International Journal of Neural Systems, Vol. 5, No. 1 (March, 1994) 59–66

A CONSTRUCTIVE ALGORITHM THAT CONVERGES FOR REAL-VALUED INPUT PATTERNS

NEIL BURGESS
Department of Anatomy, University College, London WC1E 6BT, U.K.

Received 7 October 1992
Revised 4 November 1993
Accepted 8 December 1993

A constructive algorithm is presented which combines the architecture of Cascade Correlation and the training of perceptron-like hidden units with the specific error-correcting roles of Upstart. Convergence to zero errors is proved for any consistent classification of real-valued pattern vectors. Addition of one extra element to each pattern allows hyper-spherical decision regions and enables convergence on real-valued inputs for existing constructive algorithms. Simulations demonstrate robust convergence and economical construction of hidden units in the benchmark "N-bit parity" and "twin spirals" problems.

1. Introduction

"Constructive" algorithms enable the size and structure of feed-forward neural networks to be determined as part of the learning rule, avoiding the need to guess a good architecture *a priori*, see e.g. Refs. 1–3. Such algorithms tend to be faster than the traditional multi-layer perceptron architecture trained by error back-propagation[4] ("BP" in the following) by at least an order of magnitude, and succeed in problems where BP has so far failed to find a solution.[5–7] Certain constructive algorithms (e.g. "Tiling",[2] "Tower"[8] and "Upstart"[9]) can be shown to converge to zero errors during training on any consistent classification of *binary* input patterns. However, pattern classification problems typically involve real-valued inputs. In this paper we describe a constructive algorithm for which convergence is also guaranteed in this case.

Existing constructive algorithms[2,8,9] rely on a modified perceptron learning rule[10] to train linear threshold units which are added to the network one by one during learning to form, for example, binary tree[9,6] or tower[1,8] architectures. The Cascade Correlation[7] algorithm, by contrast, uses continuous units and a modified BP learning rule.[11] The power of the cascade architecture, see Fig. 1, is due to the output and each new hidden unit receiving connections from all previous hidden units as well as from the inputs. Adding hidden units results in dimensional expansion of input space until the task for the output unit becomes linearly separable. No convergence proof exists for the Cascade Correlation algorithm.

It is shown that if the cascade architecture is used with linear threshold units and the error-correcting learning rule of Upstart,[9] it will converge for any consistent classification of real-valued input vectors. We refer to this combined algorithm as a "Perceptron Cascade". Simulations indicate robust convergence on the "N-bit parity" and "twin spirals"[12] problems, constructing small networks in a short time.

Guaranteed convergence in a finite number of steps is a useful indication of the power of an algorithm; lack of guaranteed convergence inevitably leaves an algorithm open to doubts of unreliability or over-dependence on parameter values or initial conditions, as is the case with BP.[13]

Notice however that in real applications, training until the algorithm has converged to zero errors, even if possible, may not necessarily optimise the generalisation ability of the network. In fact the final size of the network should match the complexity of the task.[14] How to achieve this in the case of a Perceptron Cascade is considered briefly in Ref. 15. A convergence result ensures the possibility of reducing the number of errors on the training set to zero if necessary.

2. The Algorithm, a Perceptron Cascade

The tasks considered are consistent classifications of a finite set $\mathcal{S}$ of real-valued n-vectors ("patterns") into two classes. The classification is "consistent" in that the same pattern may appear in $\mathcal{S}$ more than once, but always has the same target. The ith element of the μth pattern is labelled v_i^μ; each pattern is assigned a "target" $t^\mu = 0$ or 1, denoting to which class it belongs. Extension of the convergence proof to more classes (more than one output unit) is straightforward.

Figure 1 shows the "cascade" architecture. Initially there is only an output ("unit 0") connected to the pattern inputs by connections with adaptable weights $\mathbf{W}(0)$. Linear threshold units are used throughout; thus on presentation of pattern $\mathbf{v}^\mu$ the output of unit 0 is $o^\mu(0)$, where:

$$o^\mu(0) = \begin{cases} 1 & \text{if } \phi^\mu(0) \geq 0\,; \\ 0 & \text{otherwise}\,, \end{cases} \qquad \phi^\mu(0) = \frac{1}{n} \sum_{i=0}^{n} W_i(0) v_i^\mu \,. \tag{1}$$

The threshold is included by having a weighted connection $W_0(0)$ to an extra input v_0^μ which is 1 for all μ. Notice that the weights $\mathbf{W}(0)$ define an $n-1$ dimensional hyperplane in the n-dimensional input space:

$$(1, x_1, x_2, \ldots, x_n) \cdot \mathbf{W}(0) = 0\,, \tag{2}$$

that separates the two regions of input space for which the output is 0 or 1 ($\mathbf{x}$ is a general n-vector, $\mathbf{W}(0) = (W_0(0), W_1(0), \ldots)$ and $\cdot$ is the scalar product). The connections to the output are trained to classify the patterns in $\mathcal{S}$ according to the targets t^μ using the Pocket learning rule,[10] see below.

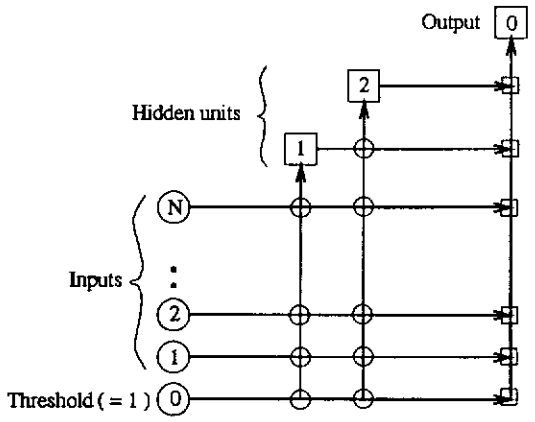

Fig. 1. The cascade architecture[7]; connections marked □ are relearnt after the addition of each hidden unit, those marked o are learnt once and frozen.

After training the output, the patterns in $\mathcal{S}$ can be partitioned into four subsets: those for which the output is "correctly on", "correctly off", "wrongly on" or "wrongly off". These subsets are labelled $\mathcal{S}_{\text{con}}(0)$, $\mathcal{S}_{\text{coff}}(0)$, $\mathcal{S}_{\text{won}}(0)$ and $\mathcal{S}_{\text{woff}}(0)$, the number of patterns in each subset is labelled $N_{\text{con}}(0)$, $N_{\text{coff}}(0)$, $N_{\text{won}}(0)$ and $N_{\text{woff}}(0)$. Hidden units (numbered 1, 2...) are added one by one; each receives connections from the inputs and all previously added hidden units, and is connected to the output. A hidden unit is trained to be "on" either for patterns in $\mathcal{S}_{\text{woff}}(0)$ — a "wrongly off corrector" — or for patterns in $\mathcal{S}_{\text{won}}(0)$ — a "wrongly on corrector" — as in Upstart.[9]

When a hidden unit, the kth, say, is added to the network it is connected to the output with a connection weight $W_{n+k}(0)$ and *all* of the connections to the output are retrained. This allows the output unit to avoid some of the errors made before the addition of that hidden unit, e.g. by making $W_{n+k}(0)$ strongly positive if it is a "wrongly off corrector". If the kth hidden unit is to be a "wrongly off corrector", its training set $\mathcal{S}(k)$ is:

$$\mathcal{S}(k) = \mathcal{S}_{\text{woff}}(0) \cup \mathcal{S}_{\text{coff}}(0) \cup \mathcal{S}_{\text{won}}(0) \tag{3}$$

with targets $t^\mu(k) = 1$ for the patterns in $\mathcal{S}_{\text{woff}}^{k-1}(0)$ and $t^\mu(k) = 0$ for those in $\mathcal{S}_{\text{coff}}^{k-1}(0) \cup \mathcal{S}_{\text{won}}^{k-1}(0)$. As in Upstart $\mathcal{S}_{\text{con}}(0)$ is omitted from the training set; another positive term in $\phi^\mu(0)$ in (1) will not change an output that is "correctly on" anyway. Similarly the training set for a "wrongly on corrector" is:

$$\mathcal{S}(k) = \mathcal{S}_{\text{won}}(0) \cup \mathcal{S}_{\text{con}}(0) \cup \mathcal{S}_{\text{woff}}(0) \tag{4}$$

with targets $t^\mu(k) = 1$ for the patterns in $\mathcal{S}_{\text{won}}(0)$ and 0 for those in $\mathcal{S}_{\text{con}}(0) \cup \mathcal{S}_{\text{woff}}(0)$.

The connection weights to a hidden unit are trained once only and then frozen. The order in which new hidden units of each type are created is not crucial, we simply say that the next hidden unit will be trained to correct the majority type of output error.

2.1. *Training*

All training uses the Pocket algorithm; thus for the output (before the addition of any hidden units) the patterns in $\mathcal{S}$ are presented N_{cycles} times in random

order. On presentation of a pattern $\mathbf{v}^\mu$ the weight $W_i(0)$ is changed by an amount:

$$\Delta W_i(0) = \alpha(t^\mu - o^\mu(0))v_i^\mu\,, \quad i = 0, \ldots, n\,, \qquad (5)$$

where α is some positive constant. A copy is kept ("in your pocket") of the set of weights that has remained unchanged for the most presentations during training. These weights will result in the minimum possible number of errors on $\mathcal{S}$ with probability 1 in the limit of large N_{cycles}, see Ref. 10.

A hidden unit k is connected to all previous hidden units by weights $W_{n+1}(k)$ to $W_{n+k-1}(k)$, so (1) and (5) include extra terms. In (1), $\phi^\mu(k)$ becomes:

$$\phi^\mu(k) = \frac{1}{n+k-1}\left(\sum_{i=0}^{n} W_i(k)v_i^\mu + \sum_{i=1}^{k-1} W_{n+i}(k)o^\mu(i)\right), \qquad (6)$$

and (5) includes the extra terms:

$$\Delta W_i(k) = \alpha(t^\mu(k) - o^\mu(k))o^\mu(i) \quad \text{for } i = n+1 \text{ to } n+k-1\,. \qquad (7)$$

Notice that the response of the output (i.e. the makeup of $\mathcal{S}_{\text{won}}(0)$, $\mathcal{S}_{\text{woff}}(0)$, $\mathcal{S}_{\text{con}}(0)$, $\mathcal{S}_{\text{coff}}(0)$) changes after the addition of each hidden unit: the training set for hidden unit k in (3) and (4) refers to the response of the output unit after the addition of hidden units 1 to $k-1$.

To aid convergence (see below), we introduce two further steps:

(i) If, after the addition of hidden unit k, the retraining of the set of weights to unit 0 does not reduce the number of errors committed on $\mathcal{S}$, then the previous set of weights is kept and the weight $W_{n+k}(0)$ from the new unit is set to zero (this unit is still useful: it provides another input to subsequent hidden units so that they can do better, and subsequent retraining of the output weights can assign a non-zero value to $W_{n+k}(0)$).

(ii) If a training set $\mathcal{S}(k)$ contains a small minority of patterns with target 1, then the Pocket algorithm may find connection weights such that unit k is "off" for all patterns in $\mathcal{S}(k)$, as this may give the minimum possible number of errors. If this occurs then it is retrained on a "balanced" training set $\mathcal{S}'(k)$ in which patterns with target 1 and 0 are present in equal numbers (formed by repeating those with target 1).

3. Results

In this section we examine the consequences of using the Pocket algorithm to train the output and hidden units, *in the limit of large* N_{cycles} where it is guaranteed to find an optimal set of connection weights (i.e. minimising the number of errors on the training set). Before the addition of a hidden unit k the number of output errors is denoted by $E(0)$ where:

$$E(0) = N_{\text{woff}}(0) + N_{\text{won}}(0)\,, \qquad (8)$$

and denoted by $E^+(0) = N^+_{\text{woff}}(0) + N^+_{\text{won}}(0)$ afterwards.

The set of vertex points of the convex hull of $\mathcal{S}$ is denoted by $\mathcal{V}(\mathcal{S})$. The patterns in $\mathcal{V}(\mathcal{S})$ can all be separated from all other distinct patterns in $\mathcal{S}$ by an $N-1$ dimensional hyper-plane in input space.

Note 1

If $\mathbf{v}^* \in \mathcal{V}(\mathcal{S})$ connection weights from the inputs exist such that a perceptron is "on" for $\mathbf{v}^*$ and "off" for all other distinct elements in $\mathcal{S}$, see Ref. 2 for the case of binary inputs. Similarly, connection weights exist such that a perceptron has output 0 for $\mathbf{v}^*$ and 1 for all other distinct patterns in $\mathcal{S}$.

Proof

Let $\mathbf{x}$ be a general n-vector and let $\mathbf{x}\cdot\mathbf{n} = d$ be a plane separating $\mathbf{v}^*$ from all other distinct patterns in $\mathcal{S}$. We have, without loss of generality,

$$\mathbf{v}^*\cdot\mathbf{n} > \mathbf{u}\cdot\mathbf{n} \quad \text{for all } \mathbf{u}\in\mathcal{S},\ \mathbf{u}\neq\mathbf{v}^*\,, \qquad (9)$$

(if $\mathbf{v}^*\cdot\mathbf{n} < \mathbf{u}\cdot\mathbf{n}$ for all $\mathbf{u}\in\mathcal{S}$, $\mathbf{u}\neq\mathbf{v}^*$ then shifting the origin to the centroid of $\mathcal{S}$ ensures (9)). Then a perceptron with connection weights $\mathbf{W}(0) = (-d, n_1, n_2, \ldots, n_n)$ will be "on" for only pattern $\mathbf{v}^*$ in $\mathcal{S}$. Similarly a perceptron with weights $\mathbf{W}(0) = (d, -n_1, -n_2, \ldots, -n_n)$ will be "off" for only pattern $\mathbf{v}^*$ in $\mathcal{S}$. □

We consider the addition to the network of one hidden unit (unit k). For simplicity suppose that unit k is a "wrongly off corrector". The number of errors unit k makes on its training set, $\mathcal{S}(k)$, is denoted by $E(k)$. A hidden unit k is "successful" if $E(k)$ is less than the number of patterns with target 1 in $\mathcal{S}(k)$, i.e. if $E(k) < N_{\text{woff}}(0)$.

Result 1

If any of the patterns in $\mathcal{S}(k)$ with target 1 are in $\mathcal{V}(\mathcal{S}(k))$ then unit k will be "successful".

Proof

Note 1 applied to unit k implies that connection weights exist such that unit k is "correctly on" for only one distinct pattern in $\mathcal{S}(k)$, i.e. weights exist which produce at most $N_{\text{woff}}(0)-1$ errors. Thus in the limit of large N_{cycles} the Pocket algorithm will find weights at least as good, i.e. weights such that $E(k) < N_{\text{woff}}(0)$. □

Result 2

If unit k is "successful" then the number of output errors will decrease after retraining the weights to the output.

Proof

The connection of unit k to unit 0 with a connection weight $W_{n+k}(0)$ where:

$$W_{n+k}(0) > \sum_{i=0}^{n+k-1} |W_i(0)|, \tag{10}$$

(leaving the weights $W_0(0)$ to $W_{n+k-1}(0)$ unchanged) would correct $N_{\text{con}}(k)$ "wrongly off" errors, and create at most $N_{\text{won}}(k)$ new "wrongly on" errors, i.e. we would have:

$$E^+(0) \leq E(0) + N_{\text{won}}(k) - N_{\text{con}}(k). \tag{11}$$

Since unit k can only be "correctly on" or "wrongly off" for the patterns in $\mathcal{S}(k)$ with target 1, we have:

$$N_{\text{woff}}(0) = N_{\text{con}}(k) + N_{\text{woff}}(k) \tag{12}$$

thus we have (from (8) and (12)) that: $N_{\text{con}}(k) > N_{\text{woff}}(k)$, so that (11): $E^+(0) < E(0)$. Thus connection weights exist such that $E^+(0) < E(0)$. Again, the Pocket algorithm will find connection weights to the output that do at least this well.[a] □

Note 2

An unsuccessful hidden unit k has output 1 for at least one pattern in $\mathcal{S}(k)$.

Proof

If this is not the case after training, unit k is retrained on a "balanced" training set $\mathcal{S}'(k)$ — see step (ii) in Sec. 2.1. Because unit k was unsuccessful, $\mathcal{V}(\mathcal{S}(k))$ and hence $\mathcal{V}(\mathcal{S}'(k))$ contain only patterns with target 0; see Result 1. Thus connection weights exist such that unit k is "correctly off" for at least one pattern (Note 1), i.e. weights such that unit k makes at most $N_{\text{coff}}(0) + N_{\text{won}}(0) - 1$ errors. Since an output of 0 for all patterns in $\mathcal{S}'(k)$ implies $N_{\text{coff}}(0) + N_{\text{won}}(0)$ errors, such connection weights will not be found by the Pocket algorithm.[a] □

Result 3

If hidden unit k is unsuccessful and the next hidden unit $(k+1)$ is trained on the same set of patterns and targets, it will make fewer errors than hidden unit k.

Proof

Notice that $\mathcal{S}(k+1) = \mathcal{S}(k)$ and $E(k) \geq N_{\text{woff}}(0)$. We form two subsets of $\mathcal{S}(k)$, $\mathcal{S}_0(k)$ and $\mathcal{S}_1(k)$, consisting of those patterns for which the output of unit k is 0 or 1 respectively, so

$$\mathcal{S}_1(k) = \mathcal{S}_{\text{con}}(k) \cup \mathcal{S}_{\text{won}}(k). \tag{13}$$

Since $\mathcal{S}_1(k)$ is non-empty (Note 2), consider a pattern $\mathbf{v}^* \in \mathcal{V}(\mathcal{S}_1(k))$. There are two cases:

1. If $\mathbf{v}^* \in \mathcal{S}_{\text{con}}(k)$ we label by $\mathbf{W}^*$ the connection weights from the inputs such that unit $k+1$ is "on" for $\mathbf{v}^*$ and "off" for all other distinct patterns in $\mathcal{S}_1(k)$ (Note 1). Now, if the weights to unit $k+1$ are similar to $\mathbf{W}^*$ but have a large positive connection to unit k and a correspondingly higher threshold, i.e.:

$$\mathbf{W}(k+1) = (W_0^* - L, W_1^*, W_2^*, \ldots, W_n^*, 0, 0, \ldots, L) \tag{14}$$

where

$$L > \max_{\mathbf{v}^\mu \in \mathcal{S}(k)} \left(\left| \sum_{i=0}^{n} W_i^* v_i^\mu \right| \right), \tag{15}$$

then unit $k+1$ will be "on" for $\mathbf{v}^*$ and "off" for all other distinct patterns in $\mathcal{S}(k)$, i.e. it will make at most $N_{\text{woff}}(0) - 1$ errors.

2. If $\mathbf{v}^* \in \mathcal{S}_{\text{won}}(k)$ we label by $\mathbf{W}^*$ the connection weights from the inputs such that unit $k+1$ is "off" for $\mathbf{v}^*$ and "on" for all other distinct patterns in $\mathcal{S}_1(k)$ (Note 1). Now, if we define $\mathbf{W}(k+1)$ in terms of this $\mathbf{W}^*$ as in (14), then unit $k+1$ will be "off" for the patterns in $\mathcal{S}_0(k)$ and for $\mathbf{v}^*$, and "on" for the remaining patterns in $\mathcal{S}_1(k)$, i.e. it will make at most $E(k) - 1$ errors.

[a] In the limit of large N_{cycles}.

In either case weights $\mathbf{W}(k)$ exist such that $E(k+1) < E(k)$, so the Pocket algorithm will find weights at least this good.[a] □

The above results apply similarly if unit k is a "wrongly on corrector", except that the number of patterns with target 1 in $\mathcal{S}(k)$ is $N_{\text{won}}(0)$, the number of patterns with target 0 is $N_{\text{woff}}(0) + N_{\text{con}}(0)$, and its connection to the output (see Result 2) should be large and negative (i.e. less than $-W_{n+k}(0)$ in (10)).

4. Convergence

The connection weights and number errors of the output are denoted $E(0)$ and $\mathbf{W}(0)$ before the addition of a hidden unit and $E^+(0)$ and $\mathbf{W}^+(0)$ after. Using the results of the previous section, we see that an added hidden unit k is either:

1. successful, in which case (Result 2) $E^+(0) < E(0)$, or
2. unsuccessful, in which case
   a) the retrained output weights $\mathbf{W}^+(0)$ are such that $E^+(0) < E(0)$, or
   b) the previous output weights $\mathbf{W}(0)$ are kept (see Sec. 2.1 step ii) and hidden unit $k+1$ will make fewer errors than unit k (Result 3): $E(k+1) < E(k)$.

Notice that only a finite number of unsuccessful hidden units can be added in succession (case 2b) since once a "wrongly off corrector" makes fewer than $N_{\text{woff}}(0)$ errors it is successful ($N_{\text{won}}(0)$ for a "wrongly on corrector"), and that when a successful hidden unit is added the number of output errors decreases.

Thus we have convergence in the sense that the number of output errors will decrease to zero after the addition of a finite number of hidden units, with probability 1 in the limit of large N_{cycles}.

The proof implies only a very weak upper bound on the size of the network: the output unit may initially make $p/2$ errors, each successful hidden unit could correct only one error and a succession of $2n-1$ unsuccessful hidden units is possible, i.e. the network could comprise Np units in the worst case. However simulations indicate that in practice much smaller networks are formed than are indicated by this bound, see Sec. 5.

In the case of many outputs (each connected to all hidden units), each will converge to zero errors if, for example, we train the "next" hidden unit as above, using the output unit that makes the most errors to generate its training set.

4.1. *Addition of a higher order term*

If we add an extra element:

$$v_{n+1}^{\mu} = \sum_{i=1}^{n} (v_i^{\mu})^2 \tag{16}$$

to each pattern, the training set is projected onto the surface of a paraboloid in $n+1$ dimensions (so that $\mathcal{S} = \mathcal{V}(\mathcal{S})$). This immediately allows convergence as all added hidden units will be successful (Result 1) so that the number of output errors will decrease with each hidden unit added (Result 2). In this case steps (i) and (ii) in Sec. 2.1 are unnecessary; indeed, existing constructive algorithms (e.g. Refs. 2, 8, 9) will converge with this set-up, see also Ref. 16.

In terms of the n-dimensional input space of the original patterns the "decision region" of each unit becomes a hyper-sphere rather than a semi-finite hyper-plane. In particular the connection weights:

$$\mathbf{W}(k) = \left(\varepsilon^2 - \sum_{i=1}^{n} (x_i)^2,\ 2x_1,\ 2x_2, \ldots,\ 2x_n,\ -1\right) \tag{17}$$

ensure that, when a pattern $\mathbf{v}^{\mu}$ is presented, unit k is on only if $\mathbf{v}^{\mu}$ is within a distance ε of $\mathbf{x}$ in the original n-dimensional input space, i.e.

$$o^{\mu}(k) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} (v_i^{\mu} - x_i)^2 \le \varepsilon^2\,; \\ 0 & \text{otherwise}. \end{cases} \tag{18}$$

If $\varepsilon = 0$ in (18) then unit k will be "on" only for pattern $\mathbf{v}^{\mu}$, i.e. each pattern can be separated from all the other distinct patterns in $\mathcal{S}$. An appropriate normalisation of patterns can also be enough to change the shape of decision regions and ensure that $\mathcal{S} = \mathcal{V}(\mathcal{S})$, e.g. a normalisation onto the surface of a hyper-sphere (see Refs. 17, 18).

5. Simulations; Convergence with Finite N_{cycles}

To check the convergence power of the perceptron cascade when the value of N_{cycles} is strictly limited, we ran the algorithm on two benchmark problems, using $N_{\text{cycles}} = 500$.

In these simulations we used the "Thermal" perceptron learning rule, as it appears to be more efficient than the Pocket learning rule.[19] Each time

a pattern μ is presented, the weights to unit k are changed according to:

$$\Delta W_i(k) = \alpha \frac{T}{T_0}(t^\mu(k) - o^\mu(k))v_i^\mu(k) \exp\left(\frac{-|\phi_k^\mu|}{T}\right). \quad (19)$$

T is a "temperature" parameter that decreases from T_0 to 0 during the 500 passes through the training set, $\alpha = 0.1$. All of the weights are initially zero; there is no dependence on random initial weights, although the result does depend on the (random) order of presentation of patterns.

Choosing a good value of T_0 is potentially a problem; it should be of the same order of magnitude as the range of values of ϕ_k^μ (see Ref. 17). Since it is possible for ϕ_k^μ to vary considerably for the training of different units, we calculate the average value of $|\phi_k^\mu|$ over the set of patterns ($\langle|\phi_k^\mu|\rangle_\mu$), over each cycle through a training set, at the end of each cycle T_0 (initially 1) is set to be $2\langle|\phi_k^\mu\rangle_\mu$ [setting $T_0 = (2T_0 + 2\langle|\phi_k^\mu|\rangle_\mu)/3$ avoids oscillatory behaviour]. The "temperature" T is then γT_0, where γ (initially 1) decreases linearly with each cycle to reach 0 after 500 cycles.

5.1. *N-bit parity*

In the N-bit parity problem, patterns of N binary inputs must be classified according to whether the number of "on" inputs is even or odd. All 2^N patterns were used in training. In solving this task for $N = 2$ to 8, the tower algorithm is quoted as growing $N/2$ hidden units,[8] the more efficient of the tree structured algorithms[9,6] grow $N - 1$, and other "tree" algorithms[20,21] grow more. All require more than 500 cycles through the training set for the larger values of N. The Cascade Correlation algorithm grows (1, 1, 2, 2–3, 3, 4–5, 5–6) hidden units respectively for $N = 2$ to 8, using Gaussian hidden units.[7] Cascade Correlation required fewer cycles through the training set than the above algorithms. However, the learning rule[11] is a second order heuristic in which each step is more computationally intensive than the perceptron-like learning rules above, and performance depends on the adjustment of many parameters.[22]

Perceptron Cascade grew (1, 2, 2, 2, 3, 3, 4, 4, 5) hidden units respectively in all tests for $N = 2$ to 10, i.e. in terms of size it was at least as efficient as existing constructive algorithms.

5.2. *Twin spirals*

The twin spirals[12] problem (separating points from two interlocking spirals, see Fig. 2) requires a highly nonlinear classification of real-valued patterns. Multi-layer perceptrons with the standard error back propagation learning rule[4] and the perceptron based constructive algorithms above do not seem able to solve this problem.[7,16]

The published figures for Cascade Correlation are for simulations in which a pool of 8 hidden units is trained at each step (using different initial sets of connection weights); the one with the best correlation score is added to the network. Stereographic projection of the 2-D input patterns onto the surface of a sphere (using an extra input) enables Upstart to solve the problem.[16]

Table 1 shows the number of hidden units created in 10 simulations of Perceptron Cascade, with and without an extra input (projecting the input patterns onto a paraboloid) or a "pool" of units to choose from. Figures for Cascade Correlation (using sigmoidal units this time) and Upstart are given for

Fig. 2. The two sets of points in the twin spirals problem (left), and the decision regions for two networks after training; middle — no extra input (30 hidden units), right — extra input (6 hidden units). There are no misclassifications.

Table 1. Number of hidden units created by Perceptron_Cascade in solving the twin spirals problem (10 trials). Figures for Cascade Correlation and Upstart are given for comparison. Parentheses indicate that a pool of 8 units was trained each time and the best chosen.

Number of Units	Perceptron Cascade		Cascade Correlation	Upstart With Extra Input
	Without Extra Input	With Extra Input		
fewest	25 (21)	7 (6)	(12)	10
most	48 (35)	39 (21)	(19)	1263
average	34.7 (26.7)	17.8 (11.2)	(15.2)	91.4

comparison; the average for Upstart is distorted by a few very large numbers. In terms of creating small nets Perceptron Cascade does worse than Cascade Correlation (although its learning rule is considerably simpler). With the higher order term allowing circular decision regions its performance is better than Cascade Correlation (without a similar extra unit) and Upstart (with a similar extra unit). Figure 2 shows the actual decision regions of two of the networks generated.

6. Conclusion

The strength of the Perceptron Cascade algorithm is that the simple Thermal perceptron learning rule (compared to the "quick-propagation" used in Cascade Correlation, see Refs. 11, 22) results in robust convergence, even with real-valued input patterns, with an economy of hidden units. Convergence is formally guaranteed for any consistent classification of patterns by using the Pocket learning rule with enough cycles through the training set. However, the restriction to binary outputs prevents many possible applications (e.g. time-series prediction).

For good generalisation it is necessary to restrict the size of the network to match the task,[14,23] rather than continuing until zero errors are committed on the training set (although the number of residual errors should be controllable: not determined by an algorithm's lack of convergence). This may be possible using a result of Vapnik[23] (see Ref. 15), however the nature of a constructive algorithm implies that changes to its complexity occur in steps rather than the smooth variation possible with "pruning" methods (see e.g. Ref. 24). In terms of complexity it may be fruitful to consider particularly the patterns near to decision boundaries, rather than all of the patterns equally; see Refs. 26, 27.

Clearly the shape of the decision regions implemented by each unit on its input is important in terms of generalisation ability. This is obviously true of using circular decision regions in the twin spirals problem. The simple way in which decision regions can be changed by the addition of a unit, or units,[25] representing a higher order (than linear) transformation of the input patterns makes it a useful option for improving generalisation, given *a priori* knowledge about the nature of a task. It is possible to guarantee the convergence of existing constructive algorithms with the addition of one of this type of unit, see Sec. 4.1. An interesting algorithm for determining the transfer function of each added unit (as well as its connection weights) during training is presented in Ref. 28, demonstrating the usefulness of considering the decision region of each unit by forming very smooth decision regions for the twin spirals problem. A review of the use of higher order terms in conventional feed-forward neural networks can be found in Ref. 29.

The way constructive algorithms use a greedy piecemeal strategy (each unit independently tries to correct the maximum number of errors) allows for greater speed and reliability of convergence than BP, and obviates the need to guess a good architecture *a priori*. However, this approach may lead to a more fragmented decision region for the network as a whole than a more global strategy, cf. Fig. 2 and Refs. 7, 28. Initial tests of the ability of Perceptron Cascade to generalise (as opposed to the proof of its ability to learn the set of training examples) have shown good results on a 3-D object recognition task[18] and in some small tests resembling real pattern recognition problems.[15] In future work, generalisation ability will be tested on large benchmark pattern classification problems, and compared with that predicted by the prescription in Ref. 15.

Acknowledgments

I should like to thank Marcus Frean and Paolo Ferragina for useful discussions, and I am grateful for the hospitality of Mario Notturno Granieri and the IBM Rome Scientific Centre.

References

1. S. I. Gallant, "Three constructive algorithms for network learning," *Proc. 8th Annual Conf. of Cognitive Science Society* (Amherst MA, 1986) 652–660.
2. M. Mezard and J.-P. Nadal, "Learning in feedforward layered networks: the Tiling algorithm," *J. Phys.* **A22** (1989), 2191–2203.
3. P. Ruján, "Learning and architectures of neural networks," in R. M. J. Cotterill, ed., *Models of Brain Function* (Cambridge University Press, 1989), pp. 517–537.
4. D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning internal representations by error propagation," in D. E. Rumelhart and J. L. McClelland, eds. *Parallel Distributed Processing 1* (MIT Press, 1986), pp. 151–193.
5. S. I. Gallant, "Perceptron-based learning algorithms," *IEEE Trans. on Neural Networks* **1** (1990), 179–191.
6. J. A. Sirat and J.-P. Nadal, "Neural trees: a new tool for classification," *Network* **1** (1990), 423–438.
7. S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in D. S. Touretzky, ed., *Neural Information Processing Systems 2* (Morgan Kaufmann, 1990), pp. 524–532.
8. J.-P. Nadal, "Study of a growth algorithm for a feedforward network," *Int. J. Neural Syst.* **1** (1989) 55–59.
9. M. R. Frean, "The Upstart algorithm: a method for constructing and training feedforward neural networks," *Neural Computation* **2** (1990), 198–209.
10. S. I. Gallant, "Optimal linear discriminants," *Proc. 8th Int. Conf. on Pattern Recognition* (IEEE Computer Soc. Press, 1986), 849–852.
11. S. E. Fahlman, "Faster-learning variations on back-propagation: an empirical study," in D. S. Touretzky, G. E. Hinton and T. Sejnowsky, eds., *Proceedings of the 1988 Connectionist Models Summer School* (Morgan Kaufmann, 1988), pp. 38–51.
12. K. J. Lang and M. J. Witbrock, "Learning to tell two spirals apart," in D. S. Touretzky, G. E. Hinton and T. Sejnowsky, eds., *Proceedings of the 1988 Connectionist Models Summer School* (Morgan Kaufmann, 1988), pp. 52–59.
13. J. F. Kolen and J. B. Pollack, "Backpropagation is sensitive to initial conditions," in R. P. Lippmann, E. J. Moody and D. S. Touretzky, eds., *Neural Information Processing Systems 3* (Morgan Kaufmann, 1991), pp. 860–868.
14. I. Guyon, V. Vapnik, B. Boser, L. Bottou and S. Solla, "Structural risk minimization for character recognition," in E. J. Moody, S. J. Hanson and R. P. Lippmann, eds., *Neural Information Processing Systems 4* (Morgan Kaufmann, 1992), pp. 471–479.
15. N. Burgess, S. Di Zenzo, P. Ferragina and M. Notturno Granieri, "The generalization of a constructive algorithm in pattern classification problems," in O. Benhar, C. Bosio, P. Del Giudice and M. Grandolfo, eds., *Proc. of the 2nd Workshop on Neural Networks: From Biology to High Energy Physics* (Singapore: World Scientific, 1993), pp. 65–70.
16. J. Saffery and C. Thornton, "Using stereographic projection as a pre-processing technique for Upstart," *Proc. International Joint Conference on Neural Networks of the IEEE, July,* II (1991), 441–446.
17. M. R. Frean, "Small Nets and Short Paths: Optimising Neural Computation," PhD thesis, Edinburgh University (1990).
18. N. Burgess, M. Notturno Granieri and S. Patarnello, "3-D object classification: application of a constructive algorithm," *Int. J. of Neural Syst.* **2** (1992), 275–282.
19. M. R. Frean, "A thermal perceptron learning rule," *Neural Computation* **4** (1992), 946–957.
20. I. K. Sethi, "Entropy nets: from decision trees to neural networks," *Proc. of the IEEE* **78** (1990), 1605–1613.
21. R. P. Brent, "Fast training algorithms for multilayer neural nets," *IEEE Trans. on Neural Networks* **2** (1991), 346–354.
22. J. Yang and V. Honavar, "Experiments with the cascade-correlation algorithm," *Proc. of the 4th UNB AI Symposium*, Fredericton, Canada (1991), pp. 369–380.
23. V. Vapnik, "Principles of risk minimization for learning theory," in E. J. Moody, S. J. Hanson and R. P. Lippmann, eds., *Neural Information Processing Systems 4* (Morgan Kaufmann, 1992), pp. 831–839.
24. B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: optimal brain surgeon," in S. J. Hanson, J. D. Cowan and C. L. Giles, eds., *Neural Information Processing Systems 5* (Morgan Kaufmann, 1993), pp. 164–171.
25. D. F. Specht, "Generation of polynomial discriminant functions for pattern recognition," *IEEE Transactions on Electronic Computers* **16** (1967), 308–319.
26. E. I. Chang and R. P. Lippman, "A boundary hunting radial basis function classifier which allocates centers constructively," in S. J. Hanson. J. D. Cowan and C. L. Giles, eds., *Neural Information Processing Systems 5* (Morgan Kaufmann, 1993), pp. 139–146.
27. I. Guyon, B. Boser and V. Vapnik, "Automatic capacity tuning of very large VC-dimension classifiers," in S. J. Hanson, J. D. Cowan and C. L. Giles, eds., *Neural Information Processing Systems 5* (Morgan Kaufmann, 1993), pp. 147–155.
28. J.-N. Hwang, S.-S. You, S.-R. Lay and I.-C. Jou, "What's wrong with a cascaded correlation learning network: a projection pursuit learning perspective," *IEEE Trans. on Neural Networks* (1993), submitted.
29. J. Ghosh and Y. Shin, "Efficient higher-order neural networks for classification and function approximation," *Int. J. of Neural Syst.* **3** (1992), 323–350.