The Upstart Algorithm : a method for constructing and training feed-forward neural networks

Marcus Frean

Department of Physics and Centre for Cognitive Science
Edinburgh University, The Kings Buildings
Mayfield Road, Edinburgh, Scotland
email : marcus@cns.ed.ac.uk

Neural Computation **2, 198-209** (1990)

Abstract

A general method for building and training multi-layer perceptrons composed of linear threshold units is proposed. A simple recursive rule is used to build the net's structure by adding units as they are needed, while a modified Perceptron algorithm is used to learn the connection strengths. Convergence to zero errors is guaranteed for *any* Boolean classification on patterns of binary variables. Simulations suggest that this method is efficient in terms of the numbers of units constructed, and the networks it builds can generalise over patterns not in the training set.

1 Introduction

The Perceptron Learning Algorithm (Rosenblatt 1962) offers a powerful but restricted method for learning binary classifications. All classifications that can in theory be learned by the Perceptron architecture *will* be learned; however, the number of classifications it can learn is only a tiny subset (*linearly separable* patterns) of those that are possible (Minsky and Papert 1969). To perform any arbitrary classification successfully, "hidden" units and/or feedback between units are required. The problem is to train such networks, and recently quite powerful methods have become available, most notably "back propagation" in its various forms (eg, Rumelhart, Hinton and Williams 1986). However, because many of these methods are based on hill-climbing which has the perennial problem of becoming stuck in local optima, they cannot guarantee that the classification will be learned. Another problem is that *a priori* no realistic estimate can be made of the number of hidden units that are required. Recently, methods such as the Tiling Algorithm (Mezard and Nadal 1989) and others (Gallant 1986a, Nadal 1989) have been proposed which get around both these problems. In these, the hidden units are constructed in layers one by one as they are needed. By showing that at least one unit in each layer makes fewer errors than a corresponding unit in the previous layer, eventual convergence to zero errors is guaranteed.

The method described here also constructs units as it goes, but in a simple and quite different way. Instead of building layers from the input outwards until convergence, new units are interpolated between the input layer and the output. The role of these units is to correct mistakes made by the output unit.

2 Basics

Suppose we are given a binary classification to be learned. Each input pattern of N binary values has an associated target output which the network must learn to produce. The units are all *linear threshold* units connected by variable weights to the inputs, with output o given by

$$o = \begin{cases} 1 & \text{if } \phi \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{where} \quad \phi = \sum_{i=0}^{N} W_i \xi_i$$

The W's are the weights, and ξ_i is the value of the i^{th} input in the given pattern. The necessary threshold or "bias" is included by having an extra input which is set to 1 for all the input patterns. On presentation of pattern μ, Perceptron Learning (Rosenblatt 1962) alters the weights if the target t^μ differs from the output:

$$\Delta W_i^\mu = (t^\mu - o^\mu)\xi_i^\mu \tag{1}$$

Using this method any linearly separable class will be learned, but when the patterns are not linearly separable the values of the weights never stabilise. However, a simple extension called the *Pocket Algorithm* (Gallant 1986b) suffices to make the system well behaved. This consists of running a Perceptron exactly as above with a random presentation of patterns, but also keeping a copy of the set of weights which has lasted longest without being changed. This set of weights will give the minimum possible number of errors with a probability approaching unity as the training time increases. That is, if a solution giving say p or fewer errors exists then the pocket algorithm can be used to find it (although unfortunately there is no bound known for the training time actually required). I make use of this algorithm to demonstrate convergence to zero errors.

3 Rationale

The basic idea is that a unit builds other units to correct its mistakes. Any unit (say Z) can make two kinds of mistake:

"wrongly ON" $\quad (o_Z^\mu = 1, \text{ but } t_Z^\mu = 0)$

"wrongly OFF" $\quad (o_Z^\mu = 0, \text{ but } t_Z^\mu = 1)$

Consider patterns for which Z is wrongly ON: Z could be corrected by a large negative weight from a new unit (say X) which is active only for those patterns. Likewise when Z is wrongly OFF it could be corrected by a large positive weight from another unit (say Y) which is active at the right time. Hence X and Y (also connected to the input layer by variable weights) can be trained with targets which depend on Z's response. These new units might be called "daughters" since they are generated[1] by the established "parent" unit, Z.

Consider, for example, the targets we should assign to X, the unit whose role is to inhibit Z. We would like X to be active if Z was wrongly ON, and silent if Z was correctly ON. Similarly X should be silent if Z was wrongly OFF (to avoid further inhibition of Z). Finally, X could be silent if Z was correctly OFF, although if X is active in this case, the effect is merely to reinforce Z's response when it was *already correct*. This doesn't itself cause an error, so in practice we can eliminate these patterns from X's training set. This elimination makes the problem easier and faster to solve, but is not essential for the error-correcting property described below. Targets for Y are similarly derived. These target assignments are summarised in Figure 1.

An important point is that the "raw" output of unit Z is used to set the daughter's targets, rather than the value of Z after the daughters have exerted any effect, since this would introduce feedback.

Two useful results follow immediately from this training method, because it essentially gives daughters (X or Y) an easier problem to solve than their parent (Z). Firstly, daughters can always make fewer errors than their parent, and secondly, connecting daughter to parent with the appropriate weight will always reduce the errors made by the parent.

Proof : Z's errors are

$$e(Z) = e(Z)_{\mathrm{ON}} + e(Z)_{\mathrm{OFF}}$$

where $e(Z)_{\mathrm{ON}}$ is the number of patterns for which Z is wrongly ON.
If X responded OFF to every pattern, it would make as many errors as there were patterns of target $t_X = 1$. However, X can always do better than this. In particular, it can always be ON for at least one input pattern whose target is 1 and OFF for all other patterns. For example if the input weights are

$$\begin{aligned} W_i &= 2\xi_i^\mu - 1 \\ \text{with a bias weight} \quad W_0 &= -\sum_j \xi_j^\mu (2\xi_j^\mu - 1) \end{aligned}$$

only the μ^{th} pattern turns the unit ON. Given that the Pocket Algorithm can find the optimal weights visited by a perceptron with any given probability, at worst X could find the above weights. Therefore

$$e(X) \; < \; e(Z)_{\mathrm{ON}} \; \leq \; e(Z) \tag{2}$$

A similar argument applies to Y. It also follows that Z's errors are reduced by X, since

$$\begin{aligned} e(Z \text{ with } X) &= e(X)_{\mathrm{ON}} \; + \; e(X)_{\mathrm{OFF}} \; + \; e(Z)_{\mathrm{OFF}} \\ &= e(X) \; + \; e(Z)_{\mathrm{OFF}} \\ &< e(Z) \end{aligned} \tag{3}$$

[1]Note however that the *activity* proceeds from daughter to parent.

The recursive element of the Upstart algorithm.

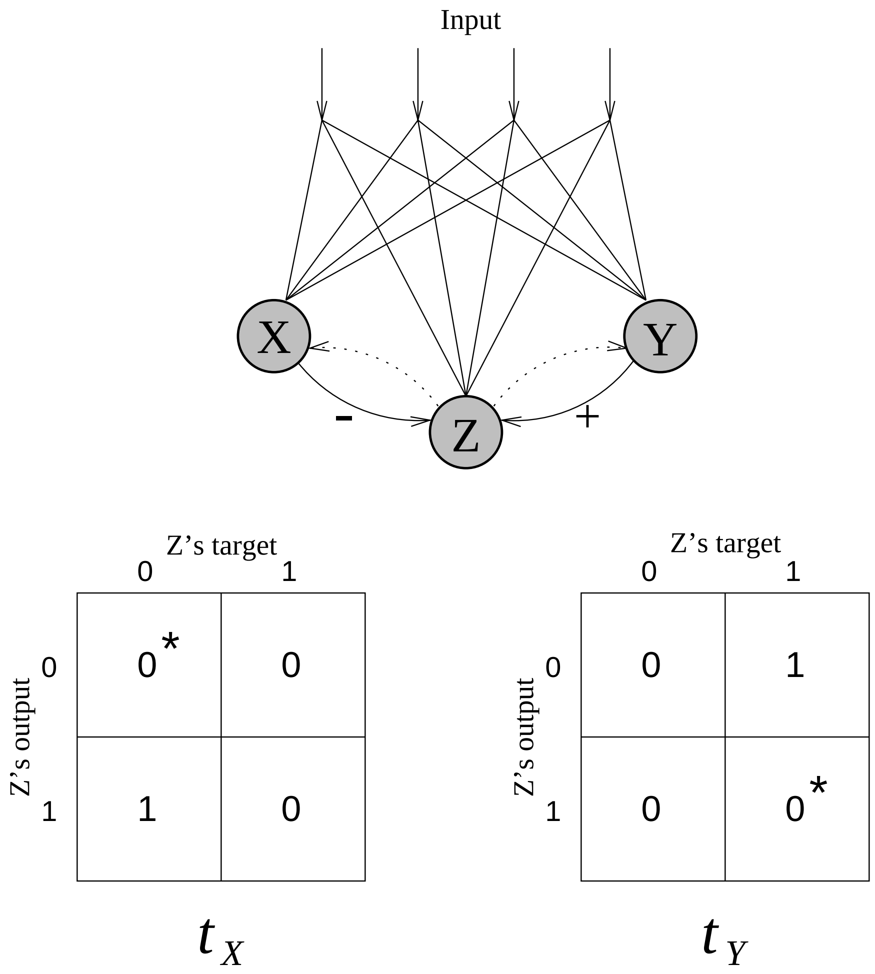

Figure 1: Correcting a parent unit: the left hand table gives the targets, t_X, for the daughter unit X for each combination of (o_Z, t_Z). For example, the lower left-hand entry assigns t_X to be 1 when $o_Z = 1$ and $t_Z = 0$: the 'wrongly ON' case. Similarly the right hand table gives the values of t_Y for the daughter unit Y. The dotted line represents the flow of this target information. The "starred" entries correspond to cases where the pattern could be eliminated from the daughter's training set.

and similarly for Y on its own. When the joint action of X and Y is considered, the same result holds, i.e. $e(Z \text{ with } X,Y) < e(Z) - 1$. In the next section an algorithm which uses the first of these results is described. Other possibilities are discussed in section 6.

4 The Upstart Algorithm

Assume we already have a unit Z which sees input patterns $\xi_i^\mu : i = 1,..,N$ and has associated targets t_Z^μ. The weights from the input layer to Z are trained to minimise the discrepancies between Z's output and target and once trained, these weights remain frozen. This "first" unit is actually the eventual output unit, and its targets are the classification to be learned. The following two steps are then applied recursively, generating a binary branching tree of units. Thus daughter units behave just as Z did, constructing daughter units themselves if they are required.

Step 1. If Z makes any "wrongly ON" mistakes, it builds a new unit X, using the targets given in Figure 1. Similarly if Z is ever "wrongly OFF" it builds a unit Y. Apart from the different targets these units are trained and then frozen just as Z was.

Step 2. The outputs of X and Y are connected as inputs to Z. The weight from X is large and negative whilst that from Y is large and positive. The size of the weight from X [Y] needs to exceed the sum of Z's positive [negative] input weights, which could be done explicitly or by Perceptron Learning.

New units are only generated if the parent makes errors, and the number of errors decreases at every branching. It follows that eventually none of the terminal daughters makes any mistakes, so neither do their parents, and neither do *their* parents and so on. Therefore every unit in the whole tree produces its target output, including Z, the output unit. Hence the classification is learned.

5 Simulations

In all the simulations shown here the "starred" entries in Figure 1 were not included in a daughter's training set.[2] To decrease training times, a fast and well behaved version of perceptron learning (Frean, in preparation) was used to train the weights, in preference to the Pocket Algorithm. While this method is not guaranteed to find the optimal weights, in practice it produces substantially fewer errors in a given time than the Pocket Algorithm. The weight changes given by the usual Perceptron Learning Rule (equation 1) were simply multiplied by

$$\frac{T}{T_0} \exp\left(\frac{-|\phi^\mu|}{T}\right)$$

[2] If the whole training set is used in every case, the number of units produced is relatively unaffected for the problems investigated here, but the training time (a combination of the time per epoch and number of epochs required to generate a comparable network) is approximately doubled for the problems discussed.

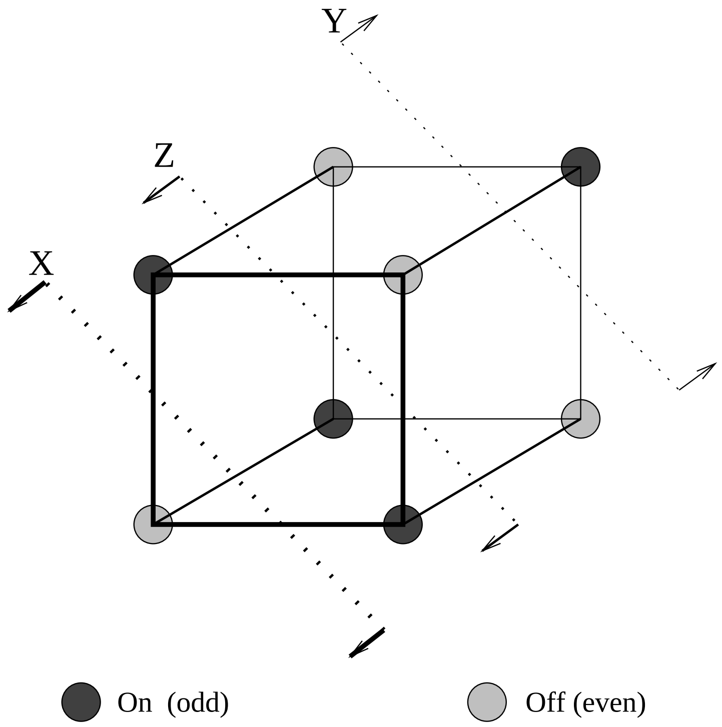

Figure 2: Solution for 3-bit Parity. The output unit Z on its own can clearly make a minimum of two mistakes, when the plane defined by its weights cuts the cube as shown. X corrects the wrongly ON pattern by responding to it alone, and similarly Y corrects the wrongly OFF pattern.

This factor decreases with $|\phi|$, which measures how "serious" the error is. The rationale behind this approach is that an error where $|\phi|$ is large is difficult to correct without causing other errors and should therefore be weighted as less significant than those where $|\phi|$ is small. The "temperature" T controls how strongly this weighting is biased to small $|\phi|$. T was reduced linearly from T_0 to zero over the entire training period. At high T the Perceptron Rule is recovered, but as T decreases the weights are "frozen". Unless otherwise stated, T_0 was set to 1, and 1000 passes were made through the training set.

Parity

In this problem the output should be ON if the number of active inputs is odd, and OFF if it is even. Parity is often cited as a difficult problem for neural networks to learn, being a predicate of order N (Minsky and Papert 1969); that is, at least one hidden unit must sample all of the N inputs. It is also of interest because there is a known solution consisting of a single layer of N hidden units projecting to an output unit. It is easy to see how the Upstart Algorithm tackles parity (see figure 2). Essentially the same structure as that shown for N=3 would arise for any N, although for large problems the optimal weights become much harder to find. I have tried parity for up to N=10, and in all cases N units are constructed, including the output unit. In all cases except N=10, 1000 passes were sufficient to generate the minimal tree. For 10-bit parity, the figure was 10,000.

Random mappings
In this problem the binary classification is defined by assigning each of the 2^N patterns its target 0 or 1 with 50% probability. Again this is a difficult problem, due to the absence of correlations and structure in the input for the network to exploit. The networks obtained for N up to 10 are summarised in Figure 3, with comparisons to the Tiling Algorithm.

Generalisation
Neural networks are often ascribed the property of generalisation: the ability to perform well on all patterns taken from a given distribution after having seen only a subset of them. Several workers (Denker et al. 1987, Mezard and Nadal 1989) have looked at generalisation using the "2-or-more clumps" predicate. The problem is this: given an input pattern, respond ON if the number of clumps is 2 or greater and OFF otherwise, where a "clump" is a group of adjacent[3] 1's bounded on either side by 0's. As with parity, there is a solution consisting of a single hidden layer of N units which would solve the problem exactly. Following Mezard and Nadal, the patterns were generated by a Monte Carlo method (Binder 1979) such that the mean number of clumps is 1.5. I used N=25 inputs, with a training set of up to 600 patterns. The set used to test the resulting net's ability to generalise was a further 600 patterns. The results, with comparisons to the Tiling Algorithm, are summarised in Figure 4.

6 Discussion

The architecture generated by this procedure is unconventional in that it has a hierarchical tree structure. However in the case where we choose not to eliminate any training patterns there is an equivalent structure with the same units arranged as a single hidden layer. Consider two daughters (say X,Y) and their parent (Z). With primes denoting "corrected" values, the corrected value o'_Z is always equal to $o_Z - o'_X + o'_Y$. This formulation is possible because the combinations which would disobey this never occur. For example Y would never be correctly ON if Z was ON. Since this is true of *every* unit in the tree, the final output is simply a sum of the "raw" responses. For example:

$$\text{output} \quad = \quad o'_Z \quad = \quad o_A - o_B + o_C + \cdots - o_X + o_Y + o_Z$$

Imagine the tree units disconnected from one another and placed in a single layer. A new output unit connected to this "hidden" layer can easily calculate the appropriate sum by, for example, having weights of $+1$ from each unit which adds to the sum and -1 from each unit that subtracts, with a bias of zero. In effect we can convert a binary tree into a single hidden layer architecture which implements the same mapping, at the expense of adding one unit and being unable to exploit pattern elimination. The algorithm for constructing a single hidden layer architecture is: construct units as before, omitting Step 2 (where they are connected into a feed-forward tree). Then connect all the units so constructed to a new output unit. The weights can be learned by Perceptron Learning, or can be inferred from the tree structure: there is a sign reversal for every "X" daughter.

The Upstart method can be extended in a number of ways.

[3] Circular boundary conditions are used: input 1 is "adjacent" to input N

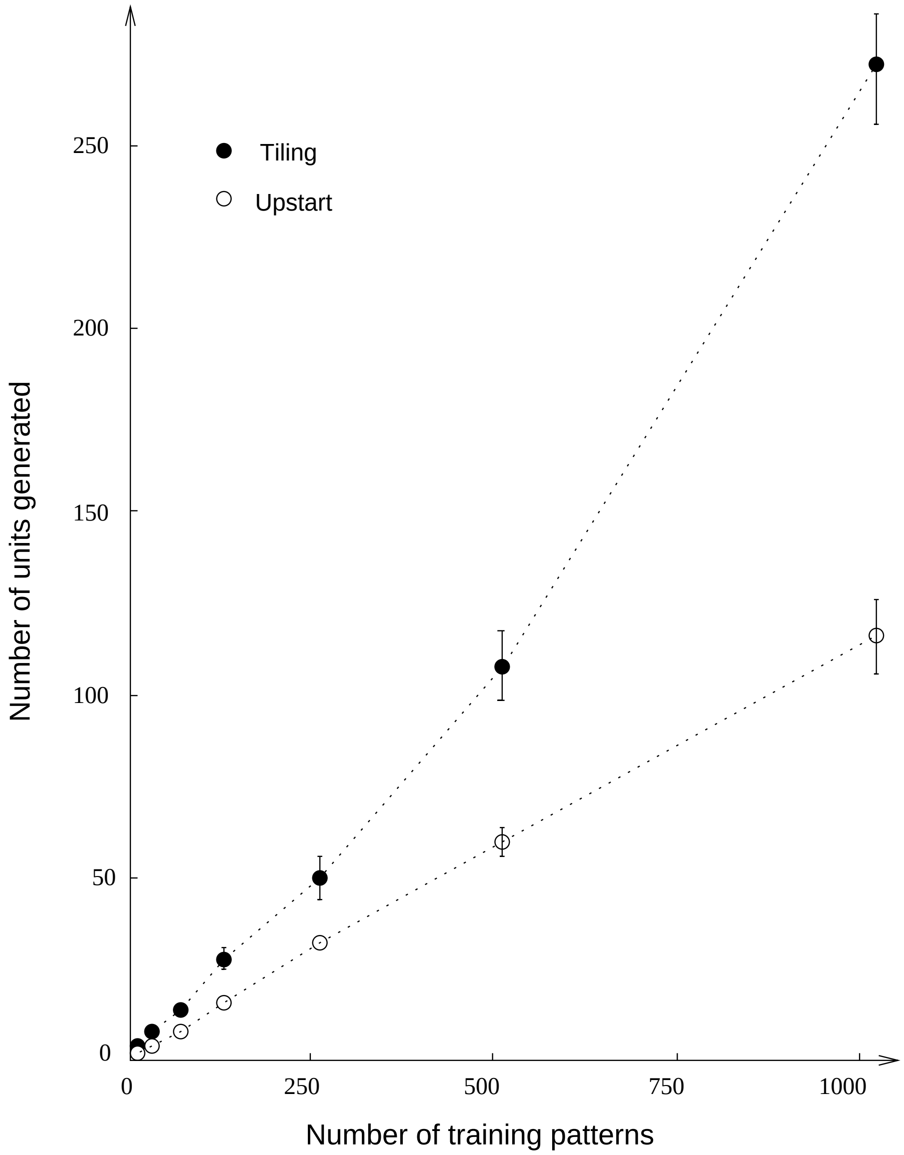

Figure 3: Number of units built vs the number of patterns (2^N) for the random mapping problem. The slope of the Upstart line is approximately 1/9. Each point is an average of 25 runs, each on a different training set.

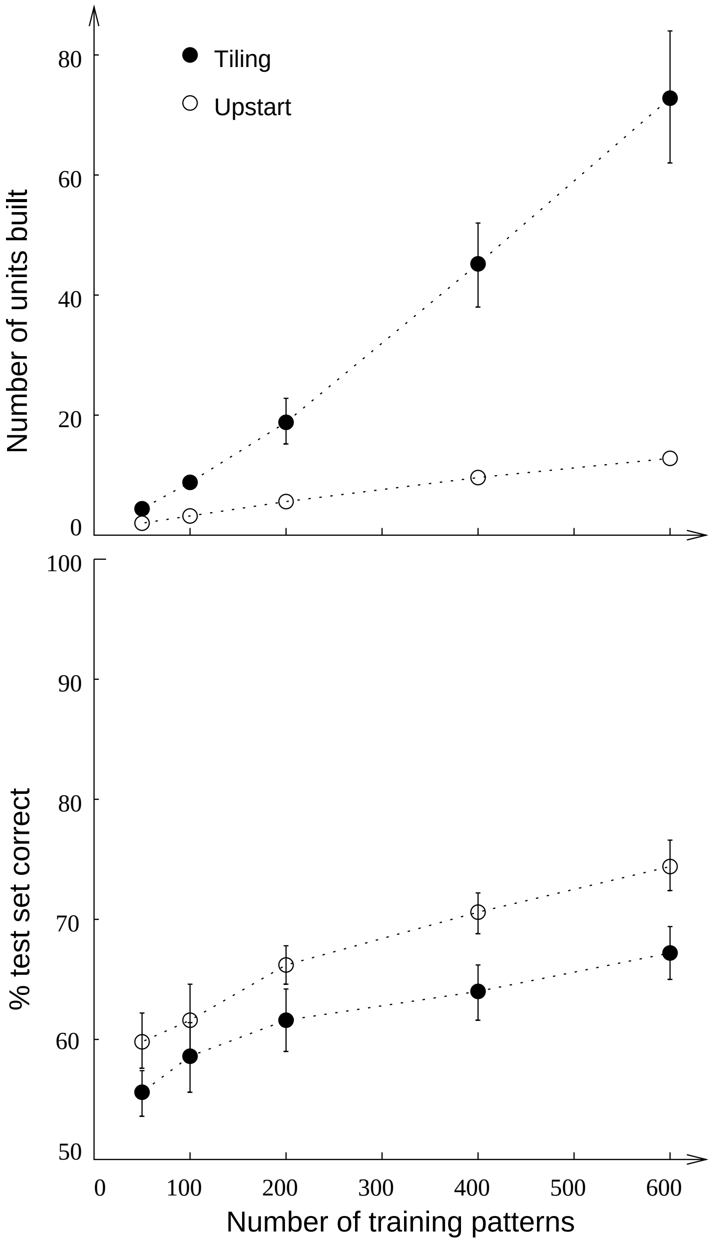

Figure 4: Performance of the method on the "2-or-more clumps" problem. The lower graph shows the % generalisation as the size of the training set is increased. Plotted above this and with the same abscissa is the size of the corresponding network. $T_0 = 4.0$. 25 runs per point, each on a different set. Where not shown, error bars are smaller than the points.

First, we are not restricted to binary branching trees. Having trained a daughter unit and connected it to the output, a new daughter can now be trained using targets derived from the *partially corrected* output, instead of the daughter, and so on until no more mistakes are made. This would build a single hidden layer. Hybrid methods, building trees of variable breadth, will also work.

Second, these algorithms can be extended to problems involving multiple output units. A good method should build considerably fewer units than would be obtained by treating each output separately (especially if the output targets are correlated); in other words, maximum mutual use should be made of hidden units. Consider the following algorithm, where steps 1 and 2 are repeated until every output unit makes no mistakes :

Start. There are no hidden units and no connections, so the output units are always OFF.

Step 1. Choose an output unit (say, the one which makes the most errors). Build the appropriate hidden unit to correct some of the mistakes being made by this output unit, as described above. Connect this new unit to *all* the output units.

Step 2. Train the weights from each unit in this enlarged hidden layer to each of the output units. Re-evaluate the numbers of errors made by each output unit.

Hence a single hidden layer is constructed.

In conclusion, the design of general purpose supervised learning algorithms for neural networks involves two important considerations: the network should succeed in correctly classifying the patterns it is trained upon, and non-trivial solutions should involve as few computational elements as possible, avoiding redundant computation. The Upstart Algorithm can build a network to implement correctly any Boolean mapping. Because each "daughter" cell makes as few errors as possible, it corrects as many "parent" errors as possible, which results in small networks. These networks are smaller than those produced by the Tiling Algorithm. In general the minimum number of units required for any given problem cannot be calculated. However, for a few special cases such as parity and the clumps problem it is known that N or fewer units are needed, and the Upstart Algorithm achieves this. In the potentially worst case where the targets are randomly assigned, m patterns can be correctly classified by approximately $m/9$ units. The basic idea can be implemented in different architectures and is extendable to the case of multiple outputs.

7 Acknowledgments

I am grateful to David Willshaw who helped greatly with the manuscript, and also David Wallace for useful comments. I particularly thank Peter Dayan for suggesting that the tree could be "squashed" into a single hidden layer, as well as for helpful discussions.

8 References

Binder,K. 1979. Monte Carlo Methods in Statistical Physics. *Topics in Current Physics,* **7** *(Berlin:Springer)*

Denker,J., Schwartz,D., Wittner,B., Solla,S., Howard,R., Jackel,L. and Hopfield,J. 1987. Large Automatic Learning, Rule Extraction and Generalization, *Complex Systems* **I**:877-922

Frean,M.R. A "Thermal" Perceptron for Efficient Linear Discrimination. *In preparation.*

Gallant,S.I. 1986a. Three Constructive Algorithms for Network Learning. *Proc. 8th Annual Conf. of Cognitive Science Soc.*

Gallant,S.I. 1986b. Optimal Linear Discriminants, *IEEE Proc. 8th Conf. on Pattern Recognition, Paris.*

Mezard,M. and J.Nadal. 1989. Learning in Feedforward Layered Networks : the Tiling Algorithm, *J.Physics A*, **22**,12:2191-2203

Minsky,M. and S.Papert. 1969. Perceptrons, *MIT Press.*

Nadal,J. 1989. Study of a Growth Algorithm for Neural Networks *International J. of Neural Systems*,**1**,1:55-59

Rosenblatt,F. 1962. Principles of Neurodynamics, *Spartan Books, New York.*

Rumelhart,D.E., Hinton,G.E., and Williams,R.J. 1986. Learning Internal Representations by error propagation. In Rumelhart,D.E., McClelland,J.L., and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume I Foundations,* MIT Press.