DIMACS Technical Report 95-35
August 1995

Constructive Training Methods for Feedforward Neural Networks with Binary Weights

by

Eddy Mayoraz[1]
RUTCOR–Rutgers University's Center for Operations Research,
P.O. Box 5062, New Brunswick, NJ 08903-5062,
mayoraz@rutcor.rutgers.edu

Frédéric Aviolat
Operations Research, Department of Mathematics,
Swiss Federal Institute of Technology, Lausanne, Switzerland,
aviolat@dma.epfl.ch

[1]First author gratefully acknowledges the support of RUTCOR and DIMACS as well as the Swiss National Science Foundation, grant 20-5637.88.

DIMACS is a cooperative project of Rutgers University, Princeton University, AT&T Bell Laboratories and Bellcore.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

Quantization of the parameters of a Perceptron is a central problem in hardware implementation of neural networks using a numerical technology. A neural model with each weight limited to a small integer range will require little surface of silicon. Moreover, according to Ockham's razor principle, better generalization abilities can be expected from a simpler computational model. The price to pay for these benefits lies in the difficulty to train these kind of networks. This paper proposes essentially two new ideas for constructive training algorithms, and demonstrates their efficiency for the generation of feedforward networks composed of Boolean threshold gates with discrete weights. A proof of the convergence of these algorithms is given. Some numerical experiments have been carried out and the results are presented in terms of the size of the generated networks and of their generalization abilities.

1 Introduction

Artificial neural networks (ANN) are proposed today as alternative solutions for a wide variety of problems. However, in most of the real size applications, the networks are simulated on conventional computers and thus, their inherent parallelism is not exploited. The hardware designer of ANN has to face many constraints, in particular the quantization of the weights (when their storage is based on a numerical technology) and the locality of the connections. In the present study, training of ANN with discrete weights will be investigated.

Many papers already discussed the effect of the quantization of the parameters in neural networks, but they are dedicated to a particular network and a particular training rule, which has been elaborated for models with continuous weights. In fact, most of these studies are devoted to the backpropagation algorithm [DG88, HHP90, AM91, HB91, HH93]. Since this algorithm is essentially a gradient descent, it requires a great precision for the parameters, namely 8 to 10 bits per parameter if the training is done 'off-chip', and 16 bits per parameter otherwise. Conversely, in our approach the discretization level of each parameter is fixed to an arbitrary small value and then, new training methods are designed for this particular model.

A feedforward Boolean neural network realizes a mapping from an input space $\mathcal{I}$ to an output space $\mathcal{O}$. Given an unknown function $\phi : \mathcal{I} \to \mathcal{O}$ and a *task* $T = \{(\boldsymbol{a}^k, b^k = \phi(\boldsymbol{a}^k))\}_{k=1}^{p} \subset \mathcal{I} \times \mathcal{O}$ supplying partial information about ϕ, the goal of the training phase consists in determining a network that computes an extension ψ of T, such that ψ is a good approximation of ϕ. Thus, a feedforward neural network realizes an interpolation of the points given in the task, and we will say that the model ψ built by the network gets a good generalization property if it is close to the target function ϕ, according to a given metric on the set of functions $\{\mathcal{I} \to \mathcal{O}\}$. Lower bounds on $p = |T|$ in order to ensure a good generalization have been derived in [BH89]. Since these bounds grow with the size of the network, a better generalization of a given task T should be achieved by a smaller network. Therefore, the aim of all the constructive training methods is to build small networks realizing the task.

Feedforward neural networks of predetermined architecture suffer from two major drawbacks. On one hand, it is intractable to decide if a given task can be loaded on a given feedforward network [Jud90]. On the other hand, there is no way to determine the most adequate size of the network for a specific application. When training a feedforward network to solve a particular problem, we are always facing the following trade-off: if the network is too large, it is easy to find a configuration such that the network realizes the given task but this solution will overfit the given task and will provide a poor generalization, and in the opposite situation the loading problem is difficult to solve.

A natural way to circumvent these difficulties is to let the training algorithm modify the topology of the network. A variety of training algorithms adapting the size of the network have been proposed. Some of them, called *constructive* algorithms, essentially increase the size of the network until the job is fully performed [MN89, Fre90, GM90, SN90, RCE82], while others start from a large network and try to prune it during the training phase [SD88, WHR90, Ree93]. Finally, other methods combine the two strategies to adapt the size of the network [dBZN94, Def95].

It is not the purpose of this paper to discuss in details the various facets of all these training algorithms remodeling the size of the network; a comparative study based on a wide selection of these methods can be found in [Fie94]. However, we will recall in section 4 the main features of some of these algorithms in order to locate our methods in their context. A formal definition of the neural model considered in this study will be given in section 2. The heuristic technique used to solve the discrete optimization problems arising in each local training phase is briefly described in section 3. Sections 5 and 6 discuss the main constituents of the new training methods proposed, while the results of the numerical experimentations are presented in section 7.

2 Majority functions and majority networks

The neural model considered in the present paper is based on the *perceptron* of Rosenblatt [Ros58], limited to Boolean inputs and output activations. For simplicity in the definition of the majority function we will use the antipodal form $\{-1,+1\}$ instead of the binary form $\{0,1\}$ as a numerical representation for the set of Booleans $\mathbb{B}=\{\text{False},\text{True}\}$.

A function $f:\mathbb{B}^n\to\mathbb{B}$ is a *linear threshold Boolean function* if and only if there exist $\boldsymbol{w}\in\mathbb{R}^n$ and $w_0\in\mathbb{R}$ such that

$$\forall \boldsymbol{b}\in\mathbb{B}^n,\quad f(\boldsymbol{b})=\operatorname{sgn}(w_0+\boldsymbol{w}^\top\boldsymbol{b}), \tag{1}$$

where sgn is the sign function which returns $+1$ if and only if its argument is positive. The vector $\boldsymbol{w}$ is called the *weight vector* of f, w_0 is the *threshold* and its sum with the dot product $\boldsymbol{w}^\top\boldsymbol{b}$ is the *potential* of f for the input $\boldsymbol{b}$.

A *Boolean perceptron* is an n-input-single-output device able to compute any linear threshold Boolean function of n arguments. A task T given by $\{(\boldsymbol{a}^k,b^k)\}_{k=1}^p\subset\mathbb{B}^n\times\mathbb{B}$ is *coherent* if $b^k\neq b^l$ implies $\boldsymbol{a}^k\neq\boldsymbol{a}^l$ for every $k\neq l$, and it is *linearly separable* if and only if it can be computed by a single Boolean perceptron. Many papers are devoted to the computational power of feedforward networks composed of Boolean perceptrons. Clearly, multiplying a weight vector and a threshold by a positive constant will not change a Boolean function f, thus $w_i, i=0,\ldots,n$ can be assumed integers. In order to simplify the hardware realization, some of them limit the model to integer weights and threshold, bounded by a polynomial in n, the number of inputs [HMP$^+$87, SB91]. In this study, we will focus our attention on a subclass of linear threshold Boolean functions with weights limited to the smallest interesting set of values: $\{-1,0,+1\}$. For linear threshold functions with arbitrary weights, the convention for the value of sgn(0) is irrelevant since w_0 can always be chosen such that the potential of f is never 0. In what follows, the only purpose of the threshold w_0 will be to fix this convention. We will take $w_0\in\{-\frac{1}{2},+\frac{1}{2}\}$, thus $f(\boldsymbol{b})=\operatorname{sgn}(w_0)$ for all $\boldsymbol{b}$ orthogonal to $\boldsymbol{w}$, and w_0 is useless when $\|\boldsymbol{w}\|_1$ is odd.

A linear threshold Boolean function defined by a weight vector $\boldsymbol{w} \in \{-1, 0, +1\}^n$ and by a threshold $w_0 \in \{\pm\frac{1}{2}\}$ will be called a *majority function*. A *majority perceptron* is a gate of fan-in n, able to compute any majority function from $\mathbb{B}^n$ to $\mathbb{B}$. The main advantage of our choice for the threshold is that the class of functions computable by a majority perceptron is closed under negation and under duality[1].

A *majority network* is a feedforward Boolean neural network where each node is a majority perceptron and such that the underlying cycle-free graph is simple, *i.e.* a pair of units is connected with at most one edge. Having 0 in the range of the weights is relevant only in the context of training a neural network of a given architecture. Otherwise, when each connection can be maintained or suppressed independently, the value of each weight can be limited to the set $\{-1, +1\}$. A preliminary study has pointed out the interest of the simple computational model provided by the majority networks [May91, May93b]. In the present study, we will concentrate on single output neural networks.

Constructive training methods can basically be decomposed into a global strategy that decides where to introduce a new neuron and which subtask the latter should perform, and a local training technique used to achieve the learning of the specific partial tasks on each new neuron. The problem of training a single majority perceptron has been addressed in [May93a, MR94]. Efficient algorithms have been proposed, either for the maximization of the stability of the perceptron on the task (defined as $\min_{k=1}^{p} \boldsymbol{w}^\top \boldsymbol{a}^k b^k$), or for the minimization of the number of mistakes. Given the success of the discrete optimization tools used in the resolution of these problems, the new algorithms designed for the constructive training problem will all exploit the same heuristic technique known as *tabu search* and briefly presented below.

3 Tabu Search

Tabu search is an efficient meta-strategy used to find good solutions for any kind of optimization problems. It is a *local search* procedure, just as *simulated annealing* or *genetic algorithms* are. A discrete optimization problem is defined by a finite set S of feasible solutions and by a cost function $c : S \to \mathbb{R}$ which has to be minimized. The use of tabu search requires the definition of a set of moves $M \subset \{S \to S\}$, usually assumed closed under inversion. The couple (S, M) can thus be represented as an undirected graph $G = (S, E)$, with $(s, s') \in E$ if and only if $\exists m \in M,\ s' = m(s)$.

Tabu search will proceed by generating a sequence of solutions $s^0, s^1, \ldots$ in S, with s^{k+1} neighbor of s^k in G. At step k, the choice of the neighbor is guided by the best value of c among the neighbors of s^k. To avoid cycling, the most recent moves are stored into a queue called the *tabu list*, and any reverse move of an element of this list is *tabu* and will be forbidden the time the corresponding element remains in the list. Nevertheless, it is possible that, sometimes, a move could be used without danger of cycling, despite its tabu status. For example, when a tabu move leads to a better solution than the best encountered so far,

[1]The dual function f^d of a Boolean function f is defined as $f^d(\boldsymbol{b}) = -f(-\boldsymbol{b})$.

the tabu status will be overridden. The present description of tabu search is summarized and simplified and the reader who needs more information will find it in [Glo89, HdW91].

As far as the training problem of a majority perceptron is concerned, the set of feasible solutions S is clearly $\{0, \pm 1\}^n \times \{\pm \frac{1}{2}\}$. A move will consist either in a small modification of one weight $w_i \leftarrow w_i \pm 1$ assuming that w_i remains in the set $\{0, \pm 1\}$, or in the inversion of the threshold $w_0 \leftarrow -w_0$. The cost function is the key component of tabu search. It is designed specifically for each method and will be detailed in sections 5 and 6.

4 Constructive Methods

There are basically two categories of constructive training algorithms according to the sense of growth of the network. The *forward* methods construct the network by adding new units beyond the existing part of the circuit. Conversely, the *backward* techniques insert new processing units between the input layer and the layer most recently built. The *tiling* algorithm [MN89] and its simplest variant called the tower algorithm [Gal86, Nad89] or the decision tree algorithms [GM90, SN90] are typical examples of forward constructive algorithms, while the construction of the network is backward in the *upstart* method [Fre90].

4.1 Forward methods

In a forward method, the network is built layer by layer from the input to the output. In the present description, we will focus on the case where connections may occur only between two consecutive layers. In this setting, during the construction of a layer $h+1$, only layer h matters, and all previous layers can be ignored. The role of a new layer, say of m units, is the computation of a mapping $\pi : \mathbb{B}^n \rightarrow \mathbb{B}^m$ transforming the previous problem $\{(\boldsymbol{a}^k, b^k)\} \subset \mathbb{B}^n \times \mathbb{B}$ into a new problem $\{(\pi(\boldsymbol{a}^k), b^k)\} \subset \mathbb{B}^m \times \mathbb{B}$, presumably simpler. The new task is then substituted to the old one and the same process is iterated until a linearly separable task is obtained. During the elaboration of a mapping π, $\pi(\boldsymbol{a}^k)$ is called the *internal representation* of $\boldsymbol{a}^k$, and the set of all the $\boldsymbol{a}^l$ with the same image than $\boldsymbol{a}^k$ through π is the *class* of the internal representation $\pi(\boldsymbol{a}^k)$ and is denoted $[\boldsymbol{a}^k]$. A class is *unfaithful* if it contains a pair $\boldsymbol{a}^k, \boldsymbol{a}^l$ with $b^k \neq b^l$. The faithfulness of all the classes defined by π is a necessary condition for the coherence of the new task $\{(\pi(\boldsymbol{a}^k), b^k)\}$.

Each mapping π is elaborated iteratively $(\pi^{(1)}; \pi^{(2)}; \ldots; \pi^{(m)} = \pi)$ by increasing the dimensionality (*i.e.* by adding a new hidden unit), without modifying the existing part: $\pi^{(t+1)} = (\pi^{(t)}, \pi_{t+1}) : \mathbb{B}^n \rightarrow \mathbb{B}^{t+1}$. This process is carried out until all the classes defined by the current mapping are faithful. Different algorithms propose different strategies to achieve this goal. In the *tiling* algorithm, when the mapping $\pi^{(t)} = (\pi_1, \ldots, \pi_t)$ leads to some unfaithful classes, one of them, say $[\boldsymbol{a}^k]$, is chosen arbitrarily, and the new unit computing π_{t+1} is trained with the task $\{(\boldsymbol{a}^l, b^l)\}$, with $\boldsymbol{a}^l \in [\boldsymbol{a}^k]$. Other heuristics have been proposed, such as the *partial task inversion* [AG93], where each new unit takes into account every unfaithful class in a particular way.

4.2 Backward methods

Among all backward constructive methods, one distinguishes between the ones which construct a single hidden layer, from the others, and for the needs of this paper, we will concentrate on the former category [Fre90, AG93]. These backward methods construct their unique hidden layer in the same way a forward method does for each layer, only the stopping criterion is different. The iterative process building the mappings $\pi^{(0)}, \pi^{(1)}, \ldots$ goes until the new task is linearly separable, instead of halting when all the classes of the current mapping are faithful. The initial mapping $\pi^{(0)} : \mathbb{B}^n \to \mathbb{B}^{m_0}$ can either be considered as the identity ($m_0 = n$) when the output unit is connected to the inputs, or the empty mapping ($m_0 = 0$) when no jumping links connect the inputs with the output.

The methods mentioned above (see [Fre90, AG93]) are backward, since formally, the output unit is introduced first, and then the hidden layer is elaborated. At each iteration t, the current set $\{v^k\}$ of potentials at the output unit is computed for every input $\boldsymbol{a}^k$. In these algorithms, the binary representation $\{0, 1\}$ is usually used for the set of Boolean values. Thus, the introduction of a new hidden unit computing $\pi_{t+1} : \mathbb{B}^n \to \{0, 1\}$ will modify the values $\{v^k\}$ only for the subset of points $\boldsymbol{a}^k$ for which $\pi_{t+1}(\boldsymbol{a}^k) = 1$, since $v^k = w_0 + \boldsymbol{w}^\top \pi^{(t)}(\boldsymbol{a}^k) + w_{t+1}\pi_{t+1}(\boldsymbol{a}^k)$. The construction is complete whenever $v^k > 0$ if and only if $b^k = 1$. Various existing algorithms of this nature propose different clever heuristics to choose the subset of points which will be modified at each step (*e.g. shift* algorithm [AG93]).

To summarize, forward as well as backward approaches construct sequences of transformations of the problem, in order to simplify it until it is solvable by a single unit. These transformations are based on considerations done beyond the non-linear functions *sgn* in forward methods, while in backward techniques the set of potentials before the non-linearity of the output unit controls the construction of the network.

5 Forward Construction of Majority Networks

Using the existing local learning algorithms for minimizing the number of mistakes in a majority perceptron (see [May93a]), classical forward constructive methods such as the *tiling* algorithm could be applied in an almost straightforward way to the construction of majority networks. However, in this research we intend to go beyond this simple adaptation by improving substantially the constructive technique.

5.1 Ideal criterion for faithfulness

Existing forward approaches such as the *tiling* or the *partial task inversion* algorithms are not optimizing the ideal criterion at each step. The main cause for this is that, in order to use always the same local algorithm, the local problem assigned to each new unit has to be of the form:

Form 1: find a linear threshold function minimizing the number of mistakes in a task $T = \{(\boldsymbol{a}^k, b^k)\}_{k=1}^p$, or in other words find a linear threshold function separating in a best way two sets of points T_+ and T_-, where $T = T_+ \times \{+1\} \cup T_- \times \{-1\}$.

If the form of this goal is adequate for the first unit in a layer, it is inappropriate for the subsequent units. Assume that t units have already been introduced in a new layer, and that they provide a mapping $\pi^{(t)}$ with several unfaithful classes $[\boldsymbol{a}^{k_1}], \ldots, [\boldsymbol{a}^{k_u}]$. Let $T_+^{k_i}$ (resp. $T_-^{k_i}$) denote the sets of points of the class $[\boldsymbol{a}^{k_i}]$ with a target output $+1$ (resp. -1) for $i = 1, \ldots, u$. To reach complete faithfulness as quickly as possible, the ideal criterion for a new unit computing π_{t+1} would be to separate $T_+^{k_i}$ from $T_-^{k_i}$ for all i. However, this should be done without imposing that $\pi_{t+1}(T_+^{k_i}) \neq \pi_{t+1}(T_-^{k_j})$ for $i \neq j$, since the internal representation of the points in $T_+^{k_i}$ differ already from that of the points in $T_-^{k_j}$, for $i \neq j$. So, the general form of the local problem that has to be solved by each new unit is:

Form 2: given a collection of pairs of disjoint sets $\{(T_+^{k_i}, T_-^{k_i})\}_{i=1}^u$, find a linear threshold function separating in a best way each pair independently.

In the context of real weights, a goal of form 2 is more difficult to address than one of form 1, since the objective function cannot be optimized using a gradient descent technique as for example the well known perceptron algorithm does ([Ros58, DH73]). On the contrary, when a local search algorithm is used, there is a lot of flexibility in the form of the objective function, and we are going to exploit this freedom to optimize at each step the ideal goal given by the form 2 and formally described as follows.

Let $o^k = \pi_{t+1}(\boldsymbol{a}^k)$ denote the output of the current output unit for the input $\boldsymbol{a}^k$ and $\mathcal{F}$ the set of faithful classes. For each class $[\boldsymbol{a}^k]$, ε^k is the number of mistakes in the class $[\boldsymbol{a}^k]$ made by the current unit

$$\varepsilon^k \;=\; \left|\{\boldsymbol{a}^l \in [\boldsymbol{a}^k] \;\mid\; o^l \neq b^l\}\right|.$$

The measure of the quality of the separation of the class $[\boldsymbol{a}^k]$ is given by $\min(\varepsilon^k, |[\boldsymbol{a}^k]| - \varepsilon^k) = \mu^k$. Indeed, if all the elements of the class are misclassified, it also means that the separation is optimal, since in the seek of faithfulness, the orientation of the separator does not matter. The cost function will be:

$$c_1 \;=\; \sum_{[\boldsymbol{a}^k] \notin \mathcal{F}} \mu^k. \tag{2}$$

Clearly, $c_1 = 0$ means that with the current unit, all the classes are faithful and thus, the construction of the current layer is complete.

5.2 Short or narrow networks ?

The cost function given by (2) corresponds to the ideal local goal as far as complete faithfulness is concerned. Nevertheless, in a more global perspective, it is difficult to discern the best goal that a unit should reach at a given time. When the construction of one layer is achieved, each point of the next task corresponds to one class of one internal representation. Thus, even if the main goal of each unit is to increase the faithfulness of the current classes, a solution which does not shatter the faithful classes will be preferred since it will lead to a smaller task for the next layer. If attention is paid exclusively to the reduction of the faithfulness, then each layer will be small, but the task of the next layer might be harder to solve, since it consists of a large number of points in a low dimensional space. On the other hand, more units will be used on one layer when a lot of care is taken to avoid shattering the classes, but the next task will probably be easier, since it will be of smaller size and in a larger dimensional space. This is a trade-off between deep and narrow networks against short and wide ones; or "time against space" in terms of computational resources.

Let γ^k denote the minimum number of points in the class $[\boldsymbol{a}^k]$ of output $+1$ or -1 at the new unit:

$$\gamma^k \;=\; \min\big(\;|\{\boldsymbol{a}^l \in [\boldsymbol{a}^k] \;\mid\; o^l = +1\}|\;,\quad |\{\boldsymbol{a}^l \in [\boldsymbol{a}^k] \;\mid\; o^l = -1\}|\;\big)\,.$$

If a faithful class $[\boldsymbol{a}^k]$ is not divided, $\gamma^k = 0$, and the worst case occurs when a faithful class is divided into two pieces of the same size. γ defined as the sum of these values over all the faithful classes, $\gamma = \sum_{[\boldsymbol{a}^k]\in\mathcal{F}} \gamma^k$, measures the shattering of the faithful classes. This parameter might be aggregated in the cost function which becomes:

$$c_2 \;=\; \omega_1 c_1 \;+\; \omega_2\gamma, \tag{3}$$

where ω_i are positive weightings that give relative importance to each of the two elements of the function. More sophisticated objective functions have been investigated and their description can be found in [Avi93], but we will not discuss this approach in more details here.

5.3 Convergence

Classically, the convergence proof for forward constructive methods is decomposed into two steps: the *vertical convergence*, which ensures the termination of the construction of each layer; and the *horizontal convergence* which refers to the fact that at one point, the new task $\{(\pi(\boldsymbol{a}^k), b^k)\}$ will be linearly separable.

> **Lemma 5.1** The minimization at each new unit of the cost function c_1 of equation (2) ensures the vertical convergence.

> **Proof**: Observe that for any two distinct points $\boldsymbol{a}^k, \boldsymbol{a}^l \in \mathbb{B}^n$, it is always possible to find a majority function f such that $f(\boldsymbol{a}^k) \neq f(\boldsymbol{a}^l)$. For this, it suffices to choose $w_i = a_i^k$ for some i such that $a_i^k \neq a_i^l$ and $w_i = 0$ for all i such that $a_i^k = a_i^l$. Therefore, there is always a way to adjust the new majority unit such that its introduction decreases strictly the quantity δ_π defined as the number of pairs of points $(\boldsymbol{a}^k, \boldsymbol{a}^l)$ such that $\pi(\boldsymbol{a}^k) = \pi(\boldsymbol{a}^l)$ — both points are in the same class—,

and $b^k = +1$, $b^l = -1$. Clearly, $\delta_\pi = 0$ if and only if all the classes defined by π are faithful. Moreover, any solution that have a non maximal cost function c_1 (*i.e.* any solutions except the worst) will lead to a strictly smaller δ_π. $\triangle$

The cost function c_2 presented in (3) however, may not have this property, particularly when ω_2/ω_1 is big. Therefore, it is safe to place a barrier on the worst possible value of c_1 when c_2 is used. We are going to place another barrier on an event that will very unlikely occur, but which would compromise the horizontal convergence. So, the complete cost function becomes:

$$c_2 = \begin{cases} +\infty & \text{if none of the unfaithful classes are divided} \\ +\infty & \left| \begin{array}{l} \text{if there will be no faithful class of size at least 2} \\ \text{and no unfaithful class of size at least 3 left} \end{array} \right. \\ \omega_1 c_1 + \omega_2 \gamma & \text{otherwise} \end{cases} \tag{4}$$

Proposition 5.2 The minimization at each new unit of the cost function c_2 defined in (4) ensures the global convergence.

Proof: The first barrier in c_2 ensures that a solution dividing no unfaithful class will never be chosen, so the argument used in the proof of lemma 5.1 works and the vertical convergence is guaranteed. When all the classes are faithful, the new task built on the mapping π will be smaller only if at least one class contains more than one point, but that is precisely what the second barrier aims at. If at each layer, the new class is strictly smaller, the process will obviously terminate. To complete the proof, we have to show that there is always a solution of value $< +\infty$.

Let call P the property stating that there is either one unfaithful class of size at least 3, or one faithful class of size at least 2, or both. Before the introduction of the first unit in a layer, there is only one class which is unfaithful (if the problem is not trivial). If this class has only two elements, the problem is easy, since two points can always be separated by one majority perceptron. So, we will assume that P is initially verified and we will show that in any case, there is a majority perceptron dividing at least one unfaithful class, while keeping the property P.

If the construction of the layer is not complete, there is at least one unfaithful class, so let $\boldsymbol{a}^1$ and $\boldsymbol{a}^2$ be two points of the same class but with $b^1 \neq b^2$. Since P holds, there exists two distinct points $\boldsymbol{a}^3$ and $\boldsymbol{a}^4$ in a same class with $b^3 = b^4$. Note however, that one of the first two points may be identical to one of the last two. It remains to show that there is a majority perceptron separating $\boldsymbol{a}^1$ from $\boldsymbol{a}^2$ while keeping $\boldsymbol{a}^3$ and $\boldsymbol{a}^4$ together.

Let I and J be the two non-empty set of indices defined as

$$I = \{i \mid a_i^1 \neq a_i^2\}, \quad J = \{i \mid a_i^3 \neq a_i^4\}.$$

Case 1: $I \not\subset J$. This case can be solved by setting all weights to 0, except one of index in $I \backslash J$.

Case 2: $I \subsetneq J$. Take $i \in I$ and $j \in J \backslash I$, set $w_i = +1$ and $w_k = 0$, $\forall k \neq j$. If $a_i^3 = a_j^3$, $w_j = -1$, otherwise $w_j = +1$, so that the potentials of $\boldsymbol{a}^3$ and $\boldsymbol{a}^4$ are both 0. Since $j \notin I$, $a_j^1 = a_j^2$, and the potential of $\boldsymbol{a}^1$ and $\boldsymbol{a}^2$ are 0 and ± 2. An adequate choice of threshold will separate these two points.

Case 3: $I = J$ and $\exists i, j \in I$ such that $a_i^1 - a_i^2 = a_i^3 - a_i^4$ and $a_j^1 - a_j^2 = a_j^4 - a_j^3$. This case is solved by setting to 0 all the weights except w_i and w_j, which will take the same value if $a_i^1 = a_j^1$, and opposite ones otherwise.

Case 4: $I = J$ and $\forall i \in I$ let say (w.l.o.g.) $a_i^1 = a_i^3 = -a_i^2 = -a_i^4$. Then there is $j \notin I$ such that $a_j^1 \neq a_j^3$, otherwise we have $\boldsymbol{a}^1 = \boldsymbol{a}^3$ and $\boldsymbol{a}^2 = \boldsymbol{a}^4$ which contradicts $b^1 \neq b^2$ and $b^3 = b^4$. By vanishing all the weights but w_j and w_i for one $i \in I$ and by choosing $w_j = a_j^1$, the potentials for $\boldsymbol{a}^1$ and $\boldsymbol{a}^2$ will be 0 and $+2$, while these for $\boldsymbol{a}^3$ and $\boldsymbol{a}^4$ will be 0 and -2. A threshold of $-\frac{1}{2}$ will solve the problem.

$\triangle$

This proof of convergence is very rough since it leads to generous upper bounds such as $O(p^2)$ units per layer and $O(p)$ layers, where p denotes the size of the task. It was not in the scope of this research to improve these bounds, and the numerical results will clearly show that they are largely over-estimated.

6 Back-Forth Constructive Method

Principles of backward methods are difficult to use with a bipolar representation $\{-1, +1\}$ for the set of Boolean values since the value of every potential v^k is moving up or down when a new unit is inserted. However, in this section we will see how the ideas of backward methods can be used to improve forward constructions.

6.1 Back-Forth is backward

As discussed in section 5, by adding a unit in a layer, we want to get internal representations as faithful as possible and we would like the next task to be not too difficult. Another way to reach these objectives is to consider, during the training of a new unit, the set of potentials at the first unit which will be placed on the next layer, as it is done in backward methods. Even if this idea can be extended to general linear threshold functions, ternary weights are particularly convenient for this purpose. Actually, when a unit u_1^L is first introduced on a new layer L, if it does not manage to completely realize the task, a supplementary layer will be necessary. So, the first unit u_1^{L+1} in next layer $L+1$ can already be introduced and connected to the unit in layer L with a weight of value $+1$ without loss of generality.

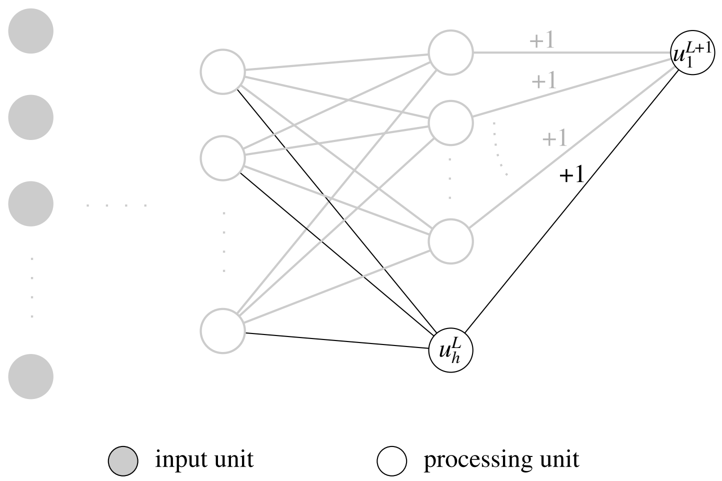

Figure 1: Introduction of a new unit updated by the *back-forth* training method. The new unit is connected to the first unit of next layer with a weight of value +1. The rest of the network (in gray) is unchanged during the training of the the new unit.

Let v_t^k denote the potentials at unit u_1^{L+1} after the introduction of t units in layer L realizing a mapping $\pi^{(t)} = (\pi_1, \ldots, \pi_t)$:

$$\begin{aligned} v_t^k &= \sum_{s=1}^{t} \pi_s(\boldsymbol{a}^k) \\ v_0^k &= 0. \end{aligned}$$

Thus the potentials v_t^k are calculated by temporarily setting all the weights between the current layer L and the first unit u_1^{L+1} of the next layer to $+1$.

The set of potentials at the first unit in layer $L+1$ is now used to guide the update of the new unit (figure 1). The problem is similar to the training of a single unit, except that the update of the weights of a unit in one layer depends on the potentials at a unit in the next layer. Following the objective function used in the well known "perceptron algorithm" for minimizing the number of mistakes in a task (see [Ros58, DH73]), our local search procedure will minimize the cost function $c(\boldsymbol{w}, w_0)$ defined in (5) when applied to the $(t+1)^{\text{th}}$ unit in layer L

$$c(\boldsymbol{w}, w_0) = -\sum_{k \text{ misc.}} v_{t+1}^k b^k = -\sum_{k \text{ misc.}} (v_t^k + \operatorname{sgn}(w_0 + \boldsymbol{w}^\top \boldsymbol{a}^k)) b^k. \tag{5}$$

where "misc." refers to the output state of the first unit of layer $L+1$ after the introduction of $t+1$ units in layer L. In order to distinguish between "strongly" and "weakly" misclassified points, we are introducing a cost function of a more general form:

$$c(\boldsymbol{w}, w_0) = \sum_{k=1}^{p} P\left((v_t^k + \operatorname{sgn}(w_0 + \boldsymbol{w}^\top \boldsymbol{a}^k)) b^k\right), \tag{6}$$

where P is a penalty function from $\mathbb{Z}$ to $\mathbb{R}$. Note that this form allows also to consider correctly classified points in the objective function. However, in this research we only experimented penalty functions of the form:

$$P(x) = \begin{cases} (1-x)^d & \text{if} \quad x \leq 0 \\ 0 & \text{if} \quad x > 0 \end{cases} . \tag{7}$$

If $d = 0$, the cost function simply count the number of mistakes and will be referred to as the *constant penalty* cost function. The cost function of equation (5) is obtained from equation (6) by using a *linear penalty*, *i.e.* by setting $d = 1$ in (7). Some experiments have also been carried out with a *quadratic penalty* ($d = 2$).

6.2 Back-Forth is forward

In [May93b] it has been shown that any Boolean function can be computed by a majority network of depth 2. So, in principle a single hidden layer is always sufficient. However, there is no certainty that after adding sufficiently many units on a hidden layer, each of which having been designed to minimize a cost function of the form (6) and then kept up while further units are added, there will finally be zero errors in the first unit of the next layer. Therefore, this *back-forth* algorithm will still construct networks of several layers. As before, the stopping criterion for the construction of one layer is the faithfulness of all the classes. Finally, to ensure the vertical and the horizontal convergence, the two barriers introduced in the cost function c_2 in (4) are maintained, and the convergence proof is the same as in the previous section.

6.3 Local or not local ?

A very important feature in the constructive algorithms mentioned in this work is the locality of the training. A global strategy guides the construction by deciding when and where a new unit is added and what task it has to solve; but this task is solved locally on the new unit and once this is done, the parameters of that unit will never be reconsidered. If locality has the advantage of simplicity, it certainly restricts the training, and very likely, some global training algorithms will lead to smaller majority networks with probably higher generalization abilities.

Due once again to the flexibility of the optimization technique used, any of the algorithms presented in this paper can be used to update several units at a time. In the following section, beside the algorithms presented above, we also experimented one version of the *back-forth* method where two units are trained at the same time. The training of the first unit is done as before. At each further step, a simple heuristic determines which of the units in the current layer is the least helpful, and this unit is trained again with a newly inserted unit. Even if this is only a "small violation" of the locality rule, in some cases it improves significantly the generalization results.

7 Numerical Experiments

Many numerical experiments were carried out to test the performances of the algorithms presented in this work. In all our experiments, the tabu list length has been fixed to 5. Moreover, the training of each new unit stops whenever the condition for the vertical convergence is fulfilled (all the classes are faithful) or when at least 500 iterations have been done and there was no improvement during the last 50 iterations.

The first series of tests concerns the ability of our methods to construct majority networks capable of implementing exactly a given Boolean function. The second series of experiments will regard the generalization performances of the networks built with our algorithms. Results will be compared to those obtained with the classical constructive algorithms, such as the *tiling* algorithm, the *partial task inversion* algorithm and the *shift* algorithm [AG93].

7.1 Synthesis of Boolean functions

Let f be a Boolean function $\mathbb{B}^n \to \mathbb{B}$. We consider tasks of the form $T = \{(\boldsymbol{a^k}, f(\boldsymbol{a^k})) \mid \boldsymbol{a^k} \in \mathbb{B}^n\}$, containing all the examples of the known function f. The purpose of this first series of tests on complete tasks is to evaluate the size of constructed networks realizing exactly the given function f.

Several quantities are of interest for measuring the size of a network. These are the number of layers, the number of neurons, and the number of connections. In the framework of majority networks, we will consider a connection as non-existent if its weight is zero.

The first experiment was made on *RANDOM* functions. The output is chosen randomly to be $+1$ or -1, with the same probability, for each input vector. The required size of the networks able to realize such tasks is a measure of the ability of the different algorithms to memorize information in a compact way. Figure 2 shows the average sizes of the obtained networks, over 10 runs, with input size ranging from 2 to 8. Performances of our different algorithms are compared to each other and to a simple adaptation of the *tiling* algorithm to majority networks, referred to as "Tiling (Majority)". In the last figure, we compare our two best algorithms to the *tiling*, the *partial task inversion* and the *shift* which build networks of linear threshold Boolean units. In all figures, "Basic" refers to the algorithm of section 5, with weighting in equation (3) chosen as $\omega_1 = 100$ and $\omega_2 = 1$, while "Back-forth" refers to the methods of section 6, with the kind of cost function specified in brackets.

The size of the networks grows exponentially with the input size, which is what could be expected since there is no structure in a random function. We observe that the simple adaptation of the *tiling* algorithm to majority networks builds deeper networks, whereas the several *back-forth* approaches give networks with fewer layers. It appears clearly that the "quadratic penalty" function ($d = 2$ in (7)) is superior to the "linear penalty" function of ($d = 1$ in (7)). It could also be expected that optimizing two neurons together still improves the results. Comparing our best algorithms with those for continuous weighted units, we observe that their performances are fair and even better than the *tiling* algorithm. In [GM90], some similar experiments have been carried out using decision tree algorithms to construct

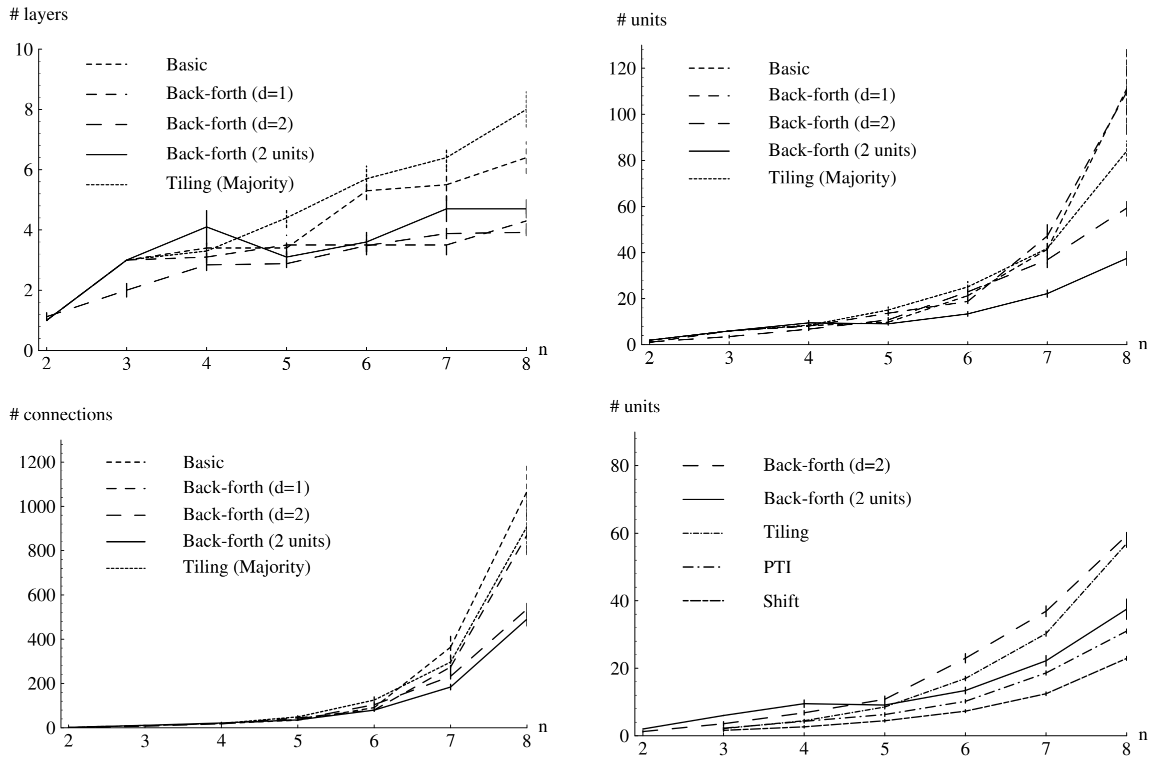

Figure 2: Construction of *RANDOM* functions. Average size of the networks built on complete tasks versus the input size n.

networks with continuous weights. For complete random tasks of size $n = 6$, the authors report an average number of 20.5 ± 3.9 units over 100 runs, which lies between the *tiling* method (16.9 ± 3.6) and the simple *back-forth* with $d = 2$ (22.9 ± 1.4).

To test the majority implementations of classical Boolean functions, experiments were made on the *PARITY* function, defined as $f(\boldsymbol{x}) = \Pi_i x_i$. The output value is $+1$ if and only if the number of -1 in $\boldsymbol{x}$ is even. The other function we implemented with our constructive algorithms is the *COMPARISON* function. Consider an input vector $\boldsymbol{x} \in \mathbb{B}^n$ (n even) written as $\boldsymbol{x} = (\boldsymbol{x}^1, \boldsymbol{x}^2)$ with $\boldsymbol{x}^1, \boldsymbol{x}^2 \in \mathbb{B}^{n/2}$. Then *COMPARISON* can be defined as $f(\boldsymbol{x}) = +1$ if and only if $\sum_{i=1}^{n/2}(x_i^1 + 1)^{i-2} < \sum_{i=1}^{n/2}(x_i^2 + 1)^{i-2}$, that is if the number with binary representation $\boldsymbol{x}^1$ is smaller than the number with binary representation $\boldsymbol{x}^2$. It is worth noting that *COMPARISON* is a linearly separable function that requires integer weights growing exponentially in n. It has been shown however that a depth 2 and polynomial size majority network can compute *COMPARISON* [AB91].

Figures 3 and 4 show the average size of the networks produced by 10 runs versus the input size n for complete tasks.

For small input sizes, the algorithms constructed the smallest known majority networks able to compute the *PARITY* function exactly [May91], except the abnormality of the method optimizing 2 neurons, for $n = 3$. This is due to the fact that, in this particular

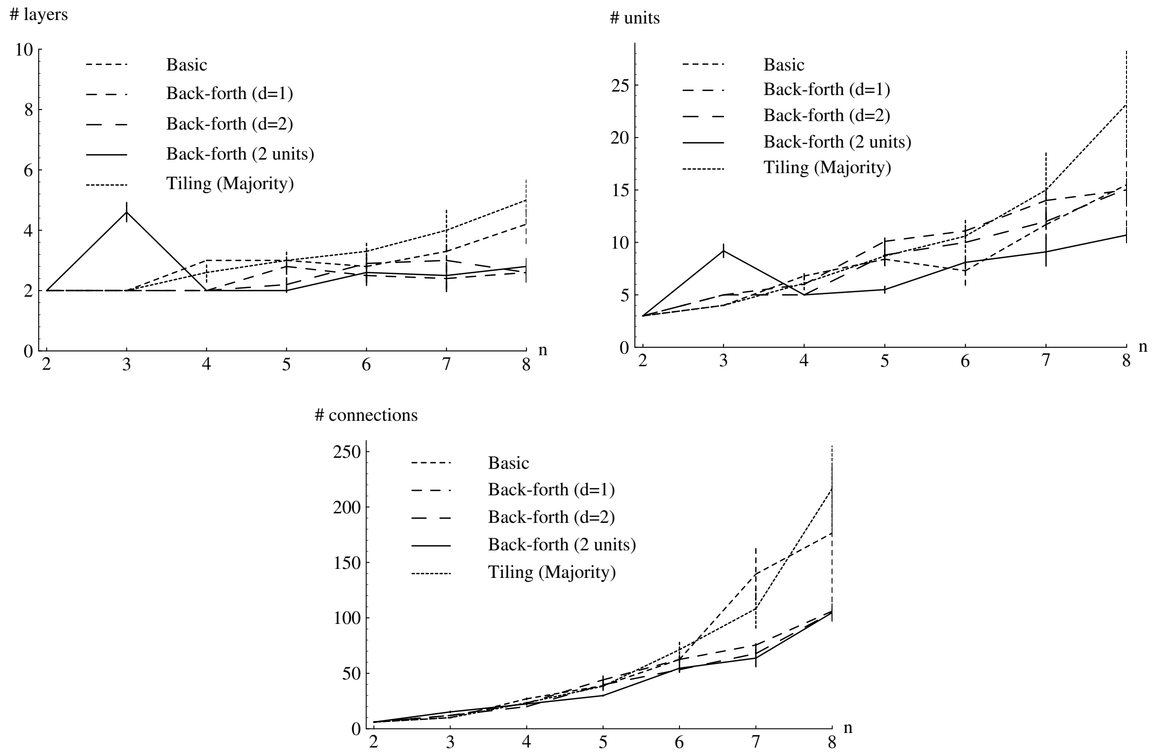

Figure 3: Construction of *PARITY* functions. Average size of the networks built on complete tasks versus the input size n.

case, reoptimizing a neuron does more harm than good; instead of realizing the function in 2 layers, the algorithm reduces the task (in 2 layers) to the *PARITY* with input size 2, and then it needs 2 more layers. As before both simpler algorithms ("Basic" and "Tiling (Majority)") generally build larger networks.

The networks obtained for the *COMPARISON* function are very small, for all three algorithms, due to the simplicity of the function. It is interesting to see that such a computational kind of function can be efficiently implemented by majority networks.

7.2 Generalization

We now present numerical experiments made to test the generalization ability of the constructed networks. As described in section 1, we consider a task $T = \{(\boldsymbol{a}^k, b^k = \phi(\boldsymbol{a}^k))\}_{k=1}^p \subset \mathbb{B}^n \times \mathbb{B}$ supplying partial information on an unknown function ϕ. The network will try to extract the most information, so as to be able to approximate ϕ as well as possible.

Our experiments were made with the *2-CLUMPS* function. A *clump* is a sequence of consecutive $+1$ in a vector, considered cyclically. This function will output $+1$ when the input vector contains 2 or more clumps. It can be formally defined as $f(\boldsymbol{x}) = +1$ if and only if $\left|\{i \mid x_i = +1 = -x_{(i \bmod n)+1}\}\right| \geq 2$. Networks were built for an input size $n = 25$

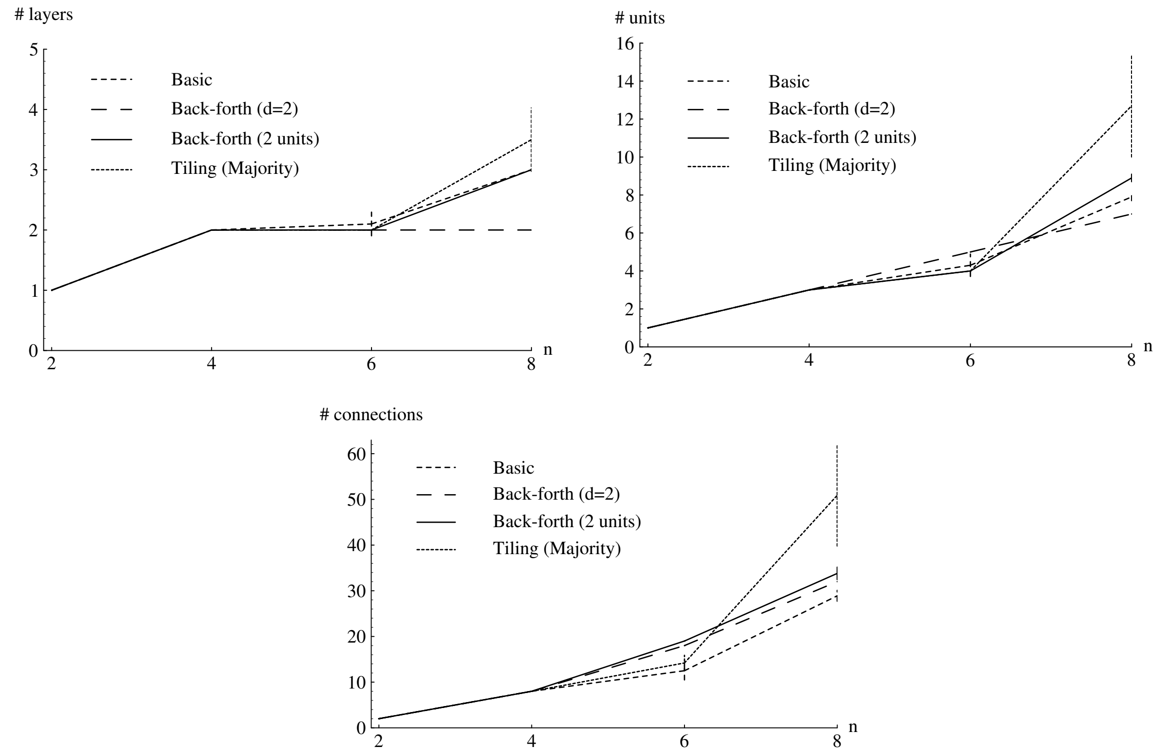

Figure 4: Construction of *COMPARISON* functions. Average size of the networks built on complete tasks versus the input size n.

and trained on sets of p random points, with p ranging from 100 to 800. Their performances were evaluated over test sets of the same size. Figure 5 shows the average size of the obtained networks and the average percentage of incorrect classifications, over 25 trainings. Performances of the *tiling*, the *partial task inversion* and the *shift* algorithms are also plotted.
The second function we used to test generalization is the *3-SIMILARITY* function (proposed in [AG93]). The input vector is partitioned into two pieces, and the output of the function will be $+1$ when at most 3 corresponding components of the two pieces differ. It can be formally defined as $f(\boldsymbol{x}) = +1$ if and only if $\left|\{i \mid x_i \neq x_{\frac{n}{2}+i}\}\right| \leq 3$.
The third function is the *COMPARISON* function, defined in section 7.1. For both of these functions, an input size $n = 20$ was chosen and training was performed on sets of p random points, with p ranging from 100 to 800. Generalization was evaluated over test sets of the same size. Figures 6 and 7 show the average percentage of incorrect classifications, over 25 trainings, as well as the average number of units in the constructed networks.
Generalization of the *2-CLUMPS* function is very good, very much like the *shift* algorithm and much better than the *tiling* algorithm. Of course, our networks are bigger, because a majority unit contains less information than a real-weighted linear threshold Boolean function. Results of the *3-SIMILARITY* function are fair, particularly for the algorithm optimizing 2 units at the same time, which produces small networks with a good generalization rate.

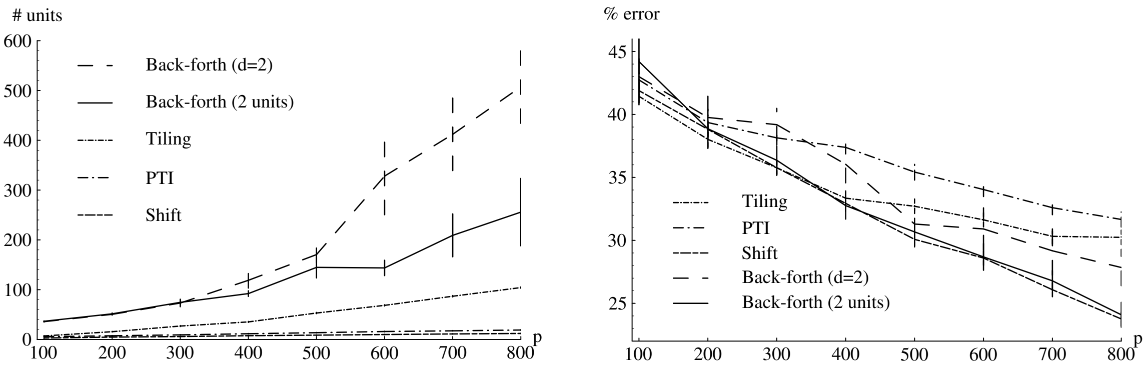

Figure 5: Generalization of *2-CLUMPS*. Average size and percentage of errors made by networks trained on p randomly chosen examples in $\mathbb{B}^{25}$.

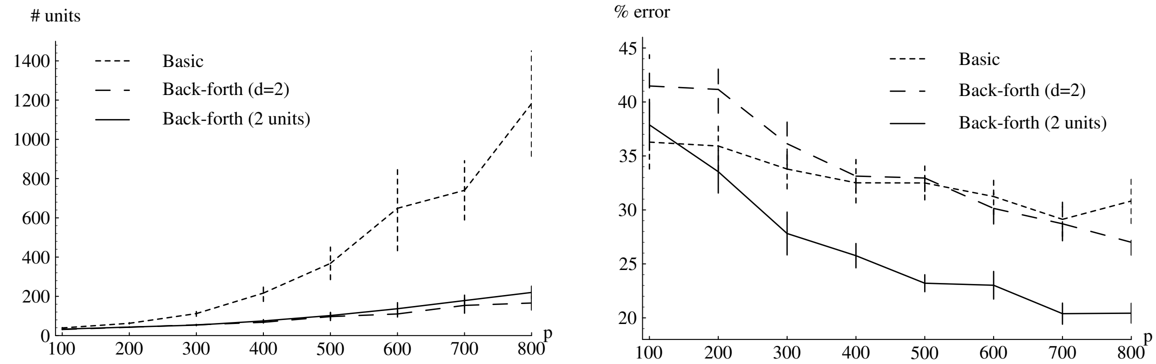

Figure 6: Generalization of *3-SIMILARITY*. Average size and percentage of errors made by networks trained on p randomly chosen examples in $\mathbb{B}^{20}$.

Finally, our networks were able to learn very well the *COMPARISON* function, with all three algorithms. The error rate is smaller than 5%, even with $p = 400$ examples. This corresponds only to 0.04% of the total number of input vectors in $\mathbb{B}^{20}$. It can also be observed that the sizes of the networks were rather small. Surprisingly, in this case generalization is achieved by the simplest algorithm described in section 5.

These generalization tests were also tried with the simple adaptation of the *tiling* algorithm to majority networks. This algorithm generally built huge networks, and sometimes it did not even converge. Indeed, this algorithm does not have any convergence guarantee, unlike the others that we designed.

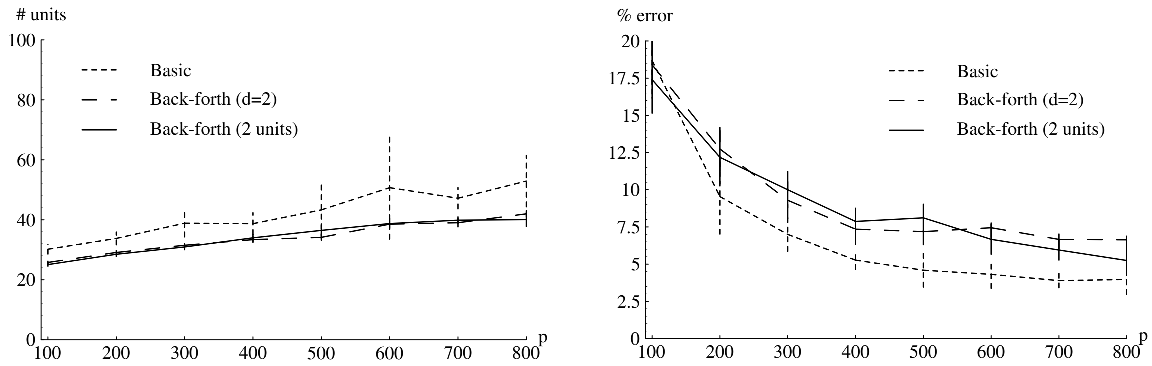

Figure 7: Generalization of *COMPARISON*. Average size and percentage of errors made by networks trained on p randomly chosen examples in $\mathbb{B}^{20}$.

8 Conclusions

Training feedforward neural networks is a difficult problem and it becomes even harder when the weights are limited to integer values. Constructive training techniques are gaining interest since they avoid the problem of dimensioning the network. We proposed two new heuristics for the construction of feedforward networks. The first one is of a forward construction type (such as the *tiling* algorithm) but the novelty is the criterion optimized at each step, which is designed to build layers as narrow as possible. The second idea takes advantages of both forward and backward approaches. Simple convergence proofs are given for these two methods. Though these are general ideas which can be used to train feedforward networks with real weights, we applied them for the construction of majority networks, *i.e.* feedforward Boolean networks with ternary weights in $\{-1, 0, +1\}$. Numerical experiments are presented and it is encouraging to see that, even if majority networks provide a quite restricted computational model, it holds the comparison with classical networks.

A trade-off between local and global learning algorithms is conceivable where a constructive algorithm inserts or updates more than one unit at the same time. The number of units introduced simultaneously should not be too high since their update would become too complex, but it appeared to be easy to train two units concurrently. Furthermore, this extension gave very good results on all our experimented problems.

The good generalization performances reached for *COMPARISON* demonstrate that a model with only Boolean parameters is also able to realize Boolean functions intrinsically based on integers coded using a binary representation. Moreover, for exhaustive *PARITY* and *COMPARISON* tasks, the best known constructions for small n's have always been found by our constructive training algorithm. This allows us to imagine that such an algorithm could be a useful tool in the search for new constructions of other important Boolean functions.

We proved in [May93b] that any Boolean function f can be computed by a majority network with a single hidden layer. However, we were not able to find in the present work a constructive algorithm with convergence guarantees and restricted to one hidden layer. For example, we unsuccessfully looked for a global energy function which measures the performance of a single hidden layer network and which will strictly decrease at each introduction of an appropriate unit on the hidden layer.

References

[AB91] Noga Alon and Jehoshua Bruck. Explicit constructions of depth-2 majority circuits for comparison and addition. Technical Report RJ 8300 (75661), IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, CA 95120-6099, August 1991.

[AG93] Edoardo Amaldi and Bertrand Guenin. Constructive methods for designing compact feedforward networks of threshold units. Technical report, Swiss Federal Institute of Technology, Department of Mathematics, 1993.

[AM91] K. Asanovic and N. Morgan. Experimental determination of precision requirements for back-propagation training of artificial neural networks. In *Proceedings of 2nd International Conference on Microelectronics for Neural Networks*, pages 9–15, Munich, 1991.

[Avi93] Frédéric Aviolat. Les réseaux de neurones artificiels multicouches à poids binaires: étude de complexité et algorithmes constructifs. Master's thesis, École Polytechnique Fédérale de Lausanne, Département de Mathématiques, March 1993.

[BH89] Eric B. Baum and David Haussler. What size net gives valid generalization ? *Neural Computation*, 1(1):151–160, 1989.

[dBZN94] F. d'Alché Buc, D. Zwierski, and J.-P. Nadal. Trio learning: a new strategy for building hybrid neural trees. *Int. Journ. of Neural Systems*, 5:259–274, 1994.

[Def95] Guillaume Deffuant. An algorithm for building regularized piecewise linear discrimination surfaces: The perceptron membrane. *Neural Computation*, 7(2):380–398, March 1995.

[DG88] R. M. Debenham and S. C. J. Garth. Investigation into the effect of numerical resolution on the performance of back-propagation. In *Neural Networks from Models to Applications, Proceedings of n'Euro 88*, pages 752–755, Paris, 1988.

[DH73] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, New York, 1973.

[Fie94] Emile Fiesler. Comparative bibliography of ontogenic neural networks. In Maria Marinaro and Pietro G. Morasso, editors, *Proceedings of the International Conference on Artificial Neural Networks (ICANN 94)*, volume 1, pages 793–796, London, U.K., 1994. Springer-Verlag.

[Fre90] Marcus Frean. The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation*, 2(2):198–209, 1990.

[Gal86] Stephen I. Gallant. Three constructive algorithms for network learning. In *Proceedings of the 8th Annual Conference of Cognitive Science Society*, pages 652–660, Amherst, MA, August 1986.

[Glo89] Fred Glover. "tabu search" part I. *ORSA J. Computing*, 1(3):190–206, 1989.

[GM90] M. Golea and M. Marchand. A growth algorithm for neural network decision trees. *Europhysics Letters*, 12(3):205–210, 1990.

[HB91] J. L. Holt and T. E. Baker. Back-propagation simulations using limited precision calculations. In *Proceedings of IJCNN'91*, pages II: 121–126, Seattle, 1991.

[HdW91] Alain Hertz and Dominique de Werra. The tabu search metaheuristic: How we used it. *Annals of Math. and Artificial Intelligence*, 1:111–121, 1991.

[HH93] Jordan L. Holt and Jenq-Neng Hwang. Finite precision error analysis of neural network hardware implementations. *IEEE Trans. on Computers*, 42(3):281–290, March 1993.

[HHP90] P. W. Hollis, J. S. Harper, and J. J. Paulos. The effects of precision constraints in a back-propagation learning network. *Neural Computation*, 2:363–373, 1990.

[HMP$^+$87] A. Hajnal, W. Maass, P. Pudlák, M. Szegedy, and G. Turán. Threshold circuits of bounded depth. In *Proceedings of* 28^{th} *IEEE FOCS Symposium*, pages 99–110, 1987.

[Jud90] J. S. Judd. *Neural Network Design and the Complexity of Learning.* MIT Press, Cambridge MA, 1990.

[May91] Eddy Mayoraz. On the power of networks of majority functions. In A. Prieto, editor, *Lecture Notes in Computer Science 540*, pages 78–85. IWANN'91, Springer-Verlag, 1991.

[May93a] Eddy Mayoraz. *Feedforward Boolean Networks with Discrete Weights: Computational Power and Training.* PhD thesis, Swiss Federal Institute of Technology, Department of Mathematics, 1993.

[May93b] Eddy Mayoraz. On the power of democratic networks. Technical Report ORWP 93/2, Swiss Federal Institute of Technology, Department of Mathematics, 1993. to appear in SIAM J. on Discr. Math.

[MN89] M. Mézard and J.-P. Nadal. Learning in feedforward layered networks: the tiling algorithm. *J. Phys. A: Math. Gen.*, 22:2191–2203, 1989.

[MR94] Eddy Mayoraz and Vincent Robert. Maximizing the robustness of a linear threshold classifier with discrete weights. *Network: Computation in Neural Systems*, 5(2):299–315, May 1994.

[Nad89] J.-P. Nadal. Study of a growth algorithm for a feedforward network. *International J. of Neural Systems*, 1(1):55–59, 1989.

[RCE82] D. L. Reilly, L. N. Cooper, and C. Elbaum. A neural model for category learning. *Biological Cybernetics*, 45:35–41, 1982.

[Ree93] R. Reed. Pruning algorithms—A survey. *IEEE Trans. on Neural Networks*, 4(5):740–747, September 1993.

[Ros58] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 63:386–408, 1958.

[SB91] Kai-Yeung Siu and Jehoshua Bruck. On the power of threshold circuits with small weights. *SIAM J. Disc. Math.*, 4(3):423–435, 1991.

[SD88] J. Sietsma and R. J. Dow. Neural net pruning—Why and how. In *IEEE International Conference on Neural Networks*, pages 325–333. IEEE, New York, 1988.

[SN90] J. A. Sirat and J.-P. Nadal. Neural trees: a new tool for classification. *Network*, 1:423–438, 1990.

[WHR90] Andreas S. Weigend, Bernardo A. Huberman, and David E. Rumelhart. Predicting future: a connectionist approach. *International J. of Neural Systems*, 1(3):193–209, 1990.