International Journal of Neural Systems, Vol. 3, (Supp. 1992) 000–000
Proceedings of the Neural Networks from Biology to High Energy Physics Workshop
© World Scientific Publishing Company

THE GENERALIZATION OF A CONSTRUCTIVE ALGORITHM IN PATTERN CLASSIFICATION PROBLEMS

Neil Burgess,
Department of Anatomy, University College London
London WC1E 6BT, UK

Silvano Di Zenzo, Paolo Ferragina and Mario Notturno Granieri
IBM Rome Scientific Center, Viale Oceano Pacifico 171
00144 Rome, Italy

Received (to be inserted
Revised by Publisher)

The use of a constructive algorithm for pattern classification is examined. The algorithm, a 'Perceptron Cascade', has been shown to converge to zero errors whilst learning any consistent classification of *real-valued* pattern vectors (Burgess, 1992). Limiting network size and producing bounded decision regions are noted to be important for the generalization ability of a network. A scheme is suggested by which a result on generalization (Vapnik, 1992) may enable calculation of the optimal network size. A fast algorithm for principal component analysis (Sirat, 1991) is used to construct 'hyper-boxes' around each class of patterns to ensure bounded decision regions. Performance is compared with the Gaussian Maximum Likelihood procedure in three artificial problems simulating real pattern classification applications.

1. Introduction

Neural networks have been increasingly applied to pattern recognition problems over the last few years. Typically multi-layer perceptrons, using the error back-propagation ('BP') learning rule[1], are trained to classify pattern vectors using a 'training set' of labelled examples of patterns from each class[2]. An early comparison of the performance of these classifiers showed that they compared well with conventional classifiers[3]. However, this approach has some disadvantages. Guessing a static architecture *a priori* is problematic in that too small a network may not be capable of implementing the desired mapping and too large a network may fail to generalize well. Further, the performance of BP tends to depend heavily on choice of a good architecture, it may be computationally expensive, or may fail to converge at all[4,5].

Another approach is that of 'constructive' algorithms[6,7]. These are algorithms in which the network is constructed during learning by successive addition of perceptron-like units. Convergence to zero errors during training on any consistent classification of *binary* pattern vectors can be guaranteed for these algorithms[7]. The 'Cascade Correlation' algorithm[5] is also 'constructive', but relies on a BP-like learning rule and, although seemingly more efficient than the above algorithms at separating highly interlocking *real-valued* pattern vectors, it has no convergence proof. These constructive algorithms tend to be at least an order of magnitude faster than multi-layer perceptrons using BP[8,4,5].

In many real applications unbounded decision regions, such as the typical semi-spaces pro-

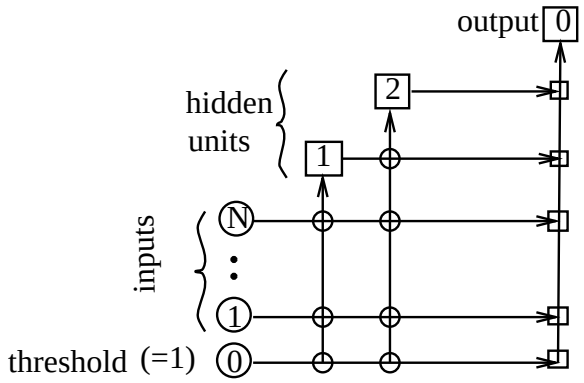

Fig. 1. The cascade architecture[5]; connections marked □ are relearnt after the addition of each hidden unit, those marked ○ are learnt once and frozen.

duced by a perceptron, are impractical because 'outliers' or unanticipated events tend to be positively classified into an incorrect class rather than being rejected. We use the Preceptron Cascade algorithm[9] to address the problems of convergence for *real-valued* inputs, optimising generalisation as a function of network size and producing bounded decision regions, with the hope that will become a useful tool for pattern classification, see also Ref. 10.

2. A Perceptron Cascade

The Perceptron Cascade ('PC') algorithmRef. 9 combines the cascade architecture[5] and perceptron-like hidden units with the error-correcting roles of Upstart[11]. Convergence to zero errors for any consistent classification of real-valued pattern vectors is proved in Ref. 9. The algorithm converges with an economy of hidden units on 'hard' tasks such as n-bit parity ($n = 2, .., 10$) and the 'twin-spirals' problem[12], using a perceptron-like learning algorithm[11] with only 500 passes through the set of training examples. A brief description follows, in which we consider a consistent classification 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.

Starting with only an output unit (unit 0), hidden units are added one by one. Each hidden unit is connected to the inputs and to all previously added hidden units; connection weights to it are trained when it is added to the network and then frozen. The output unit is connected to the inputs and to all hidden units by weights $\vec{W}(0)$ which are re-trained after each addition. See Fig. 1. Linear threshold units are used throughout; thus on presentation of pattern $\vec{v}^{\mu}$ the output of unit 0 is $o^{\mu}(0)$, where:

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

A threshold is included by having a weight $W_0(0)$ to an extra input v_0^{μ} which is 1 for all μ.

The output unit is trained to classify the patterns in $\mathcal{S}$ into two classes denoted by outputs of 0 and 1. Hidden units are trained to correct each of the two types of output error: 'wrongly off correctors' are trained to be 'on' for patterns for which the output is 'wrongly off', 'wrongly on correctors' are trained to be 'on' for patterns for which the output is 'wrongly on'. Two further steps aid convergence:
(i) if the re-training of the set of weights to the output after the addition of a hidden unit does not reduce the number of output errors, then the previous set of output weights is kept;
(ii) if a training set $\mathcal{S}$ for a hidden unit contains a small minority of patterns with target 1 then the algorithm may find connection weights such that the hidden unit is off for all patterns in $\mathcal{S}$. If this occurs then it is re-trained on a 'balanced' training set $\mathcal{S}'$ in which patterns with target 1 and 0 are present in equal numbers (formed by repeating patterns in $\mathcal{S}$ with target 1).

Classification into many classes is achieved by using one output for each class – this makes rejection easier in ambiguous cases: a positive classification is made only when exactly one output unit is 'on', any other output configuration is interpreted as a 'reject'.

2.1. *Convergence*

In the limit of a large number of passes through the training set, the Pocket version of the perceptron learning algorithm is guaranteed to find a set of connection weights that minimises the number of errors[13]. The results below apply in this limit, when the Pocket algorithm is used to train the connection weights to each unit, for details see Ref. 9.

We say that a unit is 'successful' if the number of errors it makes on its training set $\mathcal{S}$ is less than the number of patterns with target 1 in $\mathcal{S}$. The vertex points of the convex hull of a training set $\mathcal{S}$ are patterns that can each be separated from all other distinct patterns in $\mathcal{S}$ by an $N-1$ dimensional hyper-plane in input space. Using this and the interpretation of connection weights as a hyper-planar division of input space, we can prove the following:
(a) if, in the training set of a hidden unit, at least one of the patterns with target 1 is a vertex point, then connection weights will be found such that the hidden unit is 'successful';
(b) if a hidden unit is 'successful' then subsequent re-training of the output unit results in a reduction in the number of output errors;
(c) if a hidden unit is unsuccessful, then either (i) the subsequent re-training of the output unit results in a reduction in the number of output errors, or (ii) the next hidden unit to be added makes fewer errors than the current one.

We can see briefly that (a) follows from the fact that connection weights exist such that the hidden unit can be 'on' for one pattern with target 1 and 'off' for all other distinct patterns, and weights found by the Pocket algorithm will produce at most the same number of errors. (b) follows from similar reasoning: weights exist such that the connection of the successful hidden unit to the output corrects more errors than it creates. (c) follows from step (i) in section 2, and the fact that connection weights exist such that a second hidden unit, trained on the same training set, is either successful, or is 'wrongly on' for one pattern fewer.

Convergence of the algorithm to zero errors in a finite number of steps follows from (b) and (c), because the number of output errors decreases after the addition of every successful hidden unit (if a succession of unsuccessful hidden units is added then each makes fewer errors than the last, until one must make so few errors that it is successful).

3. Generalization and network size

Although the convergence proof tells us that we could allow the network to grow until it made no errors at all on the training set, this might correspond to 'over-fitting' and result in poor generalization. However, if the network is restricted to growing only a very small number of hidden units, it may not be capable of implementing the desired classification. A simple commonly used heuristic is to reserve a portion of the training set to be used solely to test generalization during training so that training can be stopped at roughly the right time. This means, of course, that less data can be used for training.

Alternatively, a bound on the probability of erring during generalization ϵ_g is provided by Vapnik[14], as a function of the number of training patterns p, the fraction of errors made on the training set ϵ_t and the *capacity* of the network h. The measure of capacity here is the maximum number of patterns h that can be found such that the network, by varying its connection weights and thresholds, can perform all possible 2^h classifications of the set of patterns (i.e. the largest number of patterns that the network can 'shatter'); h is called the Vapnik-Chervonenkis or 'VC' dimension. If $\epsilon_g << 1$ (the usual situation for a classifier) then[14], with probability $1-\eta$:

$$\epsilon_g \leq \epsilon_t + C_0^2\left(1+\sqrt{1+\frac{\epsilon_t}{C_0^2}}\right), \quad \text{where} \quad C_0^2(p,h,\eta) = \frac{h(\ln(2p/h)+1)-\ln\eta}{p}. \tag{2}$$

During training ϵ_t decreases, but the capacity h, and hence the second term in (2), increases. Experiments[15,16] indicate that a minimum occurs in ϵ_g, *before* ϵ_t reaches zero, and that ϵ_g follows the form of the upper bound implied by Eq. 2, i.e. both have a minimum at approximately the same value of h. The only difficulty in using this scheme to minimise ϵ_g lies in estimating the VC dimension for a multi-layer perceptron.

A result of Baum and Haussler[17] gives an upper bound for the VC dimension for the cascade architecture in terms of the number of units N and inputs n, namely: $h \leq 2N(N+2n+1)\ln(eN)$. However this is very much an over-estimate of the effective h as most weights are frozen after learning. Since the VC dimension of a perceptron with m independent inputs is $m+1$ (Ref. 18), a more reasonable (under) estimate comes from considering that, after a new hidden unit is added, the network can shatter $n+1$ more points independently of how many points the network could shatter before its addition i.e. $h \geq N(n+1)$ is a closer estimate of h. With a good approximation of h as a function of N we could restrict the size of the network N during training so as to minimise the right hand side of Eq. 2 and hence minimise ϵ_g, see Ref. 16.

4. Bounded decision regions

In most real applications a high fraction of correctly classified objects is not the only important thing; a classifier must also be able to refrain from making decisions whenever it does not have sufficient information. For example, it is often desirable in an OCR application to refrain from discriminating between D and O if sufficient contextual information is not available. However there is another (more subtle) kind of error that a recogniser can make, namely classifying an object which does not belong to any of the predefined categories that the recogniser should discriminate e.g. a recogniser of cats and dogs should be able to reject rabbits. Notice that this kind of 'outlier' is, by nature, untrainable: it is not possible to find representative examples to include in a training set.

4.1. *Putting a hyper-box around each class*

To avoid the miss-classification of outliers we must ensure bounded decision regions for each class $\mathcal{S}_j$ that enclose the set of training patterns for that class fairly tightly. To this end we used a simple neural algorithm that finds a hyperparallelopiped (or 'hyper-box') containing all of the training patterns of a class, oriented along the principal components of the class, and computes which hyper-box or boxes an input pattern lies in.

The 'self-organising perceptron'[19] is a simple, algorithm for estimating the directions of the principal components of a set of examples. We use this to find the normalised principle components P_n of each class in the training examples. Then for each class $\mathcal{S}_j$ we form a pair of perceptrons with weights: $\vec{W}^1 = -\vec{P}_n$, and $\vec{W}^2 = \vec{P}_n$ respectively, and set the respective thresholds to be: $W_0^1 = \max_{\vec{v}^\mu \in \mathcal{S}_j}(\vec{P}_n \cdot \vec{v}^\mu) - \epsilon$, and $W_0^2 = \min_{\vec{v}^\mu \in \mathcal{S}_j}(\vec{P}_n \cdot \vec{v}^\mu) + \epsilon$. The resulting pair of perceptrons have parallel hyper-planar decision surfaces (separated in the $\vec{P}_n$ direction) on either side of the training set such that both are 'on' only for input patterns falling between them. This process was repeated for each component. Thus, if a presented input pattern falls within the hyper-box for a given class, all of the perceptrons in the set of pairs of perceptrons for that class will be 'on'. The parameter ϵ can be greater than zero to shrink the hyper-box, excluding some of the training examples, so as to reduce the chance of incorrect classifications as opposed to rejects. The hyper-box algorithm ('HB') classifies an input pattern into a class only if it falls into exactly one hyper-box, otherwise it is rejected.

We combine the HB and PC algorithms so that the combined decision region is the intersection of the decision regions from each method. This allows the separating power of PC to be used whilst ensuring that the decision regions for each class are bounded. We refer to this combination as HPC. Thus a pattern is classified into a class by HPC if it is classified by both PC and HB into that class, otherwise it is rejected.

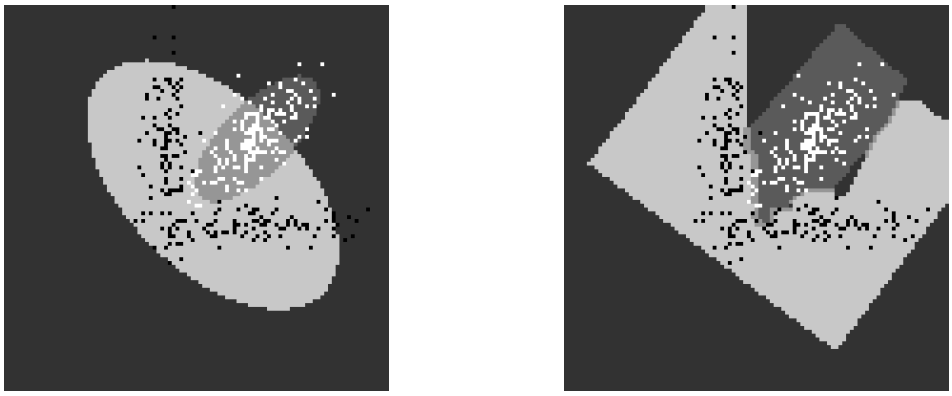

Fig. 2. Decision regions in the 2-dimensional equivalent of problem 2. Left: GML, right HPC. The black and white points (respectively) are the training examples from the two classes.

5. Comparison with a Gaussian classifier

We tested the above algorithms on some artificial data designed to highlight the problem of untrainable outliers. A Gaussian Maximum Likelihood classifier (GML) was used for comparison. The GML procedure assumes that the class density functions are Gaussians and uses the training data to estimate the parameters. In our simulations a pattern vector was classified as belonging to a class if exactly one estimated class density function is above a threshold (0.1) for that pattern, otherwise it is rejected. The Perceptron Cascade algorithm was allowed to grow hidden units until the fraction of errors during training was less than 0.1 and ϵ was set to 0 in HB. Ideally we should calculate the optimum network size as in section 3, and shrink the hyper-box for each class to exclude a fraction 0.1 of the training examples.

In all three artificial problems the classifiers were trained to classify 1000 4-vectors drawn from each of two different class density functions. The generalization of each classifier was tested with 1000 vectors from the two classes, plus 1000 'outliers' drawn from a Gaussian distribution with mean far from the two classes and much larger variance. In problem 1 the two class densities are simple overlapping Gaussians. In problem 2 one class density is concave, made by the superposition of two Gaussians elongated along the x and y axes, the other class density is a Gaussian wedged in the concavity (Fig. 2. shows an equivalent 2-D problem). In problem 3 one class is disjoint, made by the superposition of two well separated Gaussians, the other class density is a Gaussian centred between the two peaks of the first class density. For details see Ref. 20.

Table 1. Performance of four classifiers on the three problems in the text. GML: Gaussian Maximum Likelihood. PC: Perceptron Cascade. HB: Hyper-Box. HPC: Hyper-Box and Perceptron Cascade.

Algorithm	Problem 1.		Problem 2.		Problem 3.	
	% reject	% error	% reject	% error	% reject	% error
GML	8.2	0.6	14.9	0.6	41.4	0.6
PC	0.06	35.94	0.2	34.7	1.3	40.5
HB	58.8	1.1	22.9	0.1	58.2	1.1
HPC	0.7	2.4	3.0	0.3	2.1	6.4

Analysis of Table 1. and the confusion matrices (Ref. 20) shows that PC performs very badly with respect to the outliers – nearly all being wrongly classified rather than rejected. However, even when the class densities are concave or disjoint (problems 2 and 3 respectively), it does a good job of separating the two classes it was trained on. GML and HB have closed decision regions and do well in terms of restricting errors, but have to reject many patterns when the class densities are concave or disjoint. The combined algorithm HPC, having good

separating power and bounded decision regions, performs well in all situations. Fig. 2. shows the kind of decision regions formed by GML and HPC. We also notice that HPC improves on the performance of Upstart on the 3-D object recognition task (Ref. 10) in terms of the number of errors as opposed to rejects.

6. Conclusions

The robust convergence and economical construction of hidden units on classification of *real-valued* inputs by Perceptron Cascade[9] seems to have taken constructive algorithms to the point where they can be tested on real pattern classification problems. The approach of Vapnik[14] gives hope that we may be able to calculate how big to let the network grow so as to optimise generalization The unbounded decision regions of most feed-forward neural classifiers are impractical in many applications due to the problem of outliers or unanticipated events. We have implemented a neural algorithm[19] to bound decision regions with a hyper-box aligned with the principal components of each class. We showed that this solution works well and outperforms the traditional Gaussian classifier in the case of disjoint or concave class densities. In future work we will test this constructive algorithm on large scale benchmarks with the hope that it will remove the problems of speed, lack of convergence and having to guess a good architecture which trouble the standard error-backpropagation approach.

References

1. D. E. Rumelhart, G. E. Hinton and R. J. Williams, In Rumelhart, D. E. and McClelland, J. L. (Eds.) *Parallel Distributed Processing,* Vol. 1, MIT Press (1986).
2. G. M. Martin and J. A. Pittman, *Neural Computation*, **3**, 258-267 (1991).
3. W. Y. Huang and R. P. Lippman, *Proceedings of the 1st IEEE Int. Conf. on Neural Networks, Vol. 4*, 485-493 (1987).
4. J. A. Sirat and J.-P. Nadal, *Network,* **1**, 423-438 (1990).
5. S. E. Fahlman and C. Lebiere, in Touretzky, D. S. (Ed.) *Advances in Neural Information Processing Systems II*, Morgan Kaufmann, 524-532 (1990).
6. S. I. Gallant, *Proc. 8th Annual Conf. of Cognitive Science Society, Amherst MA,* 652-660 (1986).
7. M. Mezard and J.-P. Nadal, *J. Phys A* **22**, 2191-2203 (1989).
8. S. I. Gallant, *IEEE Trans. on Neural Networks*, **1** No. 2, 179-191 (1990).
9. N. Burgess, *in preparation.*
10. N. Burgess, M. Notturno Granieri and S. Patarnello, *International Journal of Neural Systems* **2** No. 4, 275-282 (1992).
11. M. R. Frean, *Neural Computation* **2**, No. 2, 198-209 (1990).
12. K. J. Lang and M. J. Witbrock, in Touretzky, D. S., Hinton, G. E. and Sejnowsky, T. (Eds.) *Proceedings of the 1988 Connectionist Models Summer School,* Morgan Kaufmann (1988).
13. S. I. Gallant, *Proc.* 8^{th} *Int. Conf. on Pattern Recognition, Paris, October*, Washington DC: IEEE Computer Soc. Press, 849-52 (1986).
14. V. Vapnik, *Proc. NIPS 4,* Morgan Kaufmann (1992).
15. I. Guyon, V. Vapnik, B. Boser, L. Bottou and S. Solla, *Proc. NIPS 4* Morgan Kaufmann (1992).
16. S. Solla, *this volume.*
17. E. B. Baum and D. Haussler, *Neural Computation*, **1**, No. 1, 151-160 (1989).
18. R. S. Wenocur and R. M. Dudley, *Discrete Math., V* **33**, 313-318 (1981).
19. J. A. Sirat, *International Journal of Neural Systems,* **2**, 147-155 (1991).
20. S. Di Zenzo, N. Burgess, P. Ferragina, and M. Notturno Granieri, *in preparation.*