# Two Constructive Methods for Designing Compact Feedforward Networks of Threshold Units

Edoardo Amaldi * Bertrand Guenin †

**Abstract**

We propose two algorithms for constructing and training compact feedforward networks of linear threshold units. The SHIFT procedure constructs networks with a single hidden layer while the PTI constructs multilayered networks. The resulting networks are guaranteed to perform any given task with binary or real-valued inputs. The various experimental results reported for tasks with binary and real inputs indicate that our methods compare favorably with alternative procedures deriving from similar strategies, both in terms of size of the resulting networks and of their generalization properties.

**Keywords**: Feedforward networks, threshold units, constructive algorithms, greedy strategy.

## 1 Introduction

Since the beginning of the 90's, methods for constructing feedforward networks of threshold units have attracted a considerable interest (see for instance [14] and the included references). Constructive algorithms do not only tune the weights of the network but also vary its topology, in other words, they *design* rather than just train the network. Starting with a small network that is not powerful enough to perform the task at hand, units are allocated incrementally until the network's performance is satisfactory.

Determining an appropriate architecture is a fundamental issue. Theoretical and experimental results show that the size of a network has a strong impact on its generalization abilities. According to Occam's principle, among all networks capable of performing a given task, more compact ones —the ones with the smallest number of free parameters— are more likely to exhibit good generalization (see for instance [7, 16]). Therefore considerable attention has been devoted to constructive algorithms that build compact networks in terms of number of weights or of units.

Designing a minimum-size network is of course at least as hard as the problem of finding an appropriate set of values for the weights of a given architecture, which is NP-complete [27, 11]. However, in practice minimum networks are not required and near-optimum ones that exhibit good generalization properties suffice. The central question is how close to the optimum one should and one can get. On one hand, any task with binary outputs can be performed by a a three layer network with a large enough number of hidden units [17, 45, 46]. On the other hand, some recent computational complexity results imply that designing close-to-optimum networks

*School of Operations Research and Theory Center, Cornell University, Ithaca, NY 14853, amaldi@cs.cornell.edu

†Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213, bg0u@andrew.cmu.edu This work was partially done while the authors were with the Department of Mathematics, Swiss Federal Institute of Technology, CH-1015 Lausanne.

is a difficult problem to solve not only optimally but also approximately. In [28] it is shown that under a cryptographic assumption (namely, if trapdoor functions exist) no polynomial time algorithm can find a feedforward network with a bounded number of layers that performs a given task and that is at most polynomially larger than the minimum possible one, where the size is measured in terms of the number of bits needed to describe the network. In [29, 43, 5] the problem of designing close-to-minimum size networks in terms of nonzero weights is investigated for single perceptrons, i.e., linear threshold units. For instance, it is NP-hard to find for any task that can be performed by a single perceptron a set of weight values which correctly classifies all input vectors and whose number of nonzero weights exceeds the minimum one by any given constant factor [43, 5]. In fact, a stronger nonapproximability bound was established in [5] under a slightly stronger assumption than $P \neq NP$. These results, which derive from a worst-case analysis, compel us to devise efficient heuristics with good average-case behavior.

As is well-known, constructive algorithms have several advantages with respect to methods for training networks with a fixed architecture. In particular, the topology of the network does not need to be guessed in advance, they have lower computational requirements and they do only involve local computation. This makes them attractive from a hardware implementation point of view. However, the units cannot be trained in parallel since they are allocated incrementally.

Although methods for constructing networks of linear threshold units have been applied so far to several practical problems (see for instance [41, 13, 30, 32, 15]), algorithms that yield more compact networks with better generalization properties are needed to deal with larger applications.

The paper is organized as follows. In Section 2 we briefly comment on the main global strategies for constructing feedforward networks of threshold units and on the procedure we use to train single units. In Section 3 we describe the Shift algorithm which constructs networks with a single hidden layer. In Section 4 we present the PTI method which builds networks with an arbitrary number of layers. For each one of them an illustrative example is given and convergence is established. Experimental results for tasks with binary and real-valued inputs are reported and discussed in Section 5. The emphasis is on the size of the resulting networks and on their generalization properties. Finally, some concluding remarks and extensions are mentioned in Section 6.

# 2 Preliminaries

## 2.1 Global strategies

In spite of their particularities, most constructive methods are based on a *greedy* strategy (see [9, 8] for a non-greedy but computationally intensive approach to build decision trees). The global design problem is subdivided into a sequence of local training problems concerning single units. Starting with a network composed of a single unit, one (or a few) unit(s) is (are) added incrementally and its (their) subtask(s) is (are) defined so as to correct errors made by the current network. In order to construct compact networks, the typical local objective is to train new units so that they make as few errors as possible on the corresponding subtasks. Clearly, the size of the resulting network depends on the global strategy as well as on the effectiveness of the method used locally.

Strategies for constructing feedforward networks of threshold units can be subdivided into two major categories. In some algorithms, such as Upstart [20], the first unit is the future output unit and new hidden units are inserted between the input and output layers. In other ones, such as Tiling [36], new units and layers are created downstream of the existing ones so that the network is extended outward. The first category includes methods for neural trees [23, 41] or for cascade architectures [13] as well as sequential methods that focus alternatively on input patterns with a certain type of desired output [33, 15]. Procedures that construct networks with a single hidden layer and a predefined output function belong to the same category [19]. The OFFSET algorithm proposed in [38] and independently in [24] generates networks whose output unit computes a parity function, which can be itself implemented by a two-layer network of threshold units. Most of the above-mentioned methods yield networks with more than two layers of threshold units, but some actually generate a single hidden layer [33, 15]. Constructive algorithms that solve a sequence of Linear Programs (LP) have also been proposed [39, 32, 30]. While the one in [32] is restricted to simple halfspace intersections and neural decision lists, the Multisurface method [30] can deal with any type of task with binary outputs.

In this paper we focus on strategies in which simple perceptron-like procedures are used to train single linear threshold units. Unlike the LP-based methods that require complex optimization solvers, such algorithms are reasonable candidates for hardware implementation.

Constructive greedy strategies share two main kinds of limitations. The first one pertains to the computational complexity of training single linear threshold units. The problems of maximizing the number of correct classifications or minimizing the number of errors are not only NP-hard to solve optimally but also to approximate within some constant factors [1, 4, 6]. These problems becomes even harder when all input patterns with one of the two desired outputs must be correctly classified, like in the methods proposed in [33, 32, 15]. The second limitation is inherent to the greedy approach. Breaking down the network design problem into that of training a sequence of single units is certainly attractive. However, even if optimum weight vectors were available for each single unit, greedy strategies would not be guaranteed to yield minimum size networks, see [2] for several examples. The computational complexity of the overall network design problem clearly accounts for these two limitations.

In the sequel we focus on networks with $n$ inputs, $h$ hidden layers and a single output unit. The *task*, which is the set of training examples, is denoted by $T = \{(\mathbf{a}^k, b^k) : 1 \leq k \leq p\}$ where $b^k \in \{0,1\}$ is the *target* for the input vector $\mathbf{a}^k$. Although all algorithms will be described for binary tasks, i.e., $\mathbf{a}^k \in \{0,1\}^n$, they can be easily extended to deal with real-valued inputs using the stereographic projection as a preprocessing technique [40], see Section 5 and [12].

## 2.2 Training single units

Consider a single linear threshold unit with a weight vector $\mathbf{w} \in \mathcal{R}^n$ and a threshold $w_0 \in \mathcal{R}$. For any input vector $\mathbf{a}^k$, the output is given by:

$$y^k = \begin{cases} 1 & \text{if } v^k \geq 0 \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where $v^k = \sum_{j=1}^n w_j \mathbf{a}_j^k - w_0$ is the *total input.* A unit is ON (resp. OFF) if $y^k = 1$ (resp. $y^k = 0$). A task $T = \{(\mathbf{a}^k, b^k) : 1 \leq k \leq p\}$ is said to be *linearly separable* if there exist a $\mathbf{w}$ and a $w_0$ such that $y^k = b^k$ for all $k \in \{1, \dots, p\}$. In such a case, we say that the unit *performs* the task $T$. For the sake of simplicity, the threshold $w_0$ is considered as an additional weight corresponding to an additional input that is always ON.

In the well-known perceptron algorithm [37], the current weight vector is updated as follows

$$\mathbf{w}^{i+1} = \mathbf{w}^i + \eta_i (b^k - y^k)\mathbf{a}^k,$$

where $\eta_i \geq 0$ is the increment. Clearly, for nonlinearly separable tasks, updates never terminate and the sequence of $\mathbf{w}^i$ does not need to tend towards a weight vector that minimizes the number of errors or maximizes the number of correct classifications. Since it is $\mathcal{NP}$-hard to find such an optimal weight vector [1] and even an approximate one that is guaranteed to be a fixed percentage away from the optimum [4, 6, 26, 5], we have to settle for an efficient heuristic with a good average-case behavior.

In this work we use the *thermal perceptron* procedure [21] to find a close-to-optimal weight vector for each single unit. Although other methods could be used, this one compares favorably with the pocket procedure [22] and with some other methods based on least mean squared error or cross-entropy minimization [19, 21, 2]. The basic idea is to pay decreasing attention to misclassified input vectors with large total inputs. Starting with an arbitrary initial weight vector $\mathbf{w}^0$, input vectors are randomly selected and the current weight vector $\mathbf{w}^i$ is updated according to (1) but with:

$$\eta_i = \frac{t}{t_0} exp\left(\frac{-|v_i^k|}{t}\right), \tag{2}$$

where $v_i^k$ is the total input for $\mathbf{w}^i$ and $\mathbf{a}^k$. The control parameter $t$, referred to as "temperature", is decreased linearly from an initial value $t_0$ to 0 in a predefined number of cycles $C$ through all input vectors. Specifically, at the beginning of the $c^{th}$ cycle we set $t = \gamma t_0$ with $\gamma = 1 - (c/C)$. See [2, 3] for more details. Since $t$ is decreased to 0, the updates clearly terminate after $C$ cycles.

Finally, we recall that for any task with binary inputs it is always possible to correctly classify one input vector with a given target and all those with the other target [20, 33]. Indeed, suppose without loss of generality that $b^l = 1$ then $w_j = 2a_j^l - 1$ for $1 \leq j \leq n$ and $w_0 = \sum_{i=1}^n a_j^l(2a_j^l - 1)$ yield $y^l = 1$ and $y^k = 0$ for all $k \neq l$ with $1 \leq k \leq p$. Thus, a unit can always be trained to make at most $min\{|T^+|, |T^-|\} - 1$ errors, where $|T^+|$ (resp. $|T^-|$) denotes the number of input vectors $\mathbf{a}^k$ with $b^k = 1$ (resp. $b^k = 0$). When using the thermal perceptron procedure, it is easy to make sure that the weight vector obtained is at least as good as the above one.

# 3 The Shift algorithm

## 3.1 Description

Let $T_0 = \{(\mathbf{a}^k, b^k) : 1 \leq k \leq p\}$ be the task at hand. We start with a network composed of a single unit (the output unit) that is trained on $T_0$. If $T_0$ is nonlinearly separable, new units are allocated in a single hidden layer until the network correctly classifies all input vectors $\mathbf{a}^k$. As in the Upstart [20], we distinguish between two types of misclassications: either the output is *wrongly-on*, i.e., the output unit is ON while the target is $b^k = 0$, or it is *wrongly-off*, i.e., the output unit is OFF while $b^k = 1$.

Let $v^k$ denote the total input of the output unit when the vector $\mathbf{a}^k$ is presented to the network. Suppose we add a new $i$-th unit in the hidden layer that has a connection of weight $\Delta$ with the output unit, see Figure 1. Let $z^k$ denote the output value of the new unit for an input vector $\mathbf{a}^k$. If $z^k = 1$ then the corresponding total input of the output unit becomes $v^k + \Delta$. If $z^k = 0$ then $v^k$ remains unchanged. In order to correct wrongly-off errors, the total input $v^k$ for the wrongly-off input vectors must be increased while leaving the sign (or even the value) of $v^k$ unchanged for the correctly-off ones, i.e., when $v^k < 0$ and $b^k = 0$. This would be clearly achieved if we could add a new unit in the hidden layer connected to the output unit with a large enough weight $\Delta$ and providing an output 1 when $v^k < 0 \wedge b^k = 1$ and 0 when $v^k < 0 \wedge b^k = 0$. Since the corresponding task $T = \{(\mathbf{a}^k, 1) : \ v^k < 0 \wedge b^k = 1, 1 \leq k \leq p\} \cup \{(\mathbf{a}^k, 0) : \ v^k < 0 \wedge b^k = 0, 1 \leq k \leq p\}$ may be nonlinearly separable, the new unit is trained to perform it as well as possible.

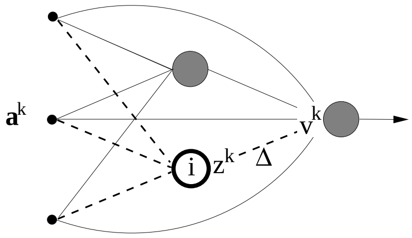


Figure 1: Adding a second unit in the hidden layer of a network with three inputs.

If $T$ is linearly separable then choosing $\Delta = -\min_k\{v^k : \ z^k = 1 \wedge b^k = 1\}$ ensures that all wrongly-off errors are corrected. However, since in general the task $T$ is not linearly separable, we shall choose $\Delta > 0$ more carefully.

Let $K \subseteq \{1, \ldots, p\}$ be the set of indices for which $v^k < 0$ and for which the output of the new unit is ON. Clearly, the output of the output unit remains unchanged for the input vectors $\mathbf{a}^k$ with $k \notin K$. Let us assume that the total inputs $v^k$ with $k \in K$ are sorted in decreasing order so that $v^{k_1} \geq v^{k_2}$ for all $k_1, k_2 \in K$ with $k_1 < k_2$. Choosing $\Delta = -v^{\tilde{k}}$ for some $\tilde{k}$ with $1 \leq \tilde{k} \leq p$ makes the output unit ON for all input vectors $\mathbf{a}^k$ with $k \leq \tilde{k}$ and $k \in K$ while it leaves the output unit OFF for all $\mathbf{a}^k$ with $k > \tilde{k}$ and $k \in K$. Thus $\Delta$ is selected among those $|K|$ values so as to minimize the total number of errors made by the new network.

Wrongly-on errors are corrected in a similar way. The new unit is assigned the task $T = \{(\mathbf{a}^k, 1) : \ v^k > 0 \wedge b^k = 0, 1 \leq k \leq p\} \cup \{(\mathbf{a}^k, 0) : \ v^k > 0 \wedge b^k = 1, 1 \leq k \leq p\}$. $\Delta$ is selected among the values $\{-v^k : \ v^k \geq 0 \wedge z^k = 1 \wedge 1 \leq k \leq p\} - \epsilon$ so as to minimize the total number of errors, where $\epsilon > 0$ is a small positive value because the output unit is ON when its total input is equal to zero.

Depending on the most frequent type of errors, the new unit is assigned to correct wrongly-on or wrongly-off errors. The process is repeated until no more errors are left.

### 3.2 An example

The *parity* task $T = \{(\mathbf{a}^k, b^k) : \ 1 \leq k \leq 2^n\}$ is defined as follows: the input vectors $\mathbf{a}^k$ are the $2^n$ possible $n$-dimensional vectors with $0, 1$ components and $b^k$ is equal to 1 (resp. 0) if $\mathbf{a}^k$ has an even (resp. odd) number of components equal to 1. Let $Q$ be an arbitrary vertex of the hypercube $\{0, 1\}^n$. All vertices that are at Hamming distance $r$ from $Q$ are said to be on the same *slice*, denoted by $S_r$. While $S_0$ consists only of $Q$ and $S_n$ only of $-Q$, slice $r$ contains $C_r^n$ vertices, i.e., all input vectors that differ from $Q$ in $r$ components. In the parity problem, the input vectors $\mathbf{a}^k$ lying on the same slice have the same target $b^k$ while those on adjacent slices $S_r$, $S_{r+1}$ have opposite targets. If $Q = (0, 0, .., 0)$, the hyperplane defined by $\sum_{i=0}^{n} x_i - r + \frac{1}{2} = 0$, separates the slices $S_0, ..., S_r$ from those $S_{r+1}, ..., S_n$.

Figure 2 illustrates how the algorithm works for the parity task with $n = 6$. Suppose that optimal weight vectors are found for each single unit. Any optimal weight vector for the output unit will separate three adjacent slices, say $0, 1$ and $2$, from the remaining four, say $3, 4, 5$ and $6$. Without loss of generality, let us assume that for the unique input vector $\mathbf{a}^k$ in slice 0 we have $b^k = 1$. Let $\mathbf{a}^k$ be any input vector in slice $k$ and let $v^k$ be the corresponding total input of the output unit. Hence we have $v^0 > v^1 > v^2 > 0 > v^3 > v^4 > v^5 > v^6$. As shown in Figure 2 (a), input vectors in slice 1 are wrongly-on while those in slices $4, 6$ are wrongly-off. It follows that there are $C_1^6 = 6$ wrongly-on input vectors and $C_6^6 + C_4^6 = 6 + 15$ wrongly-off ones. Therefore, we start by correcting wrongly-off errors.

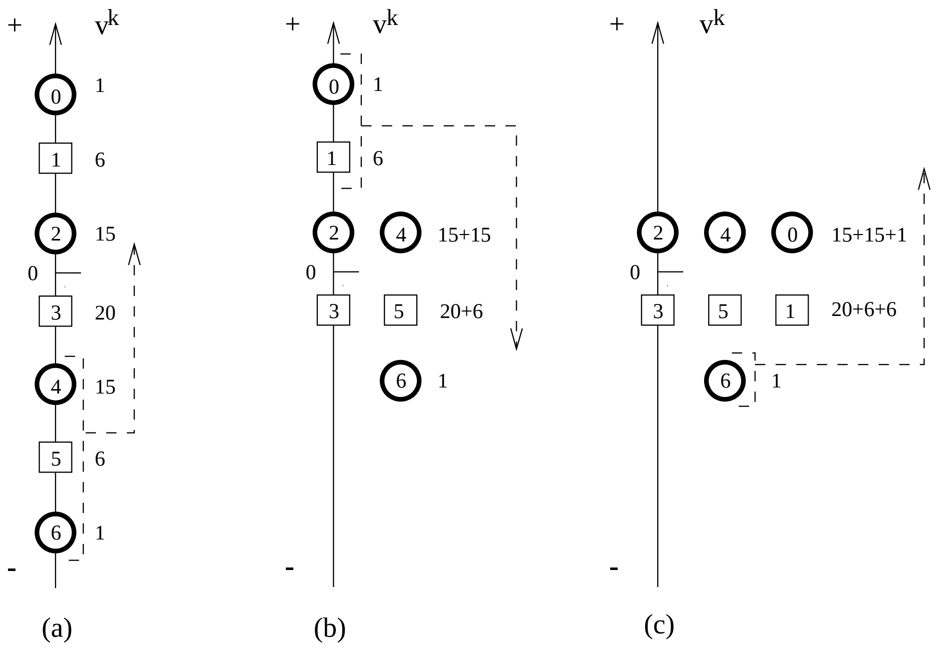


Figure 2: Parity problem with $n = 6$ and $p = 64$. White squares represent slices containing the input vectors with target $b^k = 0$ while solid circles represent those with target $b^k = 1$. The index and the cardinality of each slice are indicated respectively inside and beside the corresponding symbol. Slices are placed along a vertical axis at a position determined by the total input value $v^k$ (of the output unit) corresponding to the input vectors in the slice. In (a) the network is composed of a single unit, the output unit, and there are 22 errors left. In (b) a unit has been added to correct wrongly-off errors and only 7 errors remain. In (c) two hidden units have been added and only a single error remains. The resulting network contains three hidden units.

The new unit should map input vectors in slices $4, 6$ to targets $b^k = 1$ and those in slices $3, 5$ to targets $b^k = 0$. Any optimal weight vector for the new hidden unit will separate slice 4 from slice 3. Let us choose $\Delta = -v^5 + \epsilon$. In the resulting network, see Figure 2 (b), all input vectors in slice 1 are wrongly-on and the one in slice 6 is wrongly-off. Intuitively, we can think of the new hidden unit as having *shifted* as many black circles as possible above the origin while keeping as many white squares as possible below the origin.

The second hidden unit is assigned to correct wrongly-on errors. Therefore, an optimal weight vector separates slice 1 from slice 2 and the resulting network, see Figure 2 (c), has only one remaining error which can be corrected by a third hidden unit. It is worth noting that, if the weight connecting the first hidden unit to the output unit was just taken as $\Delta = -\min_k\{v^k : z^k = 1 \wedge b^k = 1\}$, at least four hidden units would be necessary in order to perform the original task [2].

In fact, it is easy to see that if optimal weights are found for single units the SHIFT algorithm yields a network which performs the parity function with $n$ inputs and $\lceil \frac{n+1}{2} \rceil - 1$ hidden units and direct input-to-output connections. Note that this solution has nearly half as many hidden units as the one mentioned in [20]. To the best of our knowledge, this very compact solution has not yet been discovered.

### 3.3 Convergence

**Theorem 3.1** *The* Shift *algorithm constructs a network consistent with any binary task $T$.*

Proof It suffices to show that each new hidden unit decreases the total number of errors by at least one. Consider a network with $i$ units in the hidden layer. Let $e_i^+$ denote the number of wrongly-on errors, $e_i^-$ the number of wrongly-off errors and $e_i = e_i^+ + e_i^-$ the total number of errors.

Suppose the $(i+1)$-th unit is used to correct wrongly-on errors. By definition, the task $T$ assigned to the $(i+1)$-th unit contains exactly $e_i^+$ associations of type $(\mathbf{a}^k, 1)$. Because of the fact mentioned at the end of Section 2.2, the number $e$ of errors made by this unit is at most $e_i^+ - 1$. After the $(i+1)$-th unit has been allocated to correct wrongly-on errors, only two types of errors are left: wrongly-off errors, which were already present before adding the $(i+1)$-th unit, and errors that have been introduced because the $(i+1)$-th unit gives the wrong output with respect to the task $T$. In other words, $e_{i+1} = e + e_i^- \leq e_i^+ - 1 + e_i^- = e_i - 1$. The wrongly-off case is dealt with similarly. □

Clearly, this convergence guarantee relies on the condition that it is possible for any binary task to separate any given input vector from all the others by a hyperplane. Although this is not necessarily true for tasks with real-valued inputs, such tasks can be dealt with using an appropriate preprocessing such as the stereographic projection [40], see Section 5.

### 3.4 Variants

The algorithm described above can be further refined. Suppose that one is about to add a new hidden unit to correct wrongly-off errors. Let $v_{\min}$ be the smallest total input $v^k$ for which the output unit is wrongly-off. In order to correct any wrongly-off error, it suffices to increase the corresponding total input $v^k$ by an amount of at most $-v_{\min}$. But if $\Delta \leq -v_{\min}$, all correctly-off input vectors with $v^k < v_{\min}$ remain correctly classified regardless of the output of the new unit. Therefore, the new hidden unit can be assigned the smaller task $T = \{(\mathbf{a}^k, 1) \mid v^k \leq 0 \wedge b^k = 1,\ 1 \leq k \leq p\} \cup \{(\mathbf{a}^k, 0) \mid 0 \geq v^k \geq v_{\min} \wedge b^k = 0,\ 1 \leq k \leq p\}$. Smaller tasks for units correcting wrongly-on errors are defined similarly.
In fact, once the weight of the connection to the output unit $\Delta$ has been selected, the number of pairs in the task assigned to the new hidden unit can still be reduced. Since all correctly-off input vectors $\mathbf{a}^k$ for which $v^k < -\Delta$ will remain correctly-off regardless of the value of the new hidden unit, one can remove from the task $T$ all pairs $\{(\mathbf{a}^k, 0) :\ v^k < -\Delta, 1 \leq k \leq p\}$ and re-train the new hidden unit on the resulting subtask. In general smaller subtasks lead to a smaller number of errors and therefore to more compact networks. One can proceed similarly for the wrongly-on case.

It is worth noting that direct connections between the input units and the output unit can be eliminated. Depending on the threshold value, all initial errors will then be either wrongly-on or wrongly-off. Since allowing direct input-to-output connection sensibly increases the representation power [42], simpler networks without direct connections might be preferred.

Thus, there are three main differences between the Shift and the Upstart algorithms. First, a single hidden layer is constructed. Second, the weight between each new hidden unit and the output unit is determined according to the task that is actually performed by this new unit instead of just considering the task it has been assigned to without taking into account what it does really achieve. Third, the task assigned to each new hidden unit is made as small as possible based on the value assigned to the weight to the output unit (by neglecting input vectors whose output would remain unchanged) so that they are more likely to be linearly separable. As we shall see in Section 5, these differences lead to substantial improvements in terms of network size and of generalization abilities.

# 4 The PTI algorithm

The second algorithm we propose is named PTI (Partial Target Inversion) and it constructs the network layer by layer in a way similar to the TILING algorithm [36]. Algorithms that generate an arbitrary number of hidden layers are worth pursuing because some functions may have a much more compact representation using a larger number of hidden layers. For threshold units with polynomially bounded weights, there exists even a class of functions that can be computed by a three-layer network with a polynomial number of units but that require an exponential number of units if only a single hidden layer is available [25]. The question of whether similar functions do also exist for unbounded weights is an important open question in circuit complexity.

## 4.1 Description

Consider a task $T_0$ and a network with more than $l$ layers. When an input vector $\mathbf{a}^k$ is presented to the network, it provides a vector $\tilde{\mathbf{a}}_l^k$ which corresponds to the output of the units in the $l$-th layer. The vector $\tilde{\mathbf{a}}_l^k$ is called the *internal representation* of $\mathbf{a}^k$ on the $l$-th layer, see Figure 3. The representations $\tilde{\mathbf{a}}_l^k$, $1 \leq k \leq p$, are said to be *faithful* if every time we have $b^{k_1} \neq b^{k_2}$ we also

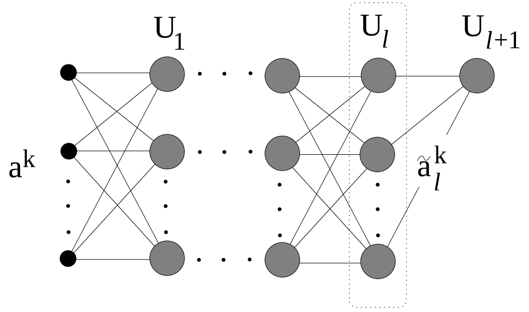


Figure 3: Example of network with an input vector $\mathbf{a}^k$ and its internal representation $\tilde{\mathbf{a}}_l^k$ on layer $l$. Units in layer $l$ are enclosed in a box. $U_1$ (resp. $U_l, U_{l+1}$) denote the first unit of layer 1 (resp. $l, l+1$).

have $\tilde{\mathbf{a}}_l^{k_1} \neq \tilde{\mathbf{a}}_l^{k_2}$. All input vectors $\mathbf{a}^k$ that have the same internal representation on a given layer are said to belong to the same *class*. The common internal representation of the $c$-th class is denoted by $\hat{\mathbf{a}}_l^c$ and is called its *prototype*, where $1 \leq c \leq q$ and $q$ is at most equal to the number of input vectors $p$. A class is *faithful* if all its input vectors $\mathbf{a}^k$ have the same target value $b^k$. Clearly, the classes induced by any layer of any network that performs the original task $T_0$ are necessarily faithful.

Unlike the SHIFT, the PTI algorithm builds the network layer by layer. Suppose $l$ layers have been created so far and that each class is faithful. The first unit in the $(l+1)$-th layer, which is denoted by $U_{l+1}$, is trained to map the prototype $\hat{\mathbf{a}}_l^c$ of each class induced by the $l$-th layer onto its corresponding target $b^k$. If $U_{l+1}$ performs this task, the algorithm stops and the network is completed. Otherwise, at least one of the two classes induced by $U_{l+1}$ is unfaithful and more units are added until one performs its task. As we shall see, every class is then faithful.

Assume that $i$ units have been inserted so far in the $l+1$-th layer. Let $T_i = \{(\hat{\mathbf{a}}_l^c, d_i^c) : c \in Q_l^i\}$ where $Q_l^i \subseteq \{1, \ldots, p\}$, denote the task assigned to the $i$-th unit, the last unit added in this layer (layer $l$).

The task $T_{i+1}$ assigned to the $i+1$-th unit is defined according to the target value in $T_i$ as well as to the actual output value of the $i$-th unit. Since the role of new units in a given layer is to ensure faithful representations, $T_{i+1}$ only contains the prototypes $\hat{\mathbf{a}}_l^c$ occurring in $T_i$ that correspond to classes induced by the current $l+1$-th layer which are unfaithful. The idea is to assign the *same* target as in $T_i$ to the prototypes $\hat{\mathbf{a}}_l^c$ for which the $i$-th unit is OFF ($d_{i+1}^c = d_i^c$) and the *opposite* target to those for which it is ON ($d_{i+1}^c = 1 - d_i^c$). The rationale is that, since we are trying to construct faithful classes, the prototypes for which the new unit has distinct outputs need not be distinguished further and hence can have the same target in the new task. When trying to break several unfaithful classes simultaneously, it may be worthwhile to flip the target of the prototypes on the $l$-th layer corresponding to several unfaithful classes. In fact, the specific targets are not important, the only requirement is that prototypes corresponding to a different $b^k$ trigger a different output for at least one unit in the $l+1$-th layer. The current layer is completed when the current unit performs its task.

## 4.2 An example

Consider the parity task $T_0$ with $n = 3$.

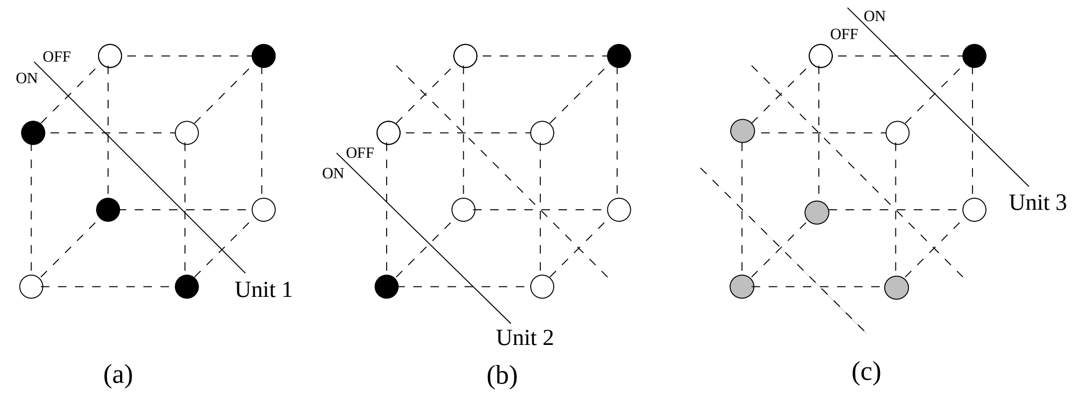


Figure 4: Tasks for the 3 first-layer units generated by the PTI for the Parity problem with $n = 3$. Black circles represent targets $b^k = 1$ and white circles represent targets $b^k = 0$. Grey circles correspond to input vectors in faithful classes, which are therefore not included in the task assigned to the third hidden unit.

The first unit in the first layer is trained on the task $T_1 = T_0$. An optimal weight vector leaves two errors, see Figure 4 (a). None of the two classes created by the first unit are faithful, thus the task $T_2$ for the second unit contains all $2^3$ input vectors $\mathbf{a}^k$. The targets for the task $T_2$ are obtained by reversing the targets in $T$ whenever the first unit is ON, see Figure 4 (b). The second unit creates two faithful classes. As shown in Figure 4 (c), the input vectors $\mathbf{a}^k$ in those faithful classes are not included in the task assigned to the third unit. After the third unit has been added all classes are faithful and the layer is completed. The first unit of the second layer has to map the 4 prototypes to their respective targets. Since the corresponding task is linearly separable, the resulting network contains a single hidden layer with three units.

In fact, it is easy to see that if optimal weights are found for single units the PTI algorithm yield a network which performs the parity function with $n$ inputs and $n$ units on the first layer and a single unit on the second layer. Note that this construction is optimal.

### 4.3 Convergence

**Theorem 4.1** *The* PTI *algorithm constructs a network consistent with any binary task* $T$.

PROOF Let $p$ denote the number of pairs in the task $T$. The result follows from the next three claims.

**Claim 1:** *Each new layer contains at most* $p$ *units.*
Suppose $l$ layers have been created so far. Let $T_i$ denote the task assigned to the $i$-th unit added in the new layer and $T_i^+$ the subtask containing only the pairs $(\mathbf{a}^k, 1)$ in $T_i$. By definition, the pairs in $T_{i+1}^+$ correspond to (1) pairs $(\hat{\mathbf{a}}_l^c, 0)$ of $T_i$ for which the $i$-th unit is ON and, (2) pairs $(\hat{\mathbf{a}}_l^c, 1)$ of $T_i$ for which it is OFF. Thus $|T_{i+1}^+|$ is equal to the number $e$ of errors made by the $i$-th unit. According to the simple fact mentioned at the end of Section 2.2, we have $e \leq |T_i^+| - 1$ and hence $|T_{i+1}^+| < |T_i^+|$. Since $|T_1^+| \leq p$, this proves the claim. $\diamond$

**Claim 2:** *Once a layer is completed (the last unit performs its task correctly), all classes it induces are faithful.*
Suppose that the $(l+1)$-th layer contains $h$ units. The claim is obviously true for $h = 1$ since the task assigned to the first unit in that layer, $T_1^{l+1}$, maps the prototype $\hat{\mathbf{a}}_l^c$ of each class onto the corresponding target. Suppose $h \geq 2$. In order to show that the $\tilde{\mathbf{a}}_{l+1}^k$ are faithful representations of the input vectors $\mathbf{a}^k$, it suffices to check that the representations of the prototypes $\hat{\mathbf{a}}_l^c$ on the $(l+1)$-th layer are faithful with respect to the task $T_1^{l+1}$. Let $\hat{\mathbf{a}}_l^{c_1}$ and $\hat{\mathbf{a}}_l^{c_2}$ be any pair of prototypes occurring in $T_1^{l+1}$ with opposite targets. Then two cases can arise:
(1) $\hat{\mathbf{a}}_l^{c_1}$ and $\hat{\mathbf{a}}_l^{c_2}$ have different targets in $T_h^{l+1}$. Since this task is performed correctly, they have different representations on the $l+1$-th layer.
(2) $\hat{\mathbf{a}}_l^{c_1}$ and $\hat{\mathbf{a}}_l^{c_2}$ have the same target in $T_h^{l+1}$. Then let $\tilde{\imath}$ be the smallest integer such that $1 \leq \tilde{\imath} < h$ and those two prototypes have the same target in $T_{\tilde{\imath}}^{l+1}$ and different targets in $T_{\tilde{\imath}+1}^{l+1}$. Given the way the tasks are defined, this can only happen if the $\tilde{\imath}$-th unit is ON for one of the prototypes and OFF for the other one. Hence, they have different representations on the $l+1$-th layer. $\diamond$

**Claim 3:** *At most* $p$ *layers are constructed.*
Consider a network with $l+1$ layers. As above, $U_l$ (resp. $U_{l+1}$) denotes the first unit of the $l$-th (resp. $l+1$-th) layer. Let $T_1^l = \{(\hat{\mathbf{a}}_{l-1}^c, d_l^c) : c \in Q_l^1\}$ and $T_1^{l+1} = \{(\hat{\mathbf{a}}_l^c, d_{l+1}^c) : cinQ_{l+1}^1\}$ be the corresponding tasks, where clearly $|Q_{l+1}^1| \leq |Q_l^l|$. In a way similar to [36], we establish the claim by showing that $U_{l+1}$ makes a number of errors on $T_1^{l+1}$ strictly smaller than the number of errors made by $U_l$ on $T_1^l$, see Figure 3.
Suppose that the $l$-th layer contains $h$ units and that $U_l$ makes at least one wrongly-off error, i.e., there is a $\hat{\mathbf{a}}_{l-1}^c$ for which it is OFF while $d_l^c = 1$. Let $\hat{\mathbf{a}}_{l-1}^{\bar{c}}$ be any one of these prototypes and let $h_1$ denote the number of 1 components in its representation $\hat{\mathbf{a}}_l^{\bar{c}}$ on the $l$-layer. If $w_0$ is the threshold of unit $U_{l+1}$, $w_1$ the weight of its connection to $U_l$ and $w_2, \ldots, w_h$ its remaining weights, it is easily verified that taking

$$w_j = \begin{cases} h_1 & \text{if} \quad j = 0 \\ h & \text{if} \quad j = 1 \\ 1 & \text{if} \quad 2 \leq j \leq h \text{ and } a_{jl}^{\bar{c}} = 1 \\ -1 & \text{if} \quad 2 \leq j \leq h \text{ and } a_{jl}^{\bar{c}} = 0 \end{cases}$$

$U_{l+1}$ is ON whenever $U_l$ is ON and OFF whenever it is OFF except for $\hat{\mathbf{a}}_l^{\bar{c}}$. Thus unit $U_{l+1}$ makes at least one fewer wrongly-off error than $U_l$.
Similarly, if $U_l$ makes some wrongly-on errors, then considering any wrongly-on prototype whose representation on the $l$-layer $\hat{\mathbf{a}}_l^{\bar{c}}$ has $h_1$ 1 components, and taking

$$w_j = \begin{cases} -h & \text{if} \quad j = 0 \\ h + h_1 - 1/2 & \text{if} \quad j = 1 \\ -1 & \text{if} \quad 2 \leq j \leq h \text{ and } a_{jl}^{\bar{c}} = 1 \\ 1 & \text{if} \quad 2 \leq j \leq h \text{ and } a_{jl}^{\bar{c}} = 0 \end{cases}$$

$U_{l+1}$ has the same output as $U_l$ except for $\hat{\mathbf{a}}_l^{\bar{c}}$. $\square$

Note that the above-mentioned upper bounds on the number of layers and of units in each layers are worst-case values. In practice, much smaller networks are constructed. The remarks given in Section 3.3 concerning tasks with real-valued inputs also hold in this case.

It is worth pointing that in the TILING algorithm a simple divide and conquer strategy is used to break down unfaithful classes, in which the new unit is trained to map internal representations $\tilde{a}_l^k$ belonging to unfaithful classes onto the corresponding desired output $b^k$. In fact, there is some similarity in the way the tasks for new units are defined in the PTI and in the OFFSET algorithms [2]. In the PTI, however, the output unit does not compute a parity function, more than one hidden layer is constructed, prototypes are considered instead of input vectors and only prototypes belonging to unfaithful classes are included in the task assigned to each new unit.

Finally, note that although the focus in this paper is on constructive algorithms using simple perceptron-like training procedures for single units, it is possible to allocate additional units to break down unfaithful classes in a more natural way at the expense of using more complex local procedures. See [35] for an to feedforward networks with binary weights.

# 5 Experimental results

Three test problems have been selected in order to compare the performances of the SHIFT and PTI algorithms to three alternative methods, namely the UPSTART [20] the TILING [36] and the OFFSET [24, 34], which derive from similar strategies. Therefore two important criteria are considered: network size and generalization abilities. Experiments have also been carried out on a real-world problem with real-valued inputs.

All results are obtained by training each single unit with 2000 cycles of the thermal perceptron procedure, see Section 2.2. A good initial temperature has been selected for each task using a trial-and-error procedure. Depending on the problem, it ranges from 1 to 4. Unless specified otherwise, we used the version of the SHIFT algorithm that does *not* re-optimize units after setting the connection weight $\Delta$ with the output unit (see Section 3.4). The reason for not using the refined version is that we wanted to compare methods which have similar computational cost.

For all methods, the computational cost is roughly proportional to the number of units. This is a fairly good estimate for SHIFT and UPSTART in which all units have the same number of inputs (namely $n$). However, the actual computational requirements may be much larger for PTI or TILING, where the units allocated in the successive layers may have a number of inputs much larger than $n$.

## 5.1 Network size

First we focus on the network size. Therefore we consider random binary tasks where all $2^n$ possible input vectors are assigned a target 1 or 0 with equal probability. This kind of tasks allows us to compare the ability of the different algorithms to construct compact networks. As previously seen, finding a compact representation is not only of interest by itself but it has also a strong impact on the network generalization properties. The size can be measured in terms of the number of units or the number of nonzero weights. For networks with more than two layers, the number of layers is also important.

Figures 5, 6 and 7 report the average number of units, weights and layers for random tasks with various input sizes $n$. All results are averages on 25 different random tasks. Standard deviations are also shown. Although the networks constructed by PTI have a considerably smaller number of units than those provided by TILING, OFFSET and UPSTART, they are larger than those produced by SHIFT.

As shown in Figure 6, the situation is slightly different in terms of weights. The networks built by PTI contain more weights than those obtained with UPSTART. This can be explained by the fact that the first layers generated by PTI contain a number of units which is in general much larger than the number of inputs $n$. The SHIFT algorithm turns out to build networks containing less hidden units and less weights than those produced by UPSTART. Moreover, they have a single hidden layer instead of a tree-like topology with a very large number of layers.

Not surprisingly, the UPSTART variant [20] which allows us to contract the tree in a single hidden layer performs sensibly worse than the original method, see Figure 8. We also compare the refined version of the SHIFT algorithm with the basic version.

## 5.2 Generalization

The generalization properties of the constructed networks have been tested on two structured problems. The first one is the classical *K-or-more clumps* problem that is frequently used as a benchmark [17]. Considering binary input vectors $\mathbf{a}^k \in \{0,1\}^n$ as strings of 0 and 1, input vectors with $K$ or more contiguous blocks of 1 have a target 1 while the others have a target 0.A cyclical boundary condition is used that makes the first and last components adjacent. Although the case $K = 2$ has sometimes been considered [20], we take $K = 3$. Indeed, for $K = 2$ there are only $n(n-1)+1$ possible input vectors with at most 1 clump. If $n = 25$, like in [20], considering tasks with $p = 600$ examples of which half of them are negative amounts to select half of all possible negative examples. In such a case, every system that just stores the negative examples and provides a positive answer for all the others, is guaranteed to make at most 25% of errors on any training set of the same size. For $K = 3$, the class of negative examples is much larger and hence the generalization problem harder.

The results obtained for the 3-or-more clumps problem with $n = 25$ are reported in Figures 9 and 10 as well as in Table 1. The tasks were generated through a stochastic procedure [10] ensuring that the average number of clumps is equal to 2.5. Thus approximately half of the examples are positive and half are negative. In order to evaluate the generalization abilities in terms of the available information, tasks of various sizes are considered. Generalization rates are estimated by measuring the percentage of misclassified vectors on a testing set, which is of the same size $p$ as the task and is generated using the same procedure.

The SHIFT algorithm produces networks that are much smaller than those obtained with all the other algorithms and that exhibit better generalization abilities. In spite of the difficulty of the problem, an average percentage of error of nearly 25% is reached for tasks with $p = 2000$

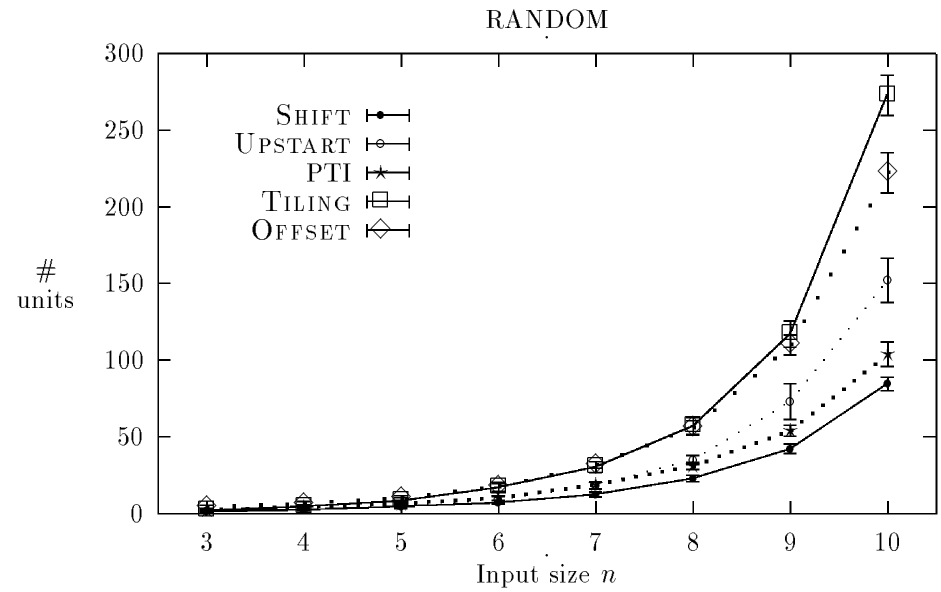


Figure 5: Random tasks with $p = 2^n$ examples for various input sizes $n$. Average number of units in the networks constructed by the different algorithms. Averages are over 25 different tasks.

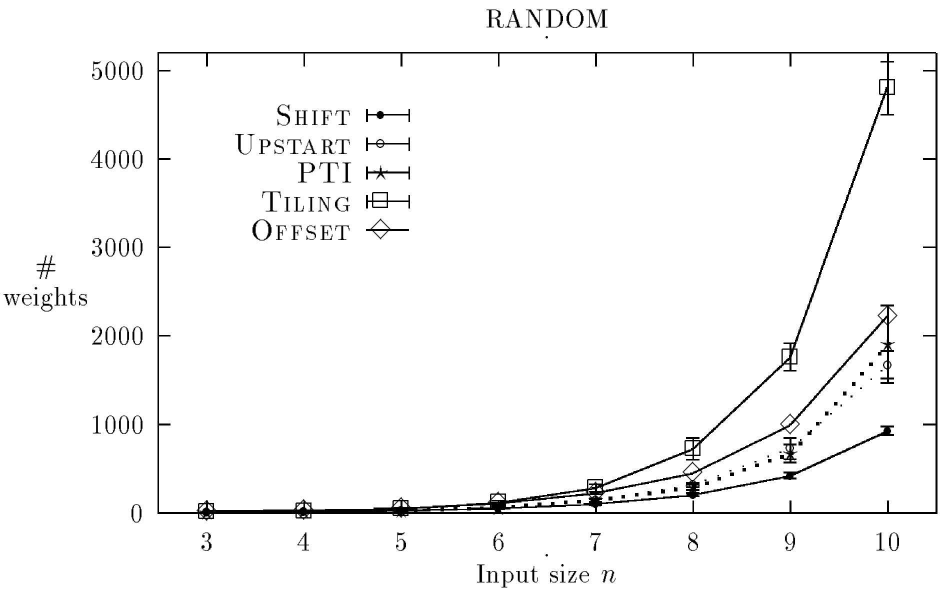


Figure 6: Average number of weights over 25 different random tasks. For OFFSET the weights involved in the network representation of the parity function are neglected.

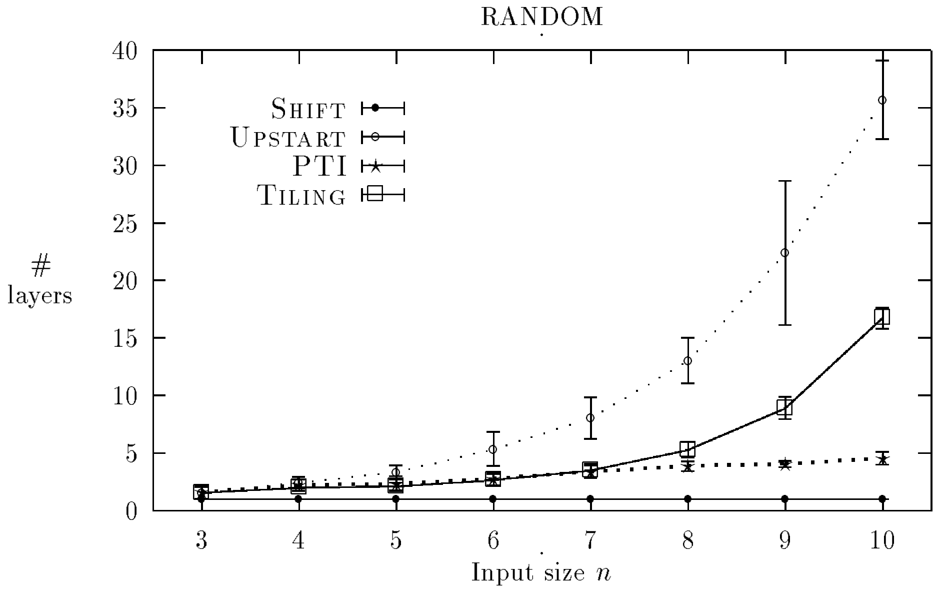


Figure 7: Average number of layers over 25 different random tasks. The Shift algorithm constructs networks with a single hidden layer.

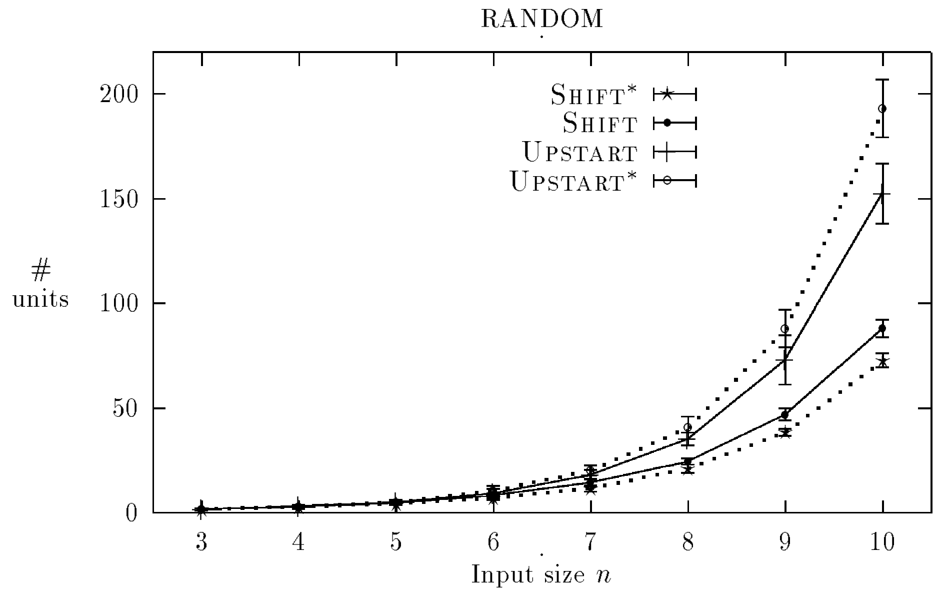


Figure 8: Comparison between the Shift and Upstart variants. Average number of units over 25 different random tasks. The Upstart* is the Upstart version that allows one to contract the tree-like topology in a single hidden layer. The Shift* is the Shift version where units are re-optimized.

examples. Moreover, the networks have a single hidden layer.

For clarity, the standard deviations relative to UPSTART are not plotted in Figure 9. As shown in Table 1, they are very large. This fact points out a further disadvantage of this algorithm. The size of the network constructed by UPSTART can vary enormously for tasks of the same type and the same sizes. The situation is different for our three algorithms. In particular, for SHIFT the standard deviations are very small.

The size of the networks produced by the TILING algorithm is not reported in Figure 9 because it is very large. As indicated in Table 1, it is greater than 550 for tasks with $p = 2000$ examples. It is worth noting that, unlike for all other algorithms, the size of the TILING networks grows so fast with respect to the number of examples $p$ that the average percentage of test error increases with $p$.

According to Table 1, the SHIFT algorithm compares very favorably to UPSTART when the version, which allows to contract the tree into a single layer, is considered. The size of the networks is much larger and they have considerably worse generalization abilities.

As for random tasks, the PTI algorithm performs much better than the TILING. The constructed networks are more compact than those obtained with all other methods except SHIFT, but they have poor generalization properties. The comparison between PTI and UPSTART is interesting. Although the resulting networks are of comparable size, their performances on the testing sets differ substantially. This may indicate that the networks built by PTI tend to have richer representation capabilities than the tree-like ones generated by UPSTART.

Finally, the OFFSET algorithm performs slightly worse than PTI in terms of network size and of generalization.

As second test problem, we propose an extension of the well-known symmetry problem that we refer to as *$K$-similarity*. The objective is to classify binary vectors of even size $n$ depending on whether or not the two subvectors defined by the first $n/2$ components and the last $n/2$ ones have a Hamming distance at most $K$. Clearly, 0-similarity corresponds to the usual symmetry problem.

The results obtained for 5-similarity tasks with up to $p = 3000$ examples are reported in Figures 11 and 12 and in Table 2. Notice that in this case the class of positive examples (i.e with Hamming distance at most $K$) is of cardinality

$$2^{n/2} \sum_{i=0}^{K} \binom{n/2}{i} .$$

Since this number is of the order of $10^8$ for $n = 30$, tasks with up to $p = 3000$ examples contain only a very small percentage of examples of the smallest class. Thus, error rates on the testing sets are meaningful.

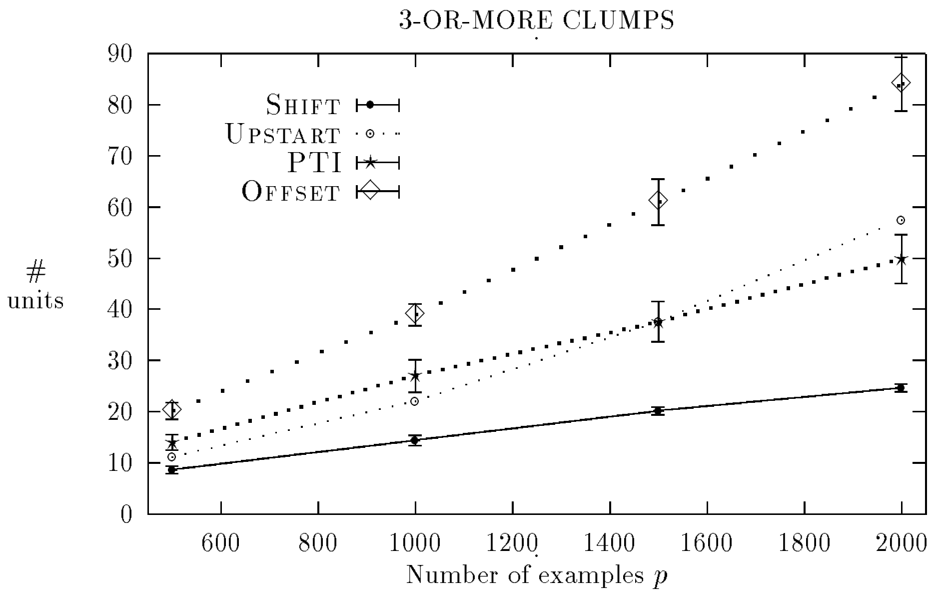


Figure 9: 3-or-more clumps tasks with $n = 25$ inputs and various number of examples $p$. Average number of units in the networks constructed by the different algorithms. Averages are over 25 different tasks.

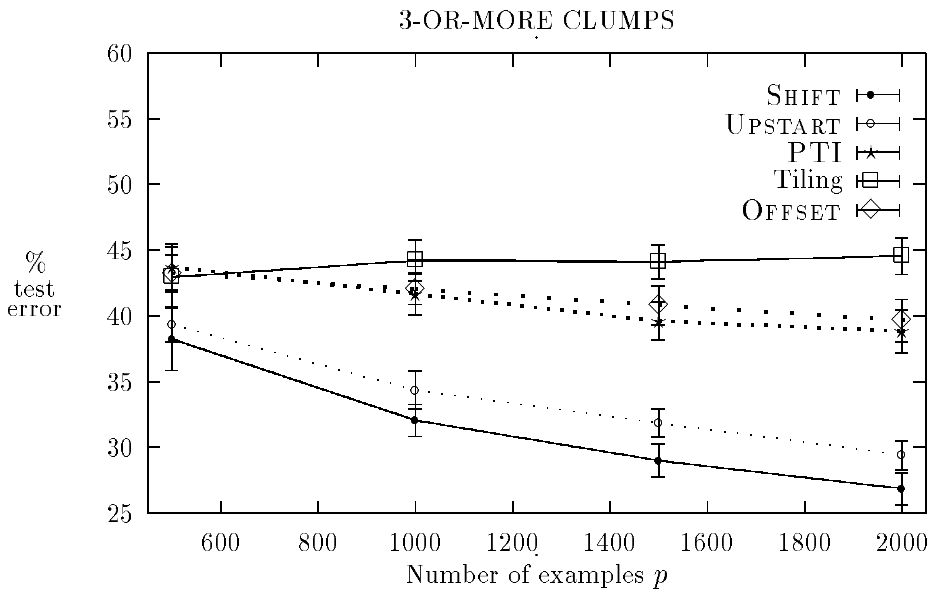


Figure 10: Generalization for 3-or-more clumps tasks with $n = 25$ inputs. Average percentage of errors on a testing set having the same size $p$ as the training set. Averages are over 25 different tasks.

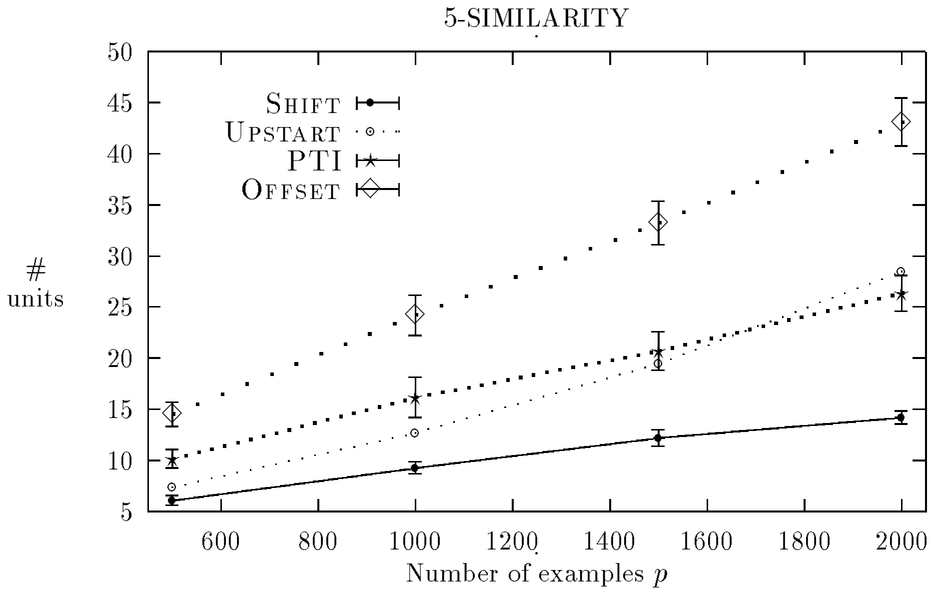


Figure 11: 5-similarity with $n = 30$ and various number of examples $p$. Average number of units in the networks constructed by the different algorithms. Averages are over 25 different tasks.

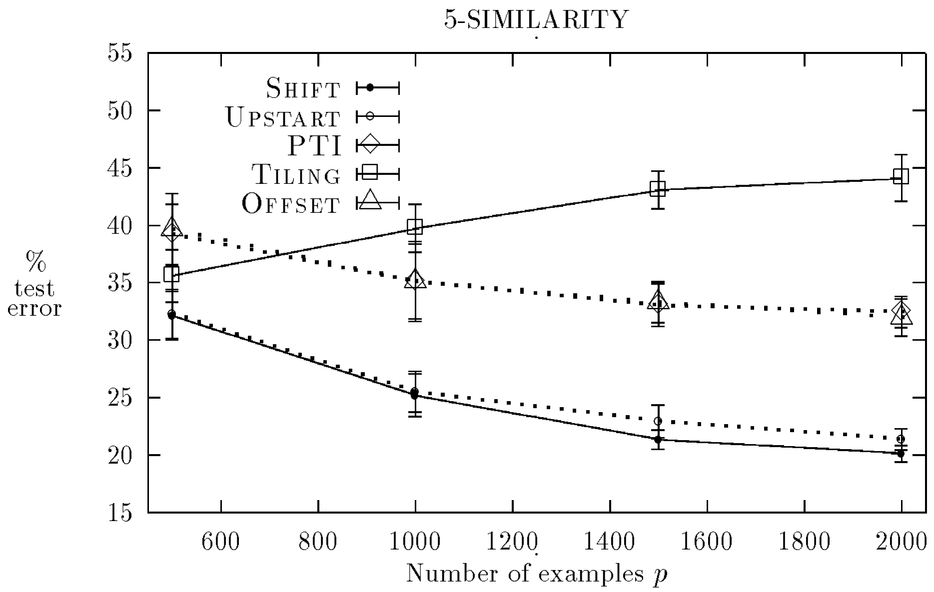


Figure 12: Generalization for 5-similarity tasks with $n = 25$ inputs and various number of examples $p$. Average percentage of errors on a testing set having the same size $p$ as the training set. Averages are over 25 different tasks.

| algorithm | units | weights | layers | % test error |
|---|---|---|---|---|
| Shift | $25.6 \pm 1.0$ | $664.6 \pm 27.5$ | 1 | $26.2 \pm 1.1$ |
| Upstart | $57.4 \pm 33.5$ | $1492.4 \pm 873.1$ | $22.1 \pm 23.1$ | $29.4 \pm 1.1$ |
| Upstart* | $94.0 \pm 70.5$ | $2444.0 \pm 1832.6$ | $34.8 \pm 30.4$ | $32.8 \pm 1.6$ |
| PTI | $49.8 \pm 4.8$ | $1276.0 \pm 170.7$ | $4.3 \pm 0.5$ | $38.8 \pm 1.6$ |
| Tiling | $559.6 \pm 34.4$ | $11322.0 \pm 685.1$ | $30.4 \pm 2.0$ | $44.6 \pm 1.4$ |
| Offset | $84.0 \pm 5.3$ | $2806.0 \pm 288.8$ | 1 | $39.7 \pm 1.6$ |

Table 1: 3-or-more clumps tasks with $n = 25$ inputs and $p = 2000$ examples. Size of the networks in terms of units, weights and layers as well as the percentage of errors on a testing set having the same size $p$ as the training set. Averages are over 25 different tasks. For the Offset algorithm, the number of weights includes those needed to compute the parity function. The Upstart* corresponds to the Upstart variant which allows one to contract the tree in a single hidden layer.

| algorithm | units | weights | layers | % test error |
|---|---|---|---|---|
| Shift | $14.7 \pm 0.8$ | $456.6 \pm 23.7$ | 1 | $19.6 \pm 0.9$ |
| Upstart | $28.4 \pm 11.7$ | $880.6 \pm 363.0$ | $11.5 \pm 8.2$ | $21.4 \pm 0.9$ |
| Upstart* | $56.3 \pm 42.4$ | $1743.7 \pm 1313.6$ | $26.7 \pm 27.8$ | $25.6 \pm 1.0$ |
| PTI | $26.4 \pm 1.8$ | $694.9 \pm 35.9$ | $3.6 \pm 0.5$ | $32.5 \pm 1.4$ |
| Tiling | $544.9 \pm 46.1$ | $11117.3 \pm 997.4$ | $29.6 \pm 2.2$ | $44.1 \pm 2.0$ |
| Offset | $43.1 \pm 2.4$ | $1096.3 \pm 88.4$ | 1 | $32.0 \pm 1.6$ |

Table 2: 5-similarity tasks with $n = 30$ inputs and $p = 2000$ examples. Size of the networks in terms of units, weights and layers as well as the percentage of errors on a testing set having the same size $p$ as the training set. Averages are over 25 different tasks. Offset and Upstart* correspond to the same variants mentioned in Table 1

The overall results are very similar to those obtained for the 3-or-more clumps problems. The relative differences between the various algorithms are confirmed. The SHIFT strategy performs best in terms of size of the networks as well as of their generalization performances. For $p = 2000$, the average percentage of testing error is smaller than 20%. Also in this case, the PTI algorithm compares very favorably to the TILING.

### 5.3 Real-valued inputs

To test the performance of SHIFT on a task with real-valued inputs, we consider the medical diagnosis application studied in [44]. The data set* is relative to breast cancer; it consists of 699 examples from two different classes referring to malignant and benign cases. The nine attributes correspond to nine cellular measurements made microscopically by a surgical oncologist. Measurements are coded as integer values between 1 and 10. The purpose is to separate benign cases from malignant ones.

This data set has been used to demonstrate the applicability of the Multisurface method [30] and the Linear Programming-trained neural network has been used at the University of Wisconsin Hospitals.

To deal with arbitrary task with real-valued inputs a simple preprocessing suffices. As seen in the previous sections, the convergence guarantee of constructive methods relies on the condition that it is possible to separate any given input vector from all the others. Although this is not necessarily true when real-valued inputs are considered, the problem can be circumvented by using the stereographic projection as a preprocessing technique [40]. Indeed, any given point on a hypersphere being linearly separable from any set of other points. This is achieved at the simple cost of adding an input dimension.

In [40] the stereographic preprocessing has been shown to work well for the famous benchmark problem that consists in separating two interlocking spirals [18]. This type of task has a circle-like structure and is well-suited to such a transformation. The question arises as to whether, using such a preprocessing, SHIFT can perform well on real-world data with real-valued inputs like in the breast cancer database.

If the first 300 examples of the database are considered, SHIFT builds a network with only 5 hidden units even when this simple adaptive procedure is used to train single units. The same size allows one to correctly classify the first 500 examples. If the whole data set is considered, 6 units suffice. These results were obtained by applying the above-mentioned thermal perceptron procedure for only 1000 cycles. For 500 cycles, the resulting networks contain at most 2 additional units. In all our experiments the number of hidden units was never greater than 7. For the sake of comparison, the Multisurface method trained on the first 369 examples generated a network with 7 hidden units [31]. The networks built by SHIFT are thus more compact in terms of units. Nevertheless, they have an additional input unit (due to the stereographic preprocessing) and direct input-to-output connections.

In this kind of application, the network ability to correctly classify new cases is of course a major concern. In [31] the authors report that the network trained with the Multisurface method on the first 369 examples correctly classified the following 45 cases. Since the data set has been expanded and slightly modified, we have carried out other tests in order to evaluate the generalization properties of the networks produced by the SHIFT algorithm. In particular, we used the first 400, 500 and respectively 654 examples for training and the other ones for testing.

*This breast cancer database, provided by Dr. William H. Wolberg, University of Wisconsin Hospitals, Madison, is available at the University of California Irvine repository of Machine Learning. `http://www.ics.uci.edu/ ~mlearn/MLRepository.html`

In all cases, including the first one where $p = 400$ and hence the training set contains less than 2/3 of the examples, the resulting networks made zero error on the unseen examples.

It is worth noting that these results have been obtained with the simplest version of the SHIFT algorithm without any optimization procedure. This suggests that SHIFT can also be successfully applied to real-world problems with real-valued inputs.

## 6 Concluding remarks

The experiments we have carried out for tasks with binary and real-valued inputs show that the SHIFT and the PTI algorithms design compact networks with good generalization abilities and that they compare favorably with three competing methods for building multilayer networks that derive from similar types of strategies.

When greedy constructive strategies are considered, it is not necessary and even desirable to train each unit optimally. So, sophisticated techniques like Linear Programming, which minimize an objective function that is a surrogate for the number of errors such as in [30], are not required and efficient perceptron variants such as the thermal one are appropriate. This is good news from the hardware implementation point of view.

In spite of the poor generalization performances of PTI networks, algorithms that generate an arbitrary number of hidden layers are worth pursuing because some functions may have a much more compact representation using a larger number of hidden layers, see Section 4.

It is worth noting that in [35] the basic idea of the PTI is extended to the deal with feedforward networks with binary weights and the $K$-similarity problem is considered as a benchmark.

Finally, the two algorithms we have presented can be extended to construct networks with multiple outputs. Although the current versions can be applied to each output unit separately, exploiting correlations would certainly lead to more compact networks with better generalization abilities. For instance, the tasks assigned to the new hidden units in the SHIFT should aim at correcting the errors of several output units simultaneously.


## Acknowledgments

The authors would like to thank Eddy Mayoraz for many helpful discussions and Marcus Frean for providing the Montecarlo code used to generate the $K$-or-more-clumps tasks.


## References


[1] E. Amaldi. On the complexity of training perceptrons. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, pages 55–60, Amsterdam, 1991. Elsevier science publishers B.V.

[2] E. Amaldi. *From finding maximum feasible subsystems of linear systems to feedforward neural network design.* PhD thesis, Department of Mathematics, Swiss Federal Institute of Technology, Lausanne, 1994.

[3] E. Amaldi and C. Diderich. On the probabilistic and thermal perceptron training algorithms. Manuscript, 1997.

[4] E. Amaldi and V. Kann. The complexity and approximability of finding maximum feasible subsystems of linear relations. *Theoretical Computer Science*, 147:181–210, 1995.

[5] E. Amaldi and V. Kann. On the approximability of minimizing nonzero variables and unsatisfied relations in linear systems. *Theoretical Computer Science*, 1997. To appear. Preliminary version available as ECCC Technical Report 96-15.

[6] S. Arora, L. Babai, J. Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes, and systems of linear equations. *Journal of Computer and System Sciences*, 54:317–331, 1997.

[7] E. B. Baum and D. Haussler. What size net gives valid generalization? *Neural Computation*, 1:151–160, 1989.

[8] K. Bennett and J. Blue. Optimal decision trees. Technical Report No 214, Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, New York, 1996.

[9] K. Bennett and E. J. Bredensteiner. Global tree optimization: A non-greedy decision tree algorithm. *Computing Science and Statistics*, 26:156–160, 1994.

[10] K. Binder. *Monte Carlo methods in statistical Physics*. Springer-Verlag, Berlin, 1979. Topics in Current Physics 7.

[11] A. Blum and R. Rivest. Training a 3-node neural network is NP-complete. *Neural Networks*, 5:117–127, 1992.

[12] N. Burgess. A constructive algorithm that converges for real-valued input patterns. *International Journal of Neural Systems*, 5:59–66, 1995.

[13] N. Burgess, M. N. Granieri, and S. Patarnello. 3-D object classification: Application of a constructive algorithm. *International Journal of Neural Systems*, 2:275–282, 1991.

[14] C. Campbell. Constructive learning techniques for designing neural network systems. In C. T. Leondes, editor, *Neural network systems, techniques and applications*. Academic Press, San Diego, 1997. To appear.

[15] C. Campbell and C. Perez Vicente. The target switch algorithm: a constructive learning procedure for feedforward neural networks. *Neural Computation*, 7:1245–1264, 1995.

[16] D. Cohn and G. Tesauro. How tight are the Vapnik-Chervonenkis bounds? *Neural Computation*, 4:249–269, 1992.

[17] J. Denker, D. Schwartz, B. Wittner, S. Solla, R. Howard, L. Jackel, and J. Hopfield. Large automatic learning, rule extraction, and generalization. *Complex Systems*, 1:877–922, 1987.

[18] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, Carnegie Mellon University, 1990.

[19] M. Frean. *Small Nets and Short Paths*. PhD thesis, Department of Physics, University of Edinburgh, Edinburgh, Scotland, 1990.

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

[21] M. Frean. A “thermal” perceptron learning rule. *Neural Computation*, 4(6):946–957, 1992.

[22] S. I. Gallant. Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 1:179–191, 1990.

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

[24] B. Guenin. Etude d'algorithmes d'apprentissage dans les réseaux de neurones en couches. Diploma Project, Swiss Federal Institute of Technology, Department of Mathematics, Lausanne, 1991.

[25] A. Hajnal, W. Maass, P. Pudlak, M. Szegedy, and G. Turan. Threshold circuits of bounded depth. In *Proc. of 28th Ann. IEEE Symp. on Foundations of Comput. Sci.*, pages 99–110, 1987.

[26] K-U. Höffgen, H-U. Simon, and K. van Horn. Robust trainability of single neurons. *Journal of Computer and System Sciences*, 50:114–125, 1995.

[27] S. J. Judd. *Neural network design and the complexity of learning.* MIT Press, Cambridge, MA, 1990.

[28] M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of ACM*, 41:67–95, 1994.

[29] J.-H. Lin and J. S. Vitter. Complexity results on learning by neural nets. *Machine Learning*, 6:211–230, 1991.

[30] O. L. Mangasarian. Mathematical programming in neural networks. *ORSA Journal on Computing*, 5:349–360, 1993.

[31] O. L. Mangasarian, R. Setiono, and W. H. Wolberg. Pattern recognition via linear programming: Theory and application to medical diagnosis. In T. F. Coleman and Y. Li, editors, *Large-sacle numerical optimization*, pages 22–30. SIAM Publications, 1990.

[32] M. Marchand and M. Golea. On learning simple neural concepts: from halfspace intersections to neural decision lists. *Network: Computation in Neural Systems*, 4:67–85, 1993.

[33] M. Marchand, M. Golea, and P. Ruján. A convergence theorem for sequential learning in two-layer perceptrons. *Europhysics Letters*, 11:487–492, 1990.

[34] D. Martinez and D. Estève. The offset algorithm: Building and learning method for multilayer neural networks. *Europhysics Letters*, 18:95–100, 1992.

[35] E. Mayoraz and F. Aviolat. Constructive training methods for feedforward neural networks with binary units. *International Journal of Neural Systems*, 7:149–166, 1996.

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

[37] M. L. Minsky and S. Papert. *Perceptrons: An introduction to computational Geometry.* MIT Press, Cambridge, MA, 1988. Expanded edition.

[38] G. J. Mitchison and R. M. Durbin. Bounds on the learning capacity of some multi-layer networks. *Biological Cybernetics*, 60:345–356, 1989.

[39] A. Roy, L. Kim, and S. Mukhopadhyay. A polynomial time algorithm for the construction and training of a class of multi-layer perceptrons. *Neural Networks*, 6:535–546, 1993.

[40] J. Saffery and C. Thorton. Using the stereographic projection as a pre-processing technique for Upstart. In *Proc. International Joint Conference on Neural Networks of the IEEE*, volume II, pages 441–446, 1991.

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

[42] E. D. Sontag. Feedforward nets for interpolation and classification. *Journal of Computer and System Sciences*, 42:20–48, 1992.

[43] K. van Horn and T. Martinez. The minimum feature set problem. *IEEE Transactions on Neural Networks*, 7:491–494, 1994.

[44] W. H. Wolberg and O. L. Mangasarian. Multisurface method for surface separation for medical diagnosis applied to breast cytology. *Proceedings of the National Academy of Science, U.S.A.*, 87:9193–9196, 1990.

[45] P. J. Zwietering, E. H. L. Aarts, and J. Wessels. The design and complexity of exact multilayer perceptrons. *International Journal of Neural Systems*, 2:185–199, 1991.

[46] P. J. Zwietering, E. H. L. Aarts, and J. Wessels. Exact classification with two-layered perceptrons. *International Journal of Neural Systems*, 3:143–156, 1992.