ESANN'1994 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 20-21-22 April 1994, D-Facto public., ISBN 2-9600049-1-4, pp. 113-128

A Constructive Training Algorithm for Feedforward Neural Networks with Ternary Weights

Frédéric Aviolat and Eddy Mayoraz
Department of Mathematics
Swiss Federal Institute of Technology, Lausanne

Abstract: We address the problem of training a special kind of Boolean feedforward neural networks, namely networks built from linear binary threshold units with ternary weights called *majority* units. We propose an original constructive training algorithm, inspired from both the approaches of the Tiling and the Upstart algorithms, and we give a simple proof of its convergence. Numerical experiments were carried out to test the size of the constructed networks as well as their generalization ability. Comparisons are made with classical threshold unit networks.

1 Introduction

The model we consider is a multilayer feedforward neural network with binary activations, represented as the set $\mathbb{B} = \{-1, +1\}$. Each neuron is a perceptron whose weights values are restricted to the set $\{-1, 0, +1\}$. The motivation of studying this model is the quantization of weights, which appears to be crucial when the network must be realized in hardware. Formally, each neuron computes a *majority* function $f : \mathbb{B}^n \to \mathbb{B}$ of the form $f(\boldsymbol{x}) = sgn(\boldsymbol{w}^\top \boldsymbol{x} + w_0)$, where $\boldsymbol{w} \in \{-1, 0, +1\}^n$, $w_0 \in \{\pm\frac{1}{2}\}$ and *sgn* is the sign function which returns $+1$ iff its argument is positive. The only use of the threshold w_0 is to determine the output when the potential $\boldsymbol{w}^\top \boldsymbol{x}$ is zero. Feedforward Boolean neural networks composed of majority units are referred to as *democratic networks*. A preliminary study has pointed out the interest of the simple computational model provided by the democratic networks [5, 7]. In this paper, we will concentrate on single output networks, that realize mappings from $\mathbb{B}^n$ to $\mathbb{B}$.

A major issue in neural networks is the training problem. Given an unknown function $f : \mathbb{B}^n \to \mathbb{B}$ and a task $T = \{(\boldsymbol{a}^k, b^k = f(\boldsymbol{a}^k))\}_{k=1}^p \subset \mathbb{B}^n \times \mathbb{B}$ supplying partial information on f, the goal of the training phase consists in determining a network that computes an extension g of T, such that g is a good approximation of f.

To tackle this problem, we propose a constructive algorithm. When designing a neural network, a crucial question is the choice of the architecture. The advantage of the dynamic approach is that we get rid of this problem and the

ESANN'1994 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 20-21-22 April 1994, D-Facto public., ISBN 2-9600049-1-4, pp. 113-128

algorithm will figure out itself the best architecture suited to the given task.

2 Constructive Training Algorithm

There are basically two categories of constructive training algorithms according to the sense of the growth of the network. The *forward* methods construct the network by adding new neurons beyond the existing part of the circuit. Conversely, the *backward* techniques insert the new processing units between the input layer and the most recently built layer. The Tiling algorithm [8] is a typical example of a forward constructive algorithm, while the construction of the network provided by the Upstart method [3] is backward.

Like the Tiling, our algorithm constructs a feedforward network layer by layer, by adding, after the last completed layer L, new units with their predecessors in L. Let $T = \{(\boldsymbol{a}^k, b^k)\}_{k=1}^{p} \subset \mathbb{B}^n \times \mathbb{B}$ denote the global task and $\boldsymbol{c}^{Lk}$ the *internal representation* of the input $\boldsymbol{a}^k$ on the layer L, i.e. the state vector of the units in L when the input is $\boldsymbol{a}^k$. Many different input vectors can share the same internal representation. The set of all the vectors $\boldsymbol{a}^k$ having a given internal representation $\boldsymbol{c}^{Lk}$ is called the *class* of the internal representation $\boldsymbol{c}^{Lk}$. A class is *faithful* if all the corresponding outputs b^k of any of its elements $\boldsymbol{a}^k$ are identical. An *unfaithful* class contains at least two elements $\boldsymbol{a}^k$ and $\boldsymbol{a}^l$ with $b^k \neq b^l$. The faithfulness of any class corresponding to internal representations on any layer is a necessary condition for the correct computation of the task.

The main ideas of the algorithm are as follows. During the construction of layer L, new units will be added one at a time in L in order to increase the global faithfulness, until the classes corresponding to $\boldsymbol{c}^{Lk}$ are faithful $\forall k$. Then, a new layer will be started, trained with the new task $T = \{(\boldsymbol{c}^{Lk}, b^k)\}_{k=1}^{p_L} \subset \mathbb{B}^{|L|} \times \mathbb{B}$, containing only one element per class. The algorithm stops when the first unit of the new layer realizes the task with no errors.

Making the classes faithful is not the only objective we should seek. While building a layer, we already know that we will need another layer to achieve the task, and we would like the next task for that layer to be as easy as possible. To handle this other objective, we can already consider the first unit u_1^{L+1} of the next layer. Thanks to ternary weights, we can assume without loss of generality that it will be connected to all units in L with $+1$ weights.

The local algorithm to train a single unit u_h^L in layer L will then work as follows. For each example of the task, the output of unit u_1^{L+1} is evaluated, receiving its input from all the units of layer L, including the currently trained unit. The weights of the trained unit are adapted in order to minimize the number of mistakes made by u_1^{L+1}. To ensure that the construction of a layer will end, we have to demand also that at least one unfaithful class is divided by the introduction of the new unit. Note that training is performed locally, on a single unit and that when it is done, the weights of that unit cannot further change. Unlike the Tiling, this approach always considers the full task, thus taking into account all the classes simultaneously. Training of the first unit of a layer can be performed the same way, because its output is the same as the first

ESANN'1994 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 20-21-22 April 1994, D-Facto public., ISBN 2-9600049-1-4, pp. 113-128

unit of the eventual next layer.

Several improvements of the local objective function to train a new inserted unit were considered, as this appears to be the crucial point. Let v_h^k denote the current potential at unit u_1^{L+1} for input $\boldsymbol{a}^k$, including the contribution of the new unit u_h^L under training. The weights of this new unit are adapted in order to minimize a function of those potentials. The most interesting objective was to minimize $\sum_{k \in N}(v^k)^2$ where N is the set of misclassified examples. This variant will be referred to as "enhanced version" in the following.

The local learning problem for unit u_h^L is a discrete optimization problem. We have to find a weight vector $\boldsymbol{w}$ with components in the set $\{-1, 0, +1\}$ and a threshold $w_0 \in \{\pm\frac{1}{2}\}$, optimizing the above defined criteria. The algorithm we use is a classical tabu search technique; a detailed description of this method can be found in [4]. In our problem, the set of feasible solutions S is $\{-1, 0, +1\}^{|L-1|} \times \{\pm\frac{1}{2}\}$. The initial solution is $\boldsymbol{w} = 0$ (except for u_1^L, for which we take the all $+1$ vector). 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 range $\{-1, 0, +1\}$, or in the inversion of the threshold $w_0 \leftarrow -w_0$. The algorithm works iteratively, exploring the set of feasible solutions S. To prevent cyling, a *tabu list* is used, remembering the last moves and forbidding their opposites.

Finiteness of the construction of a layer is ensured by the fact that at least one unfaithful class is divided by the insertion of a new unit. This implies that when adding a unit, we increase the number of classes; thus, after at most p_L insertions, all classes are faithful and the layer is completed. Global convergence of the algorithm is guarranteed if there is at least one (faithful) class containing two or more points, when a layer is completed. Indeed, this implies that the task for the next layer will have at most one association less than the previous task. The great advantage of tabu search is that it allows almost any objective function. It is thus easy to take into account the various objectives discussed above.

3 Numerical Experiments

In this section, we will present some of the results obtained with the simple version and with the more enhanced version of our algorithm. We will compare our results to those obtained with real weighted networks by the Tiling algorithm and by the Shift algorithm, which is a variant of the Upstart algorithm proposed by E. Amaldi and B. Guenin [1].

3.1 Tests of complete tasks

Let f be a Boolean function $\mathbb{B}^n \rightarrow \mathbb{B}$. 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 capable of implementing exactly the given function f.

Experiments were made on the *PARITY* function, defined as $f(\boldsymbol{x}) = \Pi_i x_i$ and on a *RANDOM* function, defined as $f(\boldsymbol{x}) = +1$ or -1 with the same prob-

ESANN'1994 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 20-21-22 April 1994, D-Facto public., ISBN 2-9600049-1-4, pp. 113-128

ablity. Figures 1 and 2 show the average size of the networks (number of units) produced by 10 runs versus the input size n for complete tasks. Performances of the simple and enhanced versions of our algorithm for democratic networks are compared with those of the Tiling and the Shift for classical networks.

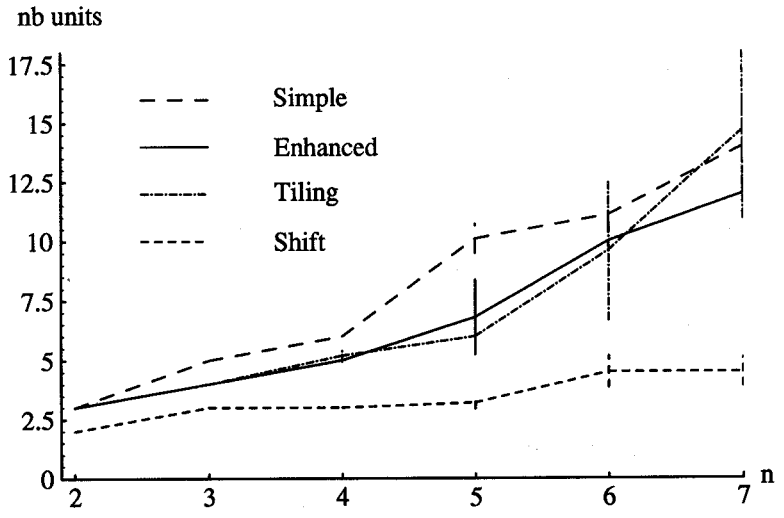

Fig. 1: Construction of *PARITY* functions. Average number of units of the networks built on complete tasks versus the input size n.

The results obtained with the refined version are comparable to those obtained with the Tiling, even though our model is much more constrained. For small input sizes, the enhanced algorithm constructed the smallest known democratic networks able to compute the *PARITY* function exactly [5].

3.2 Generalization tests

We present now numerical experiments done to test the generalization ability of the constructed networks. Consider the *2-CLUMPS* function defined as $f(\boldsymbol{x}) = +1$ if and only if $\text{Card}\{i \mid x_i = +1 = -x_{(i \bmod n)+1}\} \geq 2$.

Networks were built for an input size $n = 25$ 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 3 shows the average percentage of incorrect classifications, over 25 trainings, obtained with an improved version of our algorithm. Performances of the Tiling and the Shift algorithms are also plotted.

We observe that the generalization ability of the networks obtained with our algorithm are very similar to the results obtained with general linear threshold units. It is very encouraging to see that the performances are better than those of the Tiling.

ESANN'1994 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 20-21-22 April 1994, D-Facto public., ISBN 2-9600049-1-4, pp. 113-128

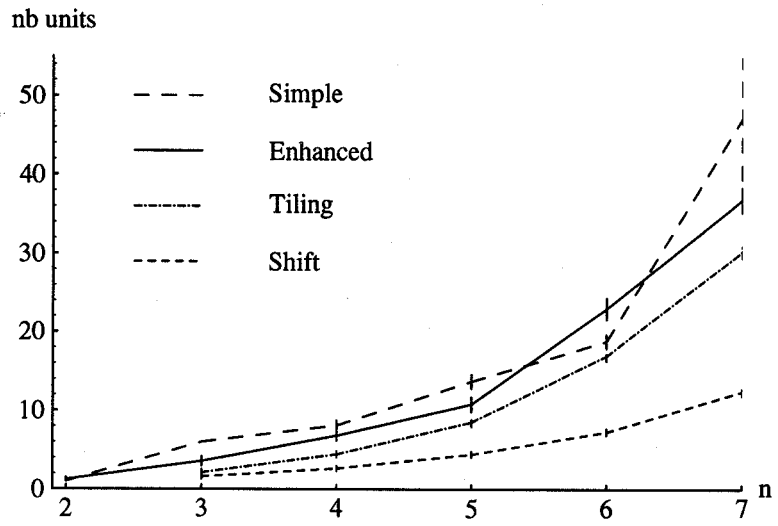

Fig. 2: Construction of *RANDOM* functions. Average number of units of the networks built on complete tasks versus the input size n.

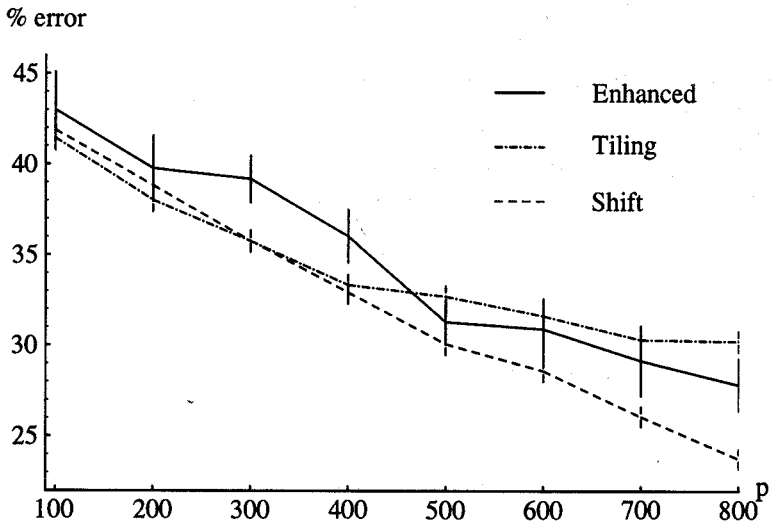

Fig. 3: Generalization of *2-CLUMPS*. Average percentage of errors made by networks trained on p randomly chosen examples in $\mathbb{B}^{25}$.

4 Conclusion

Training feedforward neural networks is a difficult problem and it becomes even harder when the weights are limited to discrete values. We proposed a constructive training method for feedforward Boolean networks with ternary weights in $\{-1, 0, +1\}$. Constructive techniques avoid the problem of dimensioning the

ESANN'1994 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 20-21-22 April 1994, D-Facto public., ISBN 2-9600049-1-4, pp. 113-128

network. The described method takes the advantages of forward and backward approaches and has simple convergence guarrantees. However, it is essential to generate networks of small size, since the generalization ability collapses when a too large network is built.

A variant that could be adopted is a method where the local learning strategy operates on more than a single 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 would be easy to train two new units concurrently. This extension has been considered and seems promising.

The reasonably small networks obtained on the complete tasks and the fair generalization performances reached for the *2-CLUMPS* function show that the discrete weights model can be a useful tool to realize Boolean functions, especially if it has to be done on chip.

References

[1] E. AMALDI AND B. GUENIN, *Constructive methods for designing compact feedforward networks of threshold units*, tech. rep., Swiss Federal Institute of Technology, Department of Mathematics, 1993.

[2] F. 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.

[3] M. FREAN, *The upstart algorithm: A method for constructing and training feedforward neural networks*, Neural Computation, 2 (1990), pp. 198–209.

[4] F. GLOVER, *"tabu search" part I*, ORSA J. Computing, 1 (1989), pp. 190–206.

[5] E. MAYORAZ, *On the power of networks of majority functions*, in Lecture Notes in Computer Science 540, A. Prieto, ed., IWANN'91, Springer-Verlag, 1991, pp. 78–85.

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

[7] ——, *On the power of democratic networks*, Tech. Rep. ORWP 93/2, Swiss Federal Institute of Technology, Department of Mathematics, 1993. submitted for publication.

[8] M. MÉZARD AND J.-P. NADAL, *Learning in feedforward layered networks: the tiling algorithm*, J. Phys. A: Math. Gen., 22 (1989), pp. 2191–2203.