving move is identified.

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 combinatorial optimization 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. If one starts at a good solution, solutions of similar quality can, on the average, be found more *in its neighborhood* than by sampling a completely unrelated random point. By the way, sampling a random point generally is much more expensive than sampling a neighbor, provided that the f value of the neighbors can be updated ("*incremental evaluation*") and it does not have to be recalculated from scratch. Stochastic elements can be immediately added to local search to make it more robust [16], for example a random sampling of the neighborhood can be adopted instead of a trivial deterministic and exhaustive consideration.

For many optimization problems of interest, a closer approximation to the global optimum is required, and therefore more complex schemes are needed in order to continue the investigation into new parts of the search space, i.e., to *diversify* the search and encourage *exploration*, see for example [17],

TABLE I
SPECIFICATION OF THE 4 BENCHMARK SETS USED IN COMPARISONS

Dataset name	Training samples	Test samples	Attributes
abalone	2000	2177	$7+1$
delta_elevators	4000	5517	6
house16H	10000	12784	16
computer_activity	4000	4192	12

[18] for a much more extended presentation and discussion of techniques.

In this paper, we will 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.

C. Benchmarks

Experimental results are based on the four benchmark real-life regression problems listed in Table I. The datasets have been chosen among the larger ones presented in [13].

The abalone dataset contains data on specimens of the Abalone mollusk; 7 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 10 inputs were used. The output column is the age of the specimen (number of accretion rings in the shell).

The delta_elevators dataset contains a subset of navigation data of an F16 aircraft, with 6 inputs related to the vessel's status; the learning task consists in predicting the corresponding variation in the position of elevators.

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

The computer_activity dataset reports counts of different activities performed by processes running on a computer; the task is predicting the percentage of time a CPU is running in user mode.

The training and test set sizes are those used by [13]. All dataset inputs have been normalized by an affine mapping of the training set inputs onto the $[-1, 1]$ range and 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.

III. STOCHASTIC LOCAL SEARCH FOR NEURAL NETWORKS: A SMART MEMORY-BASED IMPLEMENTATION

In this Section we illustrate the main building blocks leading to a fast and effective implementation of SLS for neural network training, with particular focus on threshold networks. The resulting algorithm is called Binary Learning Machine (BLM) to underline the fact that it is based on the binary representation of weights.

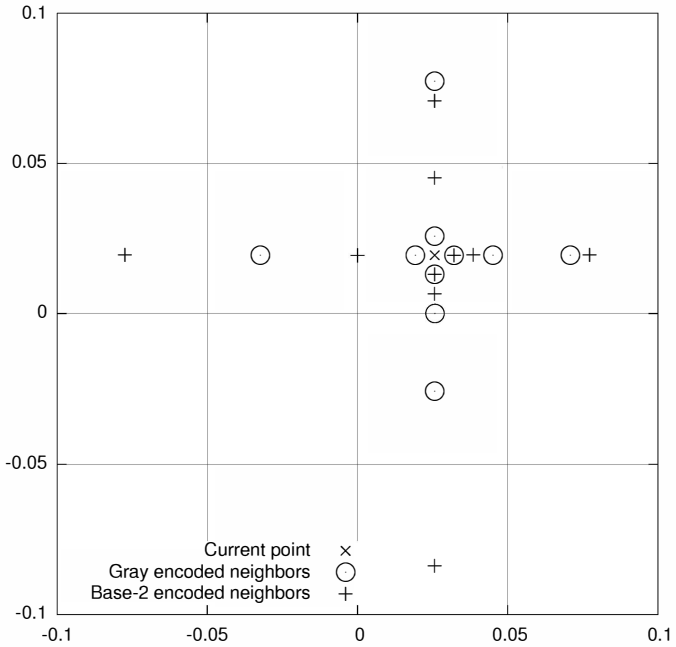

Fig. 1. Neighbors of the current point with Gray coding vs. standard base-2 coding.

A. Neural network

The network used in all simulations is a feed-forward MLP with a single hidden layer and full connections between layers. Hidden layer neurons are equipped with a $0, 1$ threshold transfer function (1), while the output neurons have a linear response, i.e., no transformation is applied to their outputs.

B. Weight representation

Many possible weight representation choices can influence the learning performance. Preliminary investigations allowed us to restrict the relevant parameters.

1) Number of bits and discretization: We represent weights and biases as integral multiples of a discretization value $\epsilon > 0$, with an n-bit two's-complement integer as multiplier. Options $n = 8, 16, 24$ were investigated. In all cases, ϵ was set so that representable weights would span a given range of weights. Empirical investigation shows that weights in the range $[-.1, .1]$, coupled with small initial random values in the range $[-.001, .001]$ are adequate for the type of network that we are considering.
Simulations performed on the abalone dataset with a 20-node hidden layer MLP on a fixed (60s) time budget show that, while a larger number of bits often outperforms the 8-bit network on the training set, performance on the test set is equivalent within a 95% confidence interval.
Subsequent experiments have been performed on 8-bit networks, with representable weight range in $[-.1, 1]$ (corresponding to discretization $\epsilon = .1/(2^7 - 2) \approx .000794$).

2) Binary vs. 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 chose a Gray encoding of the n-bit integer multiplier. The Gray code has the property that the nearby integers

$n-1$ and $n+1$ are obtained by changing a single bit of the code of n. (i.e., the codes of $n+1$ and $n-1$ 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 00000000 and 11111111 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 contains the nearest weights on the discretized grid, plus a cloud of points at growing distances in weight space.

C. Memory-based incremental neighborhood evaluation

Let N_{layers} be the number of layers (hidden and outputs) in the network; let the input layer be identified as layer 0, while the output is layer N_{layers}. For $l = 0, \ldots, N_{\text{layers}}$, let n_l be the number of neurons in layer l. For $j = 1, \ldots, n_l$, let o_j^l be the output of neuron j in layer l. Therefore, let o_j^0 be the value of the j-th input. Let $f^l(x)$ be the transfer function of neurons at layer l, and w_{ij}^l be the weight connecting neuron i at layer $l-1$ to neuron j at layer l. Starting from the input values, the feed-forward computation consists of iterating the following operation for $l = 1, \ldots, N_{\text{layers}}$:

$$o_j^l = f^l(s_j^l), \qquad \text{where} \qquad s_j^l = b_j^l + \sum_{i=1}^{n_{l-1}} w_{ij}^l o_i^{l-1}. \quad (6)$$

The simple neighborhood scheme chosen for this investigation requires a large number of steps, each requiring the modification of a single weight. A very significant speedup can be obtained by storing the s_j^l values for all neurons and all samples. In order to obtain the network output when weight w_{IJ}^L is replaced by a new value W, a simpler incremental evaluation is possible by the following algorithm:

1) Recompute the output of neuron J in layer L:
$$o'^L_J = f^L\big(s_J^L + (W - w_I^L J)o_I^{L-1}\big).$$
If $L = N_{\text{layers}}$, the J-th network output is o'^L_J, the others are unchanged. Stop.
2) Let the output variation be
$$\Delta o_J^L = o'^L_J - o_J^L.$$
If $\Delta o_J^L = 0$, then network output is unchanged. Stop.
3) Recompute all contributions of output o_J^L to the neurons of the next layer, for $k = 1, \ldots, n_{L+1}$:
$$s'^{L+1}_k = s_k^{L+1} + w_J^{L+1} k \Delta o_J^L;$$
$$o'^{L+1}_k = f^{L+1}(s'^{L+1}_k).$$
4) If there are other layers, apply (6) for $l = L+2, \ldots, N_{\text{layers}}$.

Obvious modifications apply for bias values.

Consider the case shown in Fig. 2, where $N_{\text{layers}} = 2$ (single hidden layer). Upon modification of an input weight or hidden bias (left part of the figure), 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 modification of an output weight or bias (right side), 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 $O\big(n_0 n_2/(n_0+n_2)\big)$ operations.

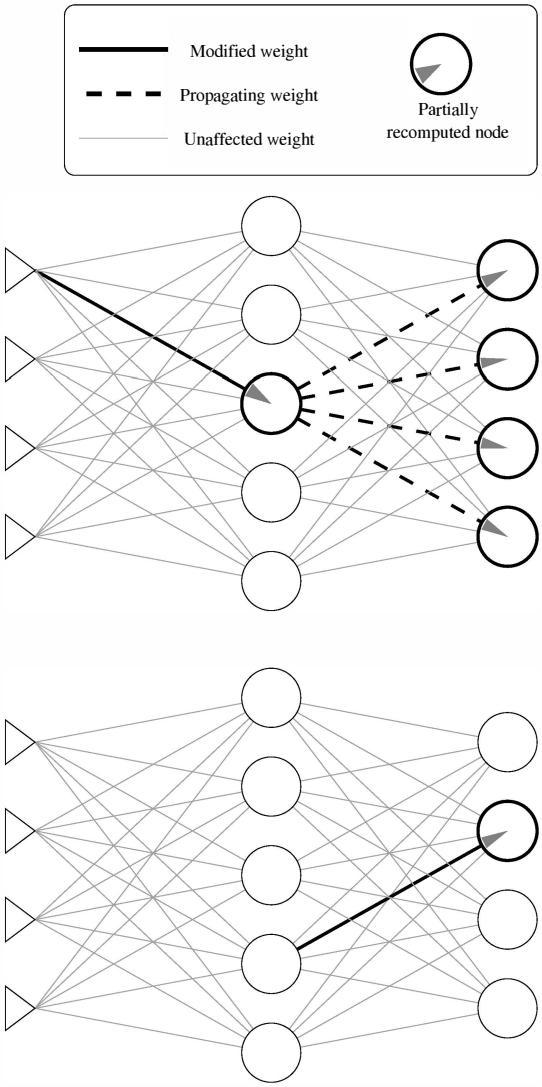

Fig. 2. Affected neurons in an incremental evaluation step with a single changed weight in the input layer (top) or in the output neurons (bottom).

For comparison, the number of operations required by the complete feed-forward procedure is roughly proportional to the number of weights in the network, i.e., $O\big(n_1(n_0+n_2)\big)$. An incremental search step on a 200 hidden node network on the `abalone` dataset is about 55 times faster than the complete evaluation.

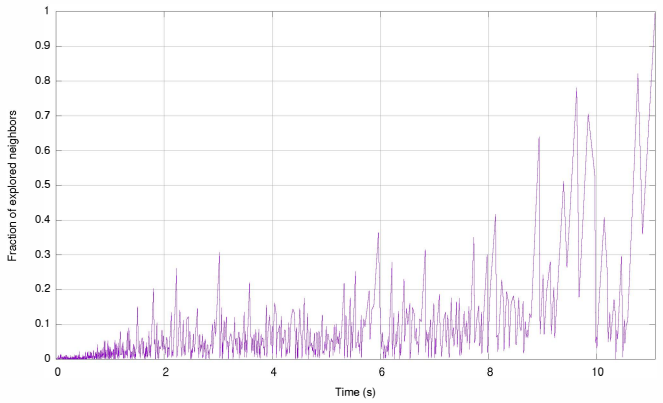

Fig. 3. Fraction of neighborhood that needs to be explored at each step as the search proceeds.

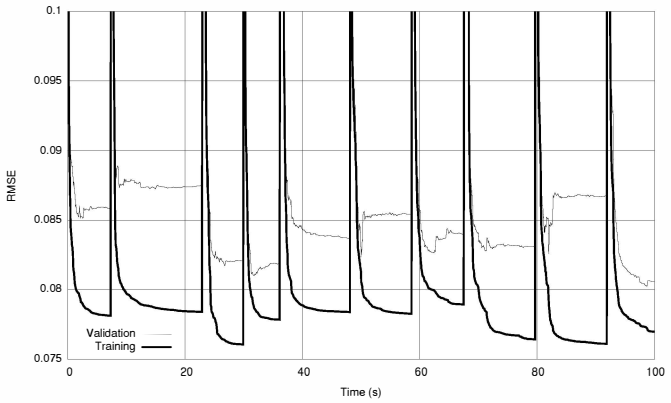

Fig. 4. Training and validation curves on the `abalone` dataset, with restarts after reaching a local minimum in the training function.

D. Neighborhood exploration

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

- *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 secondary objectives.
- *First improving move* — feasible moves are explored in random order, and the first move that leads to an improvement of the target objective is selected.

Preliminary tests of the BLM algorithm on the `abalone` dataset and a 20-node hidden layer MLP suggest that the Best move strategy causes a 10-fold slowdown on iteration times and, while having a steeper initial improvement, on the long run it achieves worse minima. Therefore, in the remainder of our work we shall only consider the First-move strategy.

Fig. 3 shows the fraction of neighborhood that the search algorithm explores at each search step of a representative run, from the initial random configuration until a local minimum is found. 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, however, 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.

The combination of incremental evaluation and first-improving strategy lead to a speedup in the order of 40 times with respect to the baseline algorithm.

E. Structure of the search space

The neighborhood topology induced into the search space by the gray-encoded bit-flip local moves can be quite complex. The main differences with backpropagation techniques based on gradient descent are the discretization of the search space (which becomes a finite, regularly spaced lattice in the weights space) and the actual size of local moves, in particular when high-order bits are changed. On an m-weights network, each represented with n bits, the search space has actually the

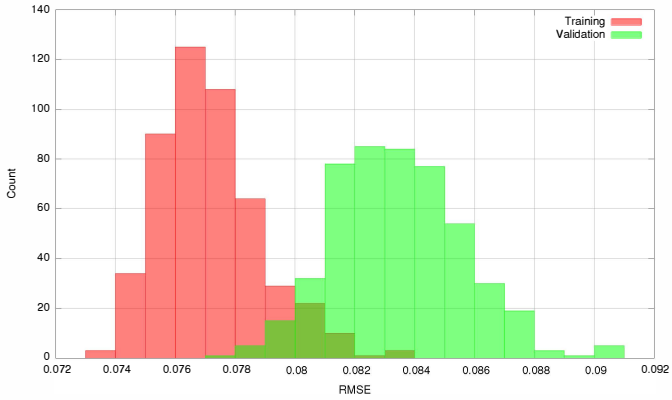

Fig. 5. Distribution of minima for the training and the validation set.

vertex topology of an mn-dimensional hypercube, mapped onto the weight space $\mathbb{R}^m$ by means of a 2^n-discretized grid to compute the target function. Using Gray encoding ensures that the hypercube topology is a superset of the natural first-neighbor topology on the weight lattice.

Preliminary insight on the performance of the proposed technique can be acquired by means of empirical examination of the search space in terms of local minima distribution on the training and validation set.

Figure 4 shows the values of the training (on a 1500-node subset) and validation (on a 500-node subset) RMSE during an execution of the BLM algorithm on a subset of the `abalone` dataset with a 20-units hidden-layer MLP, 8-bit gray encoding of weights with discretization $\epsilon = .1/(2^7 - 2)$. Every time the training procedure cannot find an improving step, a local minimum in the hypercube topology is recorded and a new random configuration is generated. All restarts are statistically independent. Local minima are not necessarily *strict* (strict means that all neighbors are worsening the current value, non-strict means that some neighbors can maintain the *same* value as the current one). Validation minima along the search trajectories are on the average 8% higher than the corresponding training minima, as shown in Figure 5.

Figs. 6 and 7 present results related to studying possible evidence of overtraining by examining the behavior of the

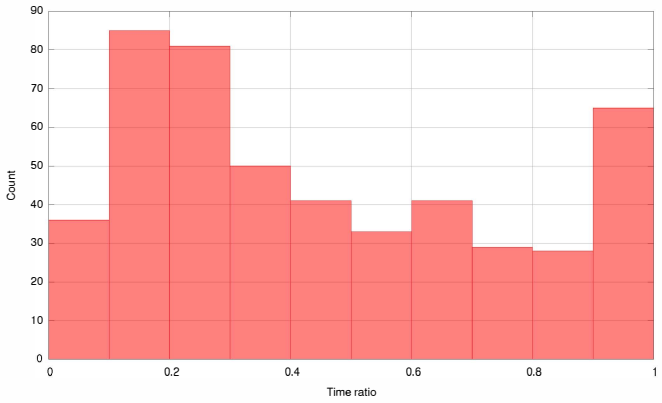

Fig. 6. Ratio between the time at which the validation minimum is found and the total run time for each search trajectory.

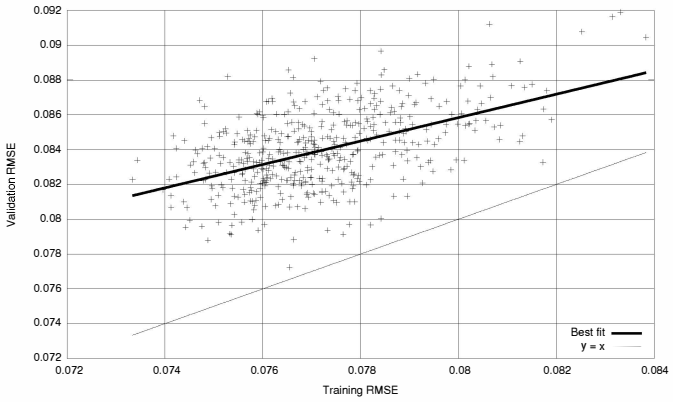

Fig. 7. Training vs. validation RMSE at the end of each local-minimum search (best fit and equality line is also plotted).

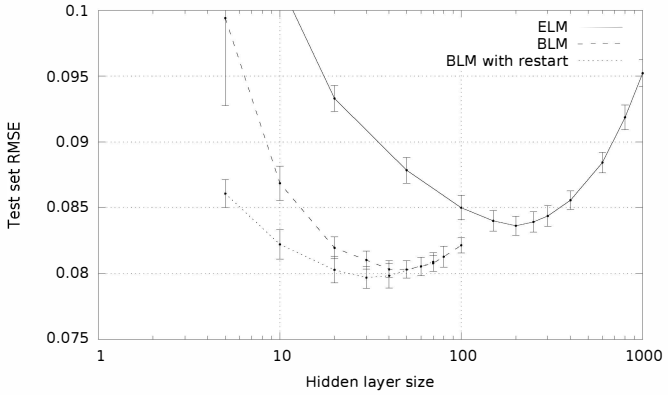

Fig. 8. Comparison between ELM and BLM on the `abalone` benchmark

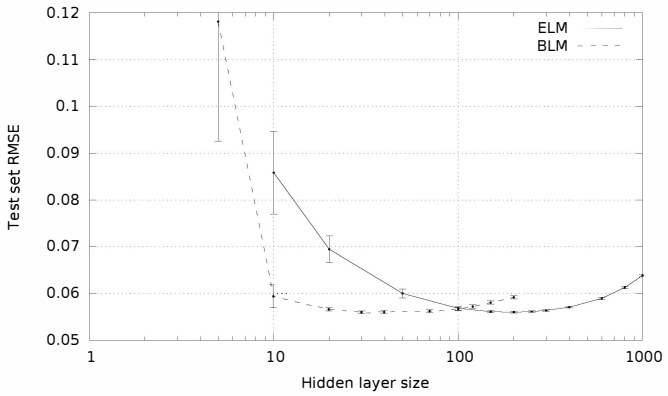

Fig. 9. Comparison between ELM and BLM on the `delta_elevators` benchmark

validation set RMSE during the training. In particular, fig. 6 shows the distribution of the ratio between the time at which the minimum RMSE on the validation set is attained and the total time of the run (the time at which the minimum RMSE on the training set is attained). While a significant fraction of validation RMSE minima is obtained early during the run, we can observe that the distribution of times associates a fairly high frequency to later minima. This result suggests that a local search procedure starting from a random configuration and terminating when a minimum is found is typically not subjected to overtraining. This is confirmed by the distribution of validation and training RMSE values at the end of each run (when a training local minimum is found), as shown in Fig. 7, where final RMSE values for the training and validation sets are compared, showing that the dispersion of validation errors is quite uniform for all training errors.

IV. Experimental comparison

After the preliminary investigation of building blocks discussed in Section III, a complete comparison of the proposed BLM technique with the ELM algorithm has been performed.

A. ELM-specific settings

In order to reproduce the results in [13], settings were maintained when possible. Input weights and hidden biases were drawn from a uniform distribution in $[-1, 1]$. A regularization constant $\lambda = 1$ has been introduced in the Moore-Penrose inverse calculation (3) in order to avoid occasional singular matrices:

$$H_\lambda^\dagger = (H^T H + \lambda^2 \boldsymbol{I})^{-1} H^T.$$

Preliminary tests have shown that the algorithm's performance on the test set is almost insensitive on the magnitude of the regularization constant. The Moore-Penrose inverse used for least-squares fit calculation was computed using the ATLAS libraries.

Various hidden-layer sizes, ranging from 10 to 1500, have been tested on the three benchmarks.

B. BLM-specific settings

For the discrete-weights simulation, the whole training set is used during RMSE minimization, without an inline validation. Initial weights have been set in $[-.001, .001]$. Biases are randomly drawn from the same range.

Hidden layer sizes from 5 to 1000, depending on the considered benchmark, have been tested on all datasets. Every run was performed until a local minimum was reached, or the maximum allotted time of 60 seconds expired.

TABLE II
RESULTS FOR THE OPTIMAL HIDDEN LAYER SIZE. EACH FIGURE IS THE OUTCOME OF 50 RUNS, WITH ERRORS ON THE MEAN REPRESENTING 95% CONFIDENCE INTERVALS.

Benchmark	ELM			BLM		
	Time (s)	Hidden layer size	RMSE $\pm 3\sigma$	Time (s)	Hidden layer size	RMSE $\pm 3\sigma$
abalone	.03	200	$.0839 \pm .0009$	16.7	50	$.0803 \pm .0007$
delta_elevators	.05	200	$.0559 \pm .0002$	13.3	40	$.0559 \pm .0004$
house16H	1.61	800	$.0869 \pm .0005$	60.0	1000	$.0763 \pm .0006$
computer_activity	2.81	1500	$.0400 \pm .0005$	60.0	200	$.0342 \pm .0005$

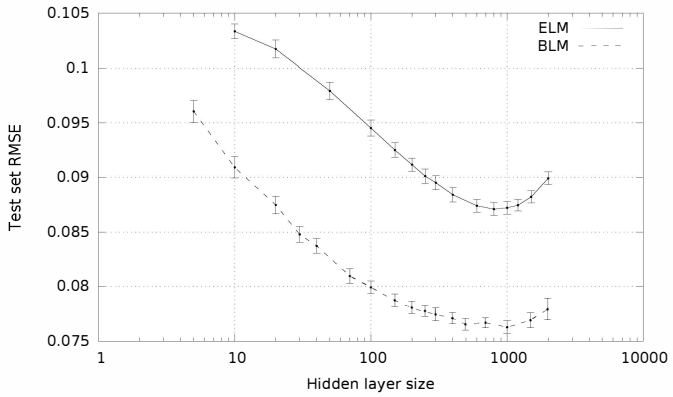

Fig. 10. Comparison between ELM and BLM on the house16H benchmark

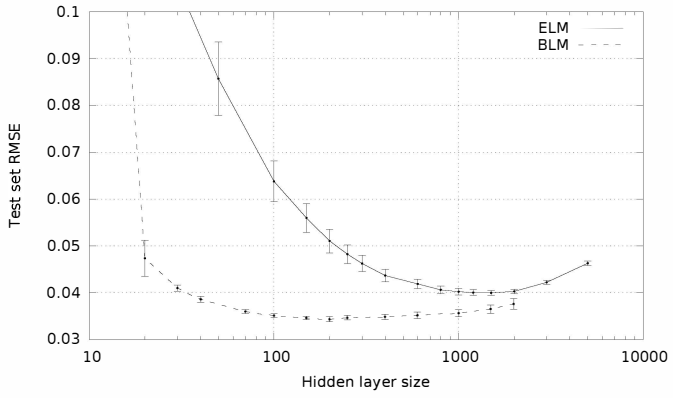

Fig. 11. Comparison between ELM and BLM on the computer_activity benchmark

C. Comparison of performance

Comparison results between BLM and ELM are shown in Figs. 8–10 and in Table II for different numbers of hidden neurons. Each point is the mean of 50 runs, while the error bar shows the 95% confidence interval for the mean value.

BLM outperforms ELM in two of the three considered benchmarks (abalone and house16H), while achieving comparable results in the delta_elevators test. For the abalone benchmark, an additional test has been performed by executing each BLM for 300 seconds with random restarts when a minimum was reached. Although this choice leads to further improvements of the solution quality, we decided to limit all comparisons to the version that halts upon reaching a local minimum.

The number of nodes required by the BLM algorithm is much lower for abalone, delta_elevators and computer_activity; in the house16H case, although the number of hidden neurons required to attain the best result is similar, BLM already outperforms the best ELM result with a much smaller number of hidden nodes.

CPU time is lower for the ELM algorithm, which is based on a matrix pseudo-inversion; however, the BLM time is small enough to be useful in many applications.

TABLE III
RMSE COMPARISON BETWEEN BLM, ELM, AND BP-BASED GAIN ADJUSTMENT TECHNIQUES DISCUSSED IN [13].

Benchmark	BP		ELM	BLM
	($\lambda = 5$)	($\lambda = 10$)		
abalone	.1491	.1501	.0839	.0803
delta_elevators	.0670	.0652	.0559	.0559
house16H	.1090	.1363	.0869	.0763
computer_activity	.2482	.3104	.0400	.0342

Table III compares the ELM and BLM values discussed before with classical backpropagation (BP) results obtained by approximating the threshold function (1) with the variable-gain sigmoid (2) [11]. The reported values were computed by [13] and are reproduced here for completeness.

V. CONCLUSIONS

The original motivation for this study was to revisit basic local search techniques to assess the baseline performance of algorithmic building blocks in order to motivate more complex methods.

The results were surprising. In the considered benchmark cases our BLM algorithm not only reproduced results obtained by more complex methods but actually surpassed them by a statistically significant gap. Also the performance produced by extreme learning was improved, in some cases with much smaller networks (with a smaller number of hidden units). The experimental results indicate that a simple method like BLM based on stochastic local search is fully competitive and produces state-of-the art results when training threshold networks.

The speedup obtained by supporting data structures, incremental evaluations, a small number of bits per weight, and stochastic first-move neighborhood explorations is of about two orders of magnitude for the benchmark problems considered, and increasing with the network dimension. The CPU training time is still much larger than that obtained by ELM via pseudo-inverse, but still acceptable if the application area does not require very fast online training.

While initially motivated by applications to threshold networks, we feel that the promising results in the benchmark

will lead to a much wider applicability of BLM also for more complex networks and machine learning schemes, including networks with smooth transfer functions, and we are continuing the investigation in this direction. Given that the obtained networks tend to be qualitatively different than those obtained with other schemes, with comparable or better performance, *ensembling* is another promising avenue [19].

ACKNOWLEDGMENTS

The research of Roberto Battiti was supported by the Russian Science Foundation, project no. 15–11–30022 "Global optimization, supercomputing computations, and applications".

REFERENCES

[1] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Harvard University, Cambridge, MA, 1974.

[2] D. Rumelhart, G. Hinton, and R. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations*, D. Rumelhart and J. L. McClelland, Eds. MIT Press, 1986.

[3] C. Cortes and V. N. Vapnik, "Support vector networks," *Machine Learning*, vol. 20, no. 3, pp. 1–25, Sep 1995.

[4] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.

[5] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[6] B. Widrow, A. Greenblatt, Y. Kim, and D. Park, "The no-prop algorithm: A new learning algorithm for multilayer neural networks," *Neural Networks*, vol. 37, pp. 182–188, 2013.

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

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

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

[10] E. B. Baum and D. Haussler, "What size net gives valid generalization?" *Neural computation*, vol. 1, no. 1, pp. 151–160, 1989.

[11] D. Toms, "Training binary node feedforward neural networks by back propagation of error," *Electronics letters*, vol. 26, no. 21, pp. 1745–1746, 1990.

[12] R. M. Goodman and Z. Zeng, "A learning algorithm for multi-layer perceptrons with hard-limiting threshold units," in *Neural Networks for Signal Processing [1994] IV. Proceedings of the 1994 IEEE Workshop*. IEEE, 1994, pp. 219–228.

[13] G.-B. Huang, Q.-Y. Zhu, K. Mao, C.-K. Siew, P. Saratchandran, and N. Sundararajan, "Can threshold networks be trained directly?" *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, no. 3, pp. 187–191, 2006.

[14] E. M. Corwin, A. M. Logar, and W. J. Oldham, "An iterative method for training multilayer networks with threshold functions," *IEEE Transactions on Neural Networks*, vol. 5, no. 3, pp. 507–508, 1994.

[15] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for lvcsr using rectified linear units and dropout," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 8609–8613.

[16] H. H. Hoos and T. Stuetzle, *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, 2005.

[17] R. Battiti, M. Brunato, and F. Mascia, *Reactive Search and Intelligent Optimization*, ser. Operations research/Computer Science Interfaces. Springer Verlag, 2008, vol. 45.

[18] R. Battiti and M. Brunato, *The LION way. Machine Learning* plus *Intelligent Optimization*. LIONlab, University of Trento, Italy, 2014.

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