Representation and Generalization Properties of Class-Entropy Networks

Sandro Ridella, *Member, IEEE*, Stefano Rovetta, *Member, IEEE*, and Rodolfo Zunino, *Member, IEEE*

***Abstract*—Using conditional class entropy (CCE) as a cost function allows feedforward networks to fully exploit classification-relevant information. CCE-based networks arrange the data space into partitions, which are assigned unambiguous symbols and are labeled by class information. By this labeling mechanism the network can model the empirical data distribution at the local level. Region labeling evolves with the network-training process, which follows a plastic algorithm. The paper proves several theoretical properties about the performance of CCE-based networks, and considers both convergence during training and generalization ability at run-time. In addition, analytical criteria and practical procedures are proposed to enhance the generalization performance of the trained networks. Experiments on artificial and real-world domains confirm the accuracy of this class of networks and witness the validity of the described methods.**

***Index Terms*— Class-entropy networks, clustering methods, generalization, minimum entropy methods, noise-injection, pruning.**

I. Introduction

CONDITIONAL class entropy (CCE) can be used in classification problems as a cost function to drive supervised training. A CCE-based network has a standard multilayer feedforward architecture with hard-limiter units [1]: at each layer, the configuration of neuron outputs supports a library of symbols, and the training process must minimize the ambiguity associated with each symbol. In that case, quantities from information theory replace the traditional mean square error cost function.

A CCE-based network implements a digital encoding of input data: the first layer partitions the data space into regions, to which each neuron contributes with a separating boundary. Classification is accomplished by labeling each region with one class. Consistent training implies that all symbol-class associations are unequivocal, i.e., all samples encoded by a symbol belong to the same class. Previous research showed that conventional Hebbian learning preserves the global entropy of a data set [2]; CCE-based networks aim to best exploit the information related to classification [1]. The lack of a gradient-based learning rule leading to a practical algorithm is the main drawback of using class entropy as a cost function. Anyway, simulated annealing [3], [4] or random search [5] provide effective optimization methods for tuning weights. Computational-cost issues typically impose the use of plastic approaches for training "growing" networks, whose neurons are tuned sequentially and incrementally.

This paper presents methodologies both to set up CCE-based classifiers and to use trained networks for domain analysis and interpretation. After describing an augmentation of the basic neuron model to support circular boundaries [6], the paper discusses convergence properties for the general class of plastic CCE networks. Then proof is given of several theoretical properties, which relate the CCE cost function to the eventual classification performance. Finally, the generalization ability of trained networks is addressed. Theory derives an analytical criterion to control network complexity by injecting noise into data, whereas two practical algorithms are given to simplify and interpret training results (pruning and clustering, respectively).

In the presented approach, joining infomation theory with neural models endows the resulting networks with good classification accuracy. At the same time, the model exhibits interesting features for domain analysis that might be difficult to implement with other classical approaches. For example, by its partitioning ability a CCE-based network can give hints about the number of significant regions in the data space. Likewise, the network can be used to segment regions of the data space according to the local classifier's confidence.

The experimental verification of the overall framework mainly aimed both to validate theoretical expectations and to characterize generalization performance on a substantive statistical basis. Therefore, experiments covered a few synthetic and several real-world standard testbeds available in the literature. Synthetic domains include nested-spirals [7], the generalized XOR [8], and the modeling of a randomly chosen classification problem. Real-domain testbeds include the "sonar" [9], "BCancer" [10], "Phoneme" [11], "Vowel" [12], and "Vowel 2D" [13] databases.

Section II presents the framework for CCE-based networks, including the neuron augmentation to circular unit and the proof of the theoretical properties of CCE. Section III addresses the generalization issue and describes several methods that can be included within the CCE framework to control a network's complexity. Section IV presents the set of experimental results from the various experiments. Some concluding remarks are made in Section V.

II. Theory of Conditional Class-Entropy Networks

A. Basic Theory for CCE-Based Networks

This brief overview of CCE-related theory follows the same outline as presented in [1]. A CCE-based network has

Manuscript received May 22, 1996; revised May 28, 1998. This work was supported in part by the Italian Ministry for University and Research (MURST).

The authors are with the Department of Biophysical and Electronic Engineering, University of Genoa, 16145 Genova, Italy.

Publisher Item Identifier S 1045-9227(99)00921-2.

a feedforward structure with hard-limiter units; the (binary) output activation of the jth neuron at level k, $b^{(j,k)}$, is given by the Heaviside function, θ: $\Re \rightarrow \{0,1\}$, of a weighted combination (stimulus) of input variables, $\mathbf{x} = \{x_i; i = 1, \cdots, d(k)\}$ and a bias

$$b^{(j,k)} = \theta\left(\sum_{i=1}^{d(k)} w_i^{(j,k)} x_i + w_0^{(j,k)}\right) \tag{1}$$

where the terms $w_i^{(j,k)}$ make up a neuron's set of adjustable coefficients. At the first layer, $d(0)$ coincides with the dimensionality of the input space, where input variables may be either digital or analog. The collection of neuron activations at each layer comprises a set of binary variables, whose configurations form a vocabulary, $\mathbf{V}^{(k)}$, of symbols

$$\begin{aligned} \mathbf{s}_n^{(k)} &= \left(b^{(1,k)}, b^{(2,k)}, \cdots, b^{(d(k),k)}\right); \\ \mathbf{V}^{(k)} &= \{\mathbf{s}_n^{(k)}; 1 \le n \le 2^{d(k)}\}; \end{aligned} \quad k \ge 1. \tag{2}$$

Thus, at each level, the network's transfer function can be regarded as a mapping, $M^{(k)}(\mathbf{x})$, of input variables into symbols. For levels above the first one, the mapping process can be expressed as elementary combinatorial functions of binary codes; feedforward network theory [14] shows that one additional layer can support any symbol-to-class function, and such approach is followed in [1]; however, this mapping functionality can be implemented by look-up table, or, more conveniently, by decision trees [15]–[17] as well.

In fact, the relevant coding process takes place in the first layer, where the input space is partitioned into regions and corresponding symbols are assigned. Since the goal of this research is to exploit CCE properties for domain analysis, in the following, only the first-layer mapping will be considered without loss of generality. Consequently, the level-indexing superscript will be omitted for simplicity. Fig. 1 gives a schematic representation of the network structure, and a sample of the data-space partitioning mechanism.

Classification is performed by considering, for each symbol, the classes of covered samples, and by measuring the ambiguity associated with the occurrence of the symbol itself. That quantity measures the uncertainty in classification when an input sample has been mapped into a given symbol. Thus, if C_c indicates the cth class among the possible C classes, one has to evaluate the conditional probability that a pattern, $\mathbf{x}$, encoded by symbol $\mathbf{S}_n$, belongs to the class C_c

$$P_c(\mathbf{S}_n) = P(\mathbf{x} \in C_c \mid M(\mathbf{x}) = \mathbf{S}_n). \tag{3}$$

Entropy [18] measures the uncertainty associated with the nth symbol

$$H(\mathbf{S}_n) = -\sum_{c=1}^{C} P_c(\mathbf{S}_n) \log P_c(\mathbf{S}_n). \tag{4}$$

The analysis presented in [19] shows that the number of symbols, N, depends on the space dimensionality, $d(0)$, and on the number, W, of units in the first layer, and is upper

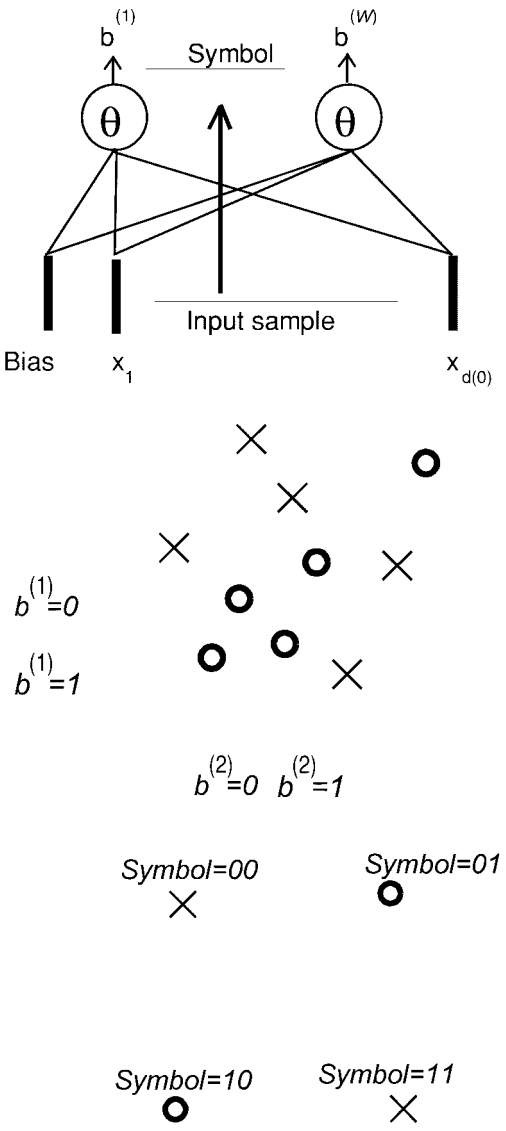


Fig. 1. The sample encoding mechanism in the first layer.

bounded as follows:

$$N \le \sum_{k=0}^{d(0)} \binom{W}{k} \tag{5}$$

where $\binom{W}{k} = 0$ if $k > W$. When $d(0) > W$, the bound (5) confirms an intuitive expectation: $N \le 2^W$.

The uncertainty associated with the overall encoding system is obtained by averaging entropy (4) over all symbols; symbols are weighted with their probabilities of occurrence, $P(\mathbf{S}_n)$. The weighted sum covers all occurred symbols and yields CCE

$$\mathrm{CCE} = \sum_{n=1}^{N} P(\mathbf{S}_n) \cdot H(\mathbf{S}_n). \tag{6}$$

The exact evaluation of the quantities related to CCE depends on the actual data distribution, which is often unknown *a*

priori, hence such quantities are estimated empirically from the training set [1]. The following notations are now introduced:

$X_{T\,\mathrm{TOT}}$ Total number of samples in the training set.
$X(\mathbf{S}_n)$ Counts the occurrences of symbol $\mathbf{S}_n$, and equals the number of samples encoded by $\mathbf{S}_n$.
$X^{(c)}(\mathbf{S}_n)$ Counts the number of samples which are encoded by $\mathbf{S}_n$ *and* belong to the cth class.

With these notations one can write empirical assessments of the required probabilities

$$\hat{P}(\mathbf{S}_n) = \frac{X(\mathbf{S}_n)}{X_{\mathrm{TOT}}}; \quad \hat{P}_c(\mathbf{S}_n) = \frac{X^{(c)}(\mathbf{S}_n)}{X(\mathbf{S}_n)}. \tag{7}$$

Substituting such estimates for the actual values in (3) and (6) yields the total cost function associated with the encoding system M

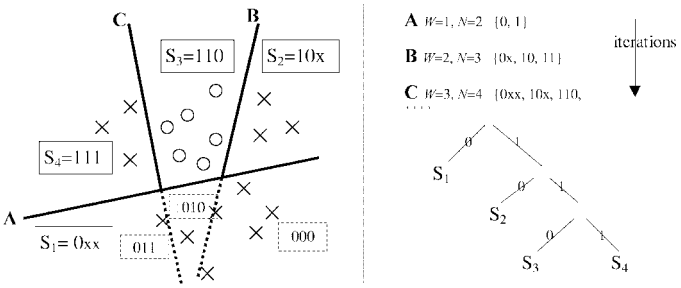
$$\mathrm{CCE}_M = -\sum_n \frac{X(\mathbf{S}_n)}{X_{\mathrm{TOT}}} \cdot \sum_{c=1}^{C} \frac{X^{(c)}(\mathbf{S}_n)}{X(\mathbf{S}_n)} \log \frac{X^{(c)}(\mathbf{S}_n)}{X(\mathbf{S}_n).} \tag{8}$$

B. Training Strategies

The goal of the training process is to find out an optimal weight set for the mapping system, M, minimizing expression (8). The global minimum is zero and indicates a perfect classifier with no ambiguities over training data. Imposing CCE (8) as a cost function inhibits the application of gradient-based optimization strategies. As a consequence, typical approaches to weight tuning involve either simulated annealing [3], [4] or random-search [5] methodologies.

The optimization problem is further complicated by the fact that the proper number, W, of neurons is not known. Therefore, constructive strategies with increasing W are usually adopted to train CCE networks, according to two different approaches. *Batch* training progressively adds units and readjusts the whole parameter set after each neuron insertion. This procedure ensures that the weight set is fully exploited in the optimization task, but optimization complexity increases with a network's size and might require massive computational efforts. *Incremental* training adds one unit after each optimization run, but freezes the weights of existing (previously added) neurons. This keeps the complexity of the optimization process constant, but represents a sort of "hill climbing" in the weight space and most likely fails to bring in the smallest number of neurons. The incremental nature of network building makes the overall CCE-training problem similar to other constructive models of neural networks [20], [21]. Incremental training will be assumed as a default throughout the paper; the following is an outline of the basic training process [1], [22]:

0) *Input*: a data set; empty hidden layer $(W = 0)$;
the maximum tolerated CCE value, ε;
1) *Repeat*
  Insert and *random initialize* a new unit;
  $W = W + 1$;
  Repeat
    Compute the quantities (7) and the cost function (8) with the current weight set;
    Adjust the weights of the newly added unit accordingly;
  Until the optimization algorithm stops;
Until $(\mathrm{CCE}_M < \varepsilon)$.



(a) (b)

Fig. 2. Freezing symbols reduces computational cost. (a) Data space partitioning. (b) Symbol heirarchy.

The plastic network-construction algorithm makes it possible to further reduce the computational cost of training as follows. A "homogeneous" symbol, covering samples belonging to the same class, yields null local entropy, hence its contribution to the total cost (8) is null. Therefore, as soon as a set of weights are "freezed" after optimizing a new neuron, one can sort out homogeneous symbols and remove all their covered samples from the training set. This mechanism has two major effects: first, pruning the sample set reduces the computational cost for adjusting future neurons; second, the symbol set keeps limited, since all symbols derived from a homogeneous one need not be computed. The latter property permits a hierarchical structure of occurred symbols, which makes it possible that several, "virtual" symbols are not accounted for explicitly.

A sample of symbol construction and the involved hierarchy is illustrated in Fig. 2. Neurons A, B, C are included sequentially: after the optimization of the first neuron ("A"), symbol S_1 ("0") is detected as homogeneous and does not proliferate. Virtual subsymbols "000," "010," and "011" will not appear in the table. Likewise, the homogeneous symbol S_2 does not expand after the optimization of neuron "B."

The freezing mechanism generating virtual symbols makes for a notable saving of computational cost and memory space in symbol handling; in the above example, the final number of stored symbols amounts to $N = 4$ instead of the expected $N = 7$. More importantly, the possibility of delimiting space regions by half-hyperplanes allows the network to define complex structures difficult to represent by standard perceptrons. This property is specific of the CCE network construction process and will be extensively exploited when dealing with generalization performance.

C. Circular Augmentation of Neurons

As shown in expression (1), each neuron in the network is a classical perceptron. A linear boundary is adequate in simple cases, but approximating more complicate class distributions may require combinations of many hyperplanes, thus failing to retain some features of the classification problem. In other words, the representation of the separating surfaces is done by an unnecessarily large number of components. This is an obstacle to an understandable domain representation. In such cases, it is often useful to approximate separating surfaces with boundaries with only a local support (closed surfaces).

Moreover, classification requires a rejection ability (or confidence assessment). Rejection requires the classifier to im-

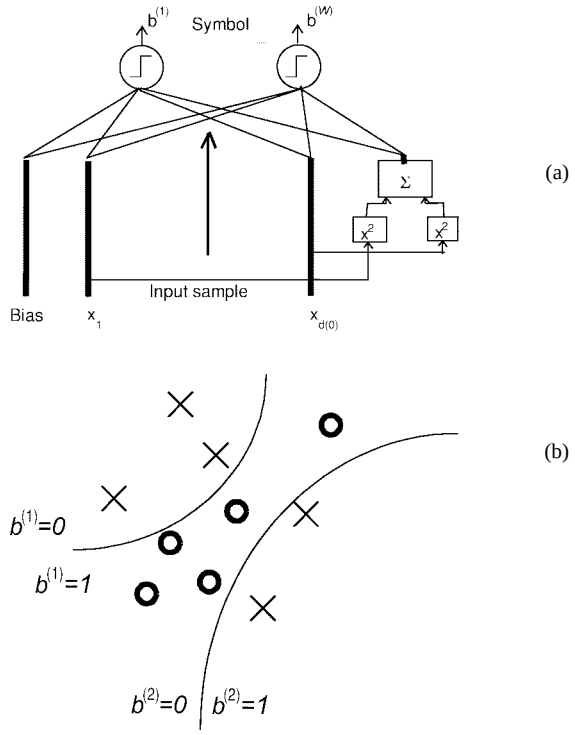


Fig. 3. (a) The circular neuron model. (b) A sample of partitioning.

plement a representation of data distribution, with closed boundaries for the high-confidence regions. It has been proved [23] that with a limited number of hidden units, desirable for many reasons, it is difficult to guarantee closed boundaries with a standard multilayer perceptron, and it is even impossible when $d(0) < d(1) + 1$.

The authors have also proved [6] that in many senses the optimal choice for implementing a closed-boundary unit is the "circular" model. Specifically, it is based on the only second-order discriminating function which guarantees closed boundaries, and features the maximum efficiency [defined as gain in representation ability versus increase in Vapnik–Cervonenkis (VC)-dimension or number of parameters] with respect to the standard perceptron.

In the "circular" neuron model, a neuron's stimulus includes a term that sums the squares of input values (Fig. 3). The augmented formulation of a neuron's output function can be rewritten as

$$b^{(j,k)} = \Theta\left(\sum_{i=1}^{d(k)} w_i^{(j,k)} x_i + w_0^{(j,k)} + w_q^{(j,k)} \sum_{i=1}^{d(k)} x_i^2\right). \quad (9)$$

It is easy to show that the argument in expression (9) draws a circular boundary; since only one additional input is involved, the increase in complexity is limited. It is known that the increase in the number of network weights has limited effects on the generalization requirements in terms of VC dimension [6], since the VC-dim of the circular unit is equal to the number of parameters [24]. Therefore, the circular neuron can be regarded as a standard perceptron, in which the quadratic input behaves as a normal additional dimension in the data space.

Incidentally, this neuron model is a superset of the basic linear perceptron, to which it reduces when $w_c = 0$. However, the ultimate choice between a linear or a circular representation relies on the optimization process and not on some *a priori* model setting. Letting the network choose the appropriate representation paradigm is the major advantage of the described augmentation [22].

D. Convergence of CCE Networks

This section analyzes some important properties of CCE as a cost function and of CCE-based classifiers. Basic properties of the entropy function and quantities derived from information theory will be used extensively.

Definition: The entropy associated with a probability distribution, $p^{(c)}$, is

$$H_C(p^{(c)}) = -\sum_{c=1}^{C} p^{(c)} \log_2 p^{(c)}$$

where the summation of C terms covers all values of $p^{(c)}$. When $C = 2$, the entropy of two terms, p and q, will be denoted as

$$H_2(p, q) = -(p \log_2 p + q \log_2 q).$$

Property: Information theory states the concavity of the entropy function [24, Th. 2.7.3, p. 30], from which follows, by Jensen's inequality [24]

$$\sum_i \lambda_i H_C(p_i^{(c)}) \le H_C\left(\sum_i \lambda_i p_i^{(c)}\right) \quad 0 \le \lambda_i \le 1, \quad \sum_i \lambda_i = 1. \quad (10)$$

In the interval $p \in [0, 1/2]$, the following property will be used in the analysis:

$$H_2(p, 1-p) \ge 2p. \quad (11)$$

Definition (Partition of a Data Set): Given a set of samples, $\mathbf{X}$, a *partition* $\{\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_N\}$ of $\mathbf{X}$ is a collection of subsets $\mathbf{S}_1, \cdots, \mathbf{S}_N$ such that: $\mathbf{X} = \mathbf{S}_1 \cup \mathbf{S}_2 \cup \cdots \cup \mathbf{S}_N; \mathbf{S}_i \cap \mathbf{S}_j = \varnothing \forall i \neq j$. For simplicity, we use a common notation for symbols and for subsets of samples. This notations is unambiguous since the network supports a one-to-one mapping between symbols and data partitions; in the following, *cluster* will be also used as a synonym for subset of data. The fraction of samples in $\mathbf{S}_n$ (with respect to the whole training set) will be denoted by s_n, and the fraction of samples in $\mathbf{S}_n$ (with respect to the total number of samples in $\mathbf{S}_n$) belonging to the cth class will be denoted by $s_n^{(c)}$, hence $\sum_{n=1}^{N} s_n = \sum_{c=1}^{C} s_n^{(c)} = 1$. The CCE (8) associated with a partition $\{\mathbf{S}_1, \cdots, \mathbf{S}_N\}_{\mathbf{X}}$ is worked out as

$$\begin{aligned} \mathrm{CCE}(\{\mathbf{S}_1, \cdots, \mathbf{S}_N\}_{\mathbf{X}}) &= \sum_{n=1}^{N} s_n \cdot \mathrm{CCE}(\mathbf{S}_n) \\ &= \sum_{n=1}^{N} s_n \cdot H_C(s_n^{(c)}). \end{aligned}$$

Definition (Nontrivial Partitioning Operator): Given a data set, $\mathbf{X}$, including at least two samples, a nontrivial partitioning operator is any mapping Ψ such that $\Psi(\mathbf{X}) = \{\mathbf{S}_1, \cdots, \mathbf{S}_N\}$, with $N \ge 2$ and $S_i \neq \varnothing \forall i$.

Definition (Homogeneous Partition): A partition is homogenous if all samples in the partition belong to the same class.

Property 1 (Partitioning Does Not Increase CCE): Let $\mathbf{S} = \{\mathbf{S}_1, \cdots, \mathbf{S}_N\}_{\mathbf{X}}$ be a partition of a data set $\mathbf{X}$, and $\mathbf{S}' = \{\mathbf{S}_{1,1}, \cdots, \mathbf{S}_{1,K_1}, \cdots, \mathbf{S}_{N,1}, \cdots, \mathbf{S}_{N,K_N}\}$ a partition derived from $\mathbf{S}$: $\mathbf{S}_n = \bigcup_{k=1}^{K_n} \mathbf{S}_{n,k}; n = 1, \cdots, N$. CCE is a monotonically nonincreasing function: $\mathrm{CCE}(\mathbf{S}') \leq \mathrm{CCE}(\mathbf{S})$.

Proof: This property derives directly from Jensen's inequality (10)

$$\begin{aligned}
\mathrm{CCE}(\mathbf{S}') &= \sum_{n=1}^{N} \sum_{k_n=1}^{K_n} s_{n,k_n} \cdot \mathrm{CCE}\left(\mathbf{S}_{n,k_n}\right) \\
&= \sum_{n=1}^{N} s_{n,k_n} H_C\left(s_{n,k_n}^{(c)}\right) \\
&= \sum_{n=1}^{N} s_n \sum_{k_n=1}^{K_n} \frac{s_{n,k_n}}{s_n} H_C\left(s_{n,k_n}^{(c)}\right) \\
&\leq \sum_{n=1}^{N} s_n H_C\left(\sum_{k_n=1}^{K_n} \frac{s_{n,k_n}}{s_n} s_{n,k_n}^{(c)}\right) \\
&= \sum_{n=1}^{N} s_n H_C\left(\sum_{k_n=1}^{K_n} \frac{X\left(\mathbf{S}_{n,k_n}\right)}{X(\mathbf{S}_n)} \frac{X^{(c)}\left(\mathbf{S}_{n,k_n}\right)}{X\left(\mathbf{S}_{n,k_n}\right)}\right) \\
&= \sum_{n=1}^{N} s_n H_C\left(s_n^{(c)}\right) \\
&= \mathrm{CCE}(\mathbf{S}). \qquad \square
\end{aligned}$$

The fact that splitting a data partition reduces CCE can be exploited to prove convergence of CCE iterative training. The following theorem encompasses the general class of CCE network models.

Theorem (Convergence of Plastic CCE Network): Let Γ be a nontrivial partitioning operator. Then, for any data set, $\mathbf{X}$, there exists a finite sequence of instantiations of $\Gamma, \{\Gamma^{(1)}, \cdots, \Gamma^{(W)}\}$, that splits $\mathbf{X}$ into N clusters such that $\mathrm{CCE}(\{\mathbf{S}_1, \cdots, \mathbf{S}_N\}_{\mathbf{X}}) = 0$. The sequence is obtained by the following procedure:

CCE Network Training Algorithm

Initialize partition $V^{(0)}$: set $V^{(0)} = \{\mathbf{X}\}$; set $w = 0$;

Repeat

1) *Pick* a cluster, $\mathbf{S}_n$, (if existing) from $V^{(w)}$ such that $CCE(\mathbf{S}_n) > 0$;
2) *If* $\mathbf{S}_n$ exists:
   b.1 *Apply* Γ to $\mathbf{S}_n$, and obtain a derived partition $V'^{(w)}$;
   b.2 *Replace* $V^{(w)}$ with $V'^{(w)}$
3) *Set* $w = w + 1$;

Until $CCE(V^{(w)}) = 0$.

Proof: The theorem does not make any specific assumption about the partitioning operator Γ, except for its nontrivial splitting, and exploits the validity of Property 1 at each algorithm iteration. The general statement addresses a whole class of plastic algorithms, for which a proof of convergence exploiting the nonincreasing property of the cost function can be found in [25]. $\square$

In the algorithm, each instance of Γ further splits but does not recombine the partitions set by previous instances; this implements incremental network construction. The theoretical framework connects to neural network modeling: an instantiation of the operator Γ has a one-to-one correspondence with the inclusion of a neuron, hence Theorem 1 ensures convergence for CCE-based neural networks.

Proving that an operator is nontrivial ensures that the operator can be successfully used for CCE network training. This encompasses a wide class of operators, some of which will be accounted for in the following; it is worth stressing that the literature presents a wide collection of convergence theorems for specific operators, which are typically sharper than the one here mentioned. Comparisons between different constructive algorithms can be found in [26].

Operator 1 (Linear Perceptron): The hyperplane-based partitioning schema, implemented by the linear perceptron, trained with the "pocket" training algorithm [27], is clearly nontrivial and satisfies the convergence theorem. Incidentally, this is the schema adopted in [1] for building CCE networks; a convergence theorem for this operator is given in [28].

Operator 2 (Circular Perceptron): The previous operator $\Gamma^{(\mathrm{LP})}$, when applied to the augmented neuron model defined in Section II.3, supports a nontrivial partitioning scheme, $\Gamma^{(\mathrm{CP})}$, drawing a hypersphere in the data space [22].

Operator 3a (Hull Clipping): Let $\Gamma^{(\mathrm{HC})}$ be an operator separating a point on the boundary of $\mathbf{X}$ from the rest of the data; for example, $\Gamma^{(\mathrm{HC})}$ might simply isolate an element of the convex hull of $\mathbf{X}$. By construction, $\Gamma^{(\mathrm{HC})}$ is nontrivial and fulfils the convergence conditions. Computing the convex hull can be very expensive in higher dimensions, hence this approach has a mere theoretical value, since there are much more efficient ways to locate boundary points. The application of $\Gamma^{(\mathrm{HC})}$ is ultimately equivalent to Huang's analysis [29] of the maximum number of hard-limiter units in feedforward networks.

Operator 3b (Cluster Aggregation):

$\Gamma^{(\mathrm{CA})}$:=*Apply* either $\Gamma^{(\mathrm{LP})}$ or $\Gamma^{(\mathrm{CP})}$ in such a way that one generated cluster, $\mathbf{S}^\star$ is homogeneous and is as large as possible;

Return the partition $\{\mathbf{S}', \mathbf{S}''\}_{\mathbf{X}}$ composed by clusters: $\mathbf{S}' = \mathbf{S}^\star; \mathbf{S}'' = \mathbf{X} - \mathbf{S}^\star$.

This algorithm improves the convex-hull clipper, as it allows a cluster to include several samples, whereas a decrease in CCE is still ensured by the homogeneous nature of created clusters. In fact, maximizing a homogeneous cluster's size may require careful optimization. $\Gamma^{(\mathrm{CA})}$ is computationally more reasonable than $\Gamma^{(\mathrm{HC})}$ and yields a more efficient representation, as it generates a smaller number of partitions. Convergence theorems for operators 3a–3b have been published in [30] and [31].

E. Theoretical Properties of CCE

The use of CCE as a cost function for training raises some issues about its relation with the sample distribution and with the classifier network's performance. This section analyzes

some theoretical properties of CCE that can help a designer in setting up a neural classifier.

Property 2 (CCE Training as a Monte Carlo Minimization): The network training strategy based on the minimization of expression (8) implements a data-driven, Montecarlo version of the minimization of the cost function, G

$$G = -\sum_{n=1}^{N}\sum_{c=1}^{C}\sum_{v=1}^{C}\int [\delta(\mathbf{x}, n, c, v)\log_2 p(c \mid n)]p(v, \mathbf{x})\, d\mathbf{x} \tag{12}$$

where

- $p(c \mid n)$ is the probability that a sample in cluster $\mathbf{S}_n$ belongs to class C_c, and is estimated by $s_n^{(c)}$;
- $\delta(\mathbf{x}, n, c, v)$ is the indicator function defined as: $\delta(\mathbf{x}, n, c, v) = \begin{cases} 1 & \text{if } \mathbf{x} \in \mathbf{S}_n \quad \text{and} \quad v = c \\ 0 & \text{otherwise} \end{cases}$;
- $p(v, \mathbf{x})$ is the (joint) probability that a sample lies at position $\mathbf{x}$ and belongs to class C_v.

This property is direct consequence of the fact that a sample-driven Montecarlo training of a network based on function (12) is obtained by evaluating the cost, G', on a discrete sample, $\mathbf{X}$

$$\begin{aligned} G' &= -\sum_{n=1}^{N}\sum_{c=1}^{C}\sum_{v=1}^{C}\frac{1}{X_{\mathrm{TOT}}}\sum_{l=1}^{X_{\mathrm{TOT}}}\delta(\mathbf{x}_l, n, c, v(\mathbf{x}_l))\log_2 p(c \mid n) \\ &= -\sum_{n=1}^{N}\sum_{c=1}^{C}\frac{1}{X_{\mathrm{TOT}}}X^{(c)}(\mathbf{S}_n)\log_2 p(c \mid n) \\ &= \sum_{n=1}^{N}\left[\frac{X(\mathbf{S}_n)}{X_{\mathrm{TOT}}}\sum_{c=1}^{C}\frac{X^{(c)}(\mathbf{S}_n)}{X(\mathbf{S}_n)}\log_2 p(c \mid n)\right] \\ &= -\sum_{n=1}^{N}P(\mathbf{S}_n)\sum_{c=1}^{C}p(c \mid n)\log_2 p(c \mid n) \\ &= \mathrm{CCE}. \end{aligned}$$

Thus CCE-based training is equivalent to the empirical minimization of a cost function supported by the true sample distribution, $p(v, \mathbf{x})$; that important property guarantees that increasing the number of samples improves convergence on the true statistics of the observed domain.

Conversely, one might wonder how minimizing CCE relates to a system's eventual classification error, both on the training data and, more importantly, on the overall data distribution. The following properties express such relations in terms of bounds on the CCE value.

Property 3 (Bounds on CCE—Training): Let ε denote the total classification error on training data (i.e., the ratio of misclassified samples to the total number of samples). The following bounds can be set to the CCE value:

$$2\varepsilon \le \mathrm{CCE} \le -\varepsilon\log_2\varepsilon - (1-\varepsilon)\log_2(1-\varepsilon) + \varepsilon\log_2(C-1). \tag{13}$$

Proof: Consider the nth cluster, $\mathbf{S}_n$, and let ε_n denote the local classification error within $\mathbf{S}_n$. Computing ε_n involves a calibration process, which is performed by identifying the locally predominant class, $\hat{c}_n$, and labeling the cluster accordingly. If the cluster has several equally predominant classes, one of them is selected at random. With these notations

$$\varepsilon_n = 1 - s_n^{(\hat{c})} = \sum_{c \ne \hat{c}_n} s_n^{(c)}. \tag{14}$$

To prove the lower bound, the "winning" class in $\mathbf{S}_n$ is considered separately; applying a known property [24] to local entropy yields

$$\begin{aligned} H(\mathbf{S}_n) = H_C\left(s_n^{(c)}\right) &= H_2\left(s_n^{(\hat{c})}, \sum_{c \ne \hat{c}} s_n^{(c)}\right) \\ &\quad + \left(\sum_{c \ne \hat{c}_n} s_n^{(c)}\right) H_{C-1}\left(\left.\frac{s_n^{(c)}}{\sum_{w \ne \hat{c}_n} s_n^{(w)}}\right|_{c \ne \hat{c}_n}\right). \end{aligned} \tag{15}$$

By using the nonnegative nature of entropy, expression (14), and property (11), one obtains

$$H(\mathbf{S}_n) \ge H_2\left(s_n^{(\hat{c})}, \sum_{c \ne \hat{c}_n} s_n^{(c)}\right) = H_2(1-\varepsilon_n, \varepsilon_n) \ge 2\varepsilon_n. \tag{16}$$

Summing contributions from all clusters and applying (16) gives

$$\begin{aligned} \mathrm{CCE} &= \sum_{n=1}^{N} s_n H(\mathbf{S}_n) \ge 2\sum_{n=1}^{N} s_n \varepsilon_n \\ &= 2\sum_{n=1}^{N}\frac{X_n}{X_{\mathrm{TOT}}}\sum_{c \ne \hat{c}_n}\frac{X_n^{(c)}}{X_n} = 2\varepsilon. \end{aligned} \tag{17}$$

which completes the first part of the proof.

As to the upper bound, one first derives an upper bound for (15) by considering that entropy is maximum for a uniform distribution of values

$$\begin{aligned} H(\mathbf{S}_n) &= H_2(1-\varepsilon_n, \varepsilon_n) + \varepsilon_n H_{C-1}\left(\left.\frac{s_n^{(c)}}{\sum_{w \ne \hat{c}_n} s_n^{(w)}}\right|_{c \ne \hat{c}_n}\right) \\ &\le H_2(1-\varepsilon_n, \varepsilon_n) + \varepsilon_n \log_2(C-1). \end{aligned} \tag{18}$$

Applying (18) and Jensen's inequality (10) to the expression for CCE gives the upper bound

$$\begin{aligned} \mathrm{CCE} &= \sum_{n=1}^{N} s_n H(\mathbf{S}_n) \\ &\le \sum_{n=1}^{N} s_n [H_2(1-\varepsilon_n, \varepsilon_n) + \varepsilon_n \log_2(C-1)] \\ &= \sum_{n=1}^{N} s_n H_2(1-\varepsilon_n, \varepsilon_n) + \varepsilon \log_2(C-1) \\ &\le H_2\left(\sum_{n=1}^{N} s_n(1-\varepsilon_n), \sum_{n=1}^{N} s_n \varepsilon_n\right) + \varepsilon \log_2(C-1) \\ &= H_2(1-\varepsilon, \varepsilon) + \varepsilon \log_2(C-1) \end{aligned}$$

which completes the proof. □

The above properties may be useful during network training, especially because the lower bound witnesses the fact that CCE is an effective estimator of the classification error. Conversely, an upper bound may be useful to some training strategies that do not pursue perfect training, but rather must meet some requirements on classification error.

When dealing with real applications, however, one is most interested in the classifier's performance on the actual data distribution, which is not usually known *a priori*. This raises the problem of the generalization ability of CCE networks. In practice, generalization performance is estimated by observing the classifier's performance on a data set not used to train the network; cross-validation is widely used in the literature for generalization assessment [14]. The following analysis defines the use of cross-validation in CCE networks, and relates the CCE value at cross-validation to the corresponding classification error. CCE at cross-validation, $\mathrm{CCE_{CV}}$, is defined as

$$\mathrm{CCE_{CV}} = \sum_{j=1}^{J} \sigma_j \sum_{c=1}^{C} \sigma_j^{(c)} \log_2 s_j^{(c)} \tag{19}$$

where J counts the partitions that are covered by test data; in analogy with the definitions given for training, σ_j, $\sigma_j^{(c)}$ indicate, for the jth cluster, the total share of test samples and the relative share of test samples belonging to class C_c, respectively.

It is important to stress that the data-space partitioning schema is fixed, and has been wired into the network weights by the training process. This explains why expression (19) uses probabilities $s_j^{(c)}$ in the argument of the logarithm. In order to evaluate $\mathrm{CCE_{CV}}$, one must: 1) supply the trained network with test data to observe the covered partitions; 2) evaluate probabilities $\sigma_j, \sigma_j^{(c)}$, accordingly; and 3) use results in (19). Quantity (19) has no longer the properties of an entropy; nevertheless, it features interesting properties when regarded as a complexity measure of data distribution.

Property 4 (Description-Length Interpretation of $\mathrm{CCE_{CV}}$): $\mathrm{CCE_{CV}}$ measures the average extra bits that are required to represent the test data distribution when using the training data distribution as an estimator.

Proof: Consider the single jth cluster covered by test data; clearly, the smallest amount of bits to represent the related classification problem within such cluster is given by

$$B_j^* = -\sum_{c=1}^{C} \sigma_j^{(c)} \log_2 \sigma_j^{(c)}. \tag{20}$$

Instead, the term in $\mathrm{CCE_{CV}}$ that uses training outcome as an estimator is

$$B_j^{(\mathrm{CV})} = -\sum_{c=1}^{C} \sigma_j^{(c)} \log_2 s_j^{(c)}. \tag{21}$$

The difference between these quantities is computed as

$$\Delta B = -\sum_{c=1}^{C} \sigma_j^{(c)} \log_2 \frac{s_j^{(c)}}{\sigma_j^{(c)}}. \tag{22}$$

Expression (22) is the Kullback–Leibler distance between the empirical distribution of classes in test data, $\sigma_j^{(c)}$, and the distribution expected from training results, $s_j^{(c)}$. This (nonnegative) quantity gives, for the jth cluster, the number of extra bits deriving from the mismatch between empirical (cross-validation) and expected (training) distributions. By construction and neglecting constant terms, $\mathrm{CCE_{CV}}$ as computed in (19) averages such distortion over the entire test set. □

Cross-validation CCE can be used as a description-length measure of distortion in estimating local class distribution. The practical usefulness of this process lies in the possibility of estimating the overall generalization performance on a limited sample of data. There is a clear practical interest in relating the cost function at cross-validation, $\mathrm{CCE_{CV}}$, with the corresponding generalization performance; the following property states such relations in terms of bounds on classification error.

Property 5 (Bounds on CCE—Cross Validation): Let ε_v denote the total classification error on cross-validation data. The following bound can be set on the value of $\mathrm{CCE_{CV}}$

$$\varepsilon_v \leq \mathrm{CCE_{CV}} \leq \infty. \tag{23}$$

Proof: Let $\hat{c}_j$ denote the predominant class in the jth cluster resulting from calibration after training; class distribution is such that: $\forall c \neq \hat{c}\ s_j^{(c)} \leq \frac{1}{2} \Rightarrow -\log_2 s_j^{(c)} \geq 1$. Substituting this result in (19) gives

$$\begin{aligned}\mathrm{CCE_{CV}} &= -\sum_{j=1}^{J} \sigma_j \left(\sigma_j^{(\hat{c})} \log_2 s_j^{(\hat{c})} + \sum_{c \neq \hat{c}} \sigma_j^{(c)} \log_2 s_j^{(c)} \right) \\ &\geq \sum_{j=1}^{J} \sigma_j \left(0 + \sum_{c \neq \hat{c}} \sigma_j^{(c)} \right) = \sum_{j=1}^{J} \sigma_j \varepsilon_{vj} = \varepsilon_v.\end{aligned}$$

The lower bound ensures that $\mathrm{CCE_{CV}}$ is a consistent estimator of the classification error. As compared with the lower bound for training, convergence is not as fast, since the doubling factor is missing.

Unfortunately, a finite upper bound on cross-validation cost cannot be stated in the general case: there may exist a sample distribution such that: $\sigma_j > 0, s_j = 0$ (i.e., some test data show up in a cluster not covered by the training set), which invalidates the argument of the logarithm, hence

$$\mathrm{CCE_{CV}} \leq \infty \qquad \square.$$

An upper bound to cross-validation cost would have allowed one to assess a system's generalization ability. The lack of such an expectation raises the crucial problem of controlling a network's generalization performance during training. This basic issue has been extensively studied in the literature, following both practical and theoretical approaches. The following section addresses this problem in both ways, and shows how some specific features of CCE-based networks can be exploited to that purpose.

III. Generalization Issues in CCE Networks

The training algorithm described in Section II-B pursues CCE minimization by sequentially adding and adjusting neurons until the cost function nullifies. In theoretical tratments of such classifiers, "convergence" typically means reaching complete, exact classification of training samples. In fact, training a classifier to the point where it correctly classifies 100% of the training data often results in extremely poor generalization. Besides, one has to provide some mechanism

for preventing exponential growth in either the size of the network or the number of steps in the algorithm. Although the VC dimension of the "circular" node is not much different from the linear node [6], allowing a network to grow arbitrarily might lead to solutions with very large VC dimensions due to a huge number of parameters.

Those issues ultimately call for a methodology that controls network complexity. The lack of a gradient-based optimization algorithm makes dynamic techniques like weight decay practically unfeasible. Otherwise, inflating the training set by injection of *random noise* may improve generalization performance [32], [33]; such an artificial data processing enhances generalization by "smoother" separation surfaces. Due to the plastic training algorithm, this *per se* does not limit the eventual number of nodes, hence *pruning* algorithms therefore seem the most promising approach to reducing network complexity in CCE-based classifiers. A network is let grow unconstrained, then complexity is reduced by some pruning algorithm affecting weights, nodes, or both [34]. This section first describes and sets the proper conditions for noise injection to improve local separation surfaces, then presents two algorithms to reduce network complexity.

A. Noise Injection

Adjusting the weights of a newly added unit implies the training of a perceptron. The discrete nature of the digital cost function and the random-search strategy of the training algorithm do not ensure that the eventual solution is optimal at generalization—a boundary may yield null cost on training data without best reflecting the true data distribution. In this sense, the addition of some random noise to training samples aims to improve the error margin associated with the decision boundary. This involves the artificial replication of each training sample $\mathbf{x}_l$ into a number U of additional distorted copies associated with the same class

$$\mathbf{x}_l^{(u)} = \mathbf{x}_l + \mathbf{r}^{(u)} \quad u = 1, \cdots U; \quad C\left(\mathbf{x}_l^{(u)}\right) = C(\mathbf{x}_l) \forall u \quad (24)$$

where $\mathbf{r}^{(u)}$ is a random vector having the same dimensionality, $d(0)$, of training samples. The beneficial effects on training are balanced by an increased computational cost of the training process, since 1) the number of processed samples is increased and 2) boundaries are more difficult to work out.

Fig. 4 gives a sketch of the underlying principle of operation. The progression of boundaries obtained with different noise levels shows the possible beneficial effects of noise injection to the "quality" of the resulting error margin. The example also indicates that the amount of injected noise heavily affects the ultimate result.

Estimating the distortion level properly is critical [33], [32]; the research presented here describes an analytical method to work out the maximum amount of tolerable injected noise. The method takes into account both the specific neural problem (perceptron training) and the empirical distribution of data (classes and positions of samples).

In the lack of specific information about the spatial distribution probability of samples, an isotropic model of noise should be used. Under this assumption, noise is modeled by

$$\mathbf{r} = r_0 \mathbf{u}; \quad r_0 \geq 0; \quad \mathbf{u} \in [-1, +1]^{d(0)} \quad (25)$$

where the random vector $\mathbf{u}$ is uniformly distributed in the hypercube $[-1, +1]^{d(0)}$. Noise energy is a random variable, whose instantaneous and average values, ρ and $\bar{\rho}$, respectively, are computed as

$$\rho = (r_0)^2 \sum_{i=1}^{d(0)} u_i^2; \quad \bar{\rho} = \frac{d(0)}{3}(r_0)^2. \quad (26)$$

The "worst" pair of data is the pair of closest samples that belong to different classes; let δ^2 denote the (square) distance between such samples

$$\delta^2 = \min_{i,j}\{\|\mathbf{x}_i - \mathbf{x}_j\|^2, C(\mathbf{x}_i) \neq C(\mathbf{x}_j)\}. \quad (27)$$

To evaluate the maximum noise, one imposes that, on average, the "cloud" of random-generated artificial samples does not overlap with training samples of different classes. This implies that the space region around a noise-inflated sample lie within a given range from the closest training sample belonging to a different class. If K indicates the factor controlling the cloud overlap (typ., $K = 1$), the expression for the optimal noise level stems from imposing the above constraints in (26) and (27)

$$\bar{\rho} \leq \delta^2 \Rightarrow r_0 \leq \sqrt{K \frac{3 \cdot \delta^2}{d(0)}}. \quad (28)$$

It is worth stressing that noise injection aims to improve the classifier's structure rather than to limit its complexity; the number of neurons might increase arbitrarily if no growth-control mechanism is provided. In the practical approach presented here, one typically sets a fixed, upper limit to the total number of neurons during network construction, then applies pruning to remove the inessential units that might have been generated erroneously by early optimization cycles.

B. Use of CCE Networks for Data Analysis

The possibility of listing the samples belonging to a partition is a crucial feature of CCE networks; indeed, the labeling of space regions gives this neural model a specific effectiveness at data analysis which is not easily shared by other approaches, and which makes it possible to use CCE networks for domain-space analysis. The basis for this task is the hierarchical arrangement of symbols and their one-to-one mapping to data partitions, which will be used extensively in the pruning and clustering algorithms. The theoretical treatment will use the following definitions.

Definition (Depth of a Symbol): The depth of a symbol, $D(\mathbf{S}_n)$, is defined as the number of bits *explicitly* composing $\mathbf{S}_n$ (i.e., a symbol's depth counts only the bits defined before any "freezing" operation). In the example of Fig. 2, $D(\mathbf{S}_1) = 1$, $D(\mathbf{S}_2) = 2$, $D(\mathbf{S}_3) = D(\mathbf{S}_4) = 3$.

Definition (Expansion of a Symbol to a More Specific One): Consider two symbols, $\mathbf{S}_n$, and $\mathbf{S}_m$, such that $\mathbf{S}_n$ lies at a higher hierarchical level than $\mathbf{S}_m$, i.e., $D(\mathbf{S}_n) < D(\mathbf{S}_m)$. The expansion of $\mathbf{S}_n$ to $\mathbf{S}_m$ is defined as the bit configuration, $\mathbf{S}_{n\to m}$, such that

$$\mathbf{S}_{n\to m} = \{b_{n\to m}^{(w)}, w = 1, \cdots, N\};\ b_{n\to m}^{(w)} = \begin{cases} b_n^{(w)}, & \text{if } w \le D(\mathbf{S}_n) \\ b_m^{(w)}, & \text{if } w > D(\mathbf{S}_n) \end{cases};\quad C(\mathbf{S}_{n\to m}) = C(\mathbf{S}_n). \tag{29}$$

An expanded symbol has the same label of the higher-level symbol. In the example of Fig. 2, $\mathbf{S}_{1\to 4}$ = "011" and $C(\mathbf{S}_{1\to 4})$ = "×." It is most important to stress that, in force of property (5) and depending on W, not all bit configurations represent valid symbols (as is the case, in Fig. 2, for "101"). Therefore, expanded symbols require a validity check. This can be carried out either empirically, by checking whether a sample yields the considered symbol, or analytically, by verifying whether the set of inequalities associated with the bit configuration admits valid solutions in the domain space.

Definition (Neighboring Partitions): Two partitions are neighboring if their associated symbols 1) are both valid and 2) are at unity Hamming distance. Likewise, two partitions are w-neighboring if they are neighboring and their associated symbols differ in the wth bit position (i.e., the wth neuron separates them).

Definition (Strongly Compatible Partitions): Two partitions, $\mathbf{S}_1$, $\mathbf{S}_2$, are strongly (w-) compatible if they are (w-) neighboring, and are labeled with the same class. Two (w-) neighboring partitions that are both unlabeled are strongly (w-) compatible. The operator that indicates strong compatibility will be denoted as $C_!(\mathbf{S}_1, \mathbf{S}_2)$, returning one iff $\mathbf{S}_1$ and $\mathbf{S}_2$ are strongly compatible, and zero otherwise. Likewise, $C_!^{(w)}(\mathbf{S}_1, \mathbf{S}_2)$ denotes strong w-compatibility.

Definition (Weakly Compatible Partitions): Two partitions, $\mathbf{S}_1, \mathbf{S}_2$, are weakly ($w$-) compatible if they are (w-) neighboring and only one of them is labeled. The operators indicating weak compatibility, and weak w-compatibility will be denoted as $C_?(\mathbf{S}_1, \mathbf{S}_2)$ and $C_?^{(w)}(\mathbf{S}_1, \mathbf{S}_2)$, respectively, returning one iff $\mathbf{S}_1$ and $\mathbf{S}_2$ are weakly (w-) compatible, and zero otherwise.

C. Pruning Algorithm

Sequential training conveys inefficiences in neuron placements, especially in the early algorithm steps and with complex data distributions. The neuron linearity and the plastic training strategy may generate redundant boundaries whose effect is overriden by later ones; in such cases irrelevant neurons can be removed. The pruning process seeks those "cuts" in the data space whose actions are overriden by subsequently added units.

The principle of operation for pruning is that a redundant cut may not separate two partitions that are labeled with different classes. In other words, if two incompatible regions share a boundary, then the neuron spanning that boundary may not be removed. Conversely, the wth neuron can be eliminated if all pairs of symbols lying across its associated boundary are w-compatible. Some issues might arise if some space regions lying on boundaries are unlabeled; their treament ultimately depends on performance requirements and are imposed by the target application. In particular, one might choose a conservative strategy, which privileges consistency over network simplification; in this case, strong compatibility between neighboring symbols should be required to allow boundary pruning. Conversely, accepting weak compatibility to permit boundary cancellation implies a kind of "optimistic" strategy that favors network simplification. The latter approach makes for a larger number of pruned nodes, hence it will be adopted in the algorithm formulation shown at the bottom of the page. The algorithm operation is illustrated in Fig. 5, where the generalized-XOR problem is used as a testbed [22]. Fig. 5(a) presents the initial situation after network training, where neurons are marked by their progressive insertion order. Fig. 5(b) highlights a pruned unit (neuron 1), and points out the pairs of compatible symbols that are considered by the pruning algorithm. Fig. 5(c) presents the final situation after pruning all irrelevant units.

D. Clustering Algorithm

Network simplification is not the only opportunity that is offered by the explicit labeling of space regions; indeeed, one might exploit the concept of neighborhood among partitions to inspect the internal structure of the domain space. In this case, the analysis typically aims to assess the number and composition of space portions that have a topological

---

0) *Input*: a trained network, including W units and spanning N symbols over the data space;
1) *For each* neuron $w = 1, \cdots, W$:
  1.A *For each* symbol, $\mathbf{S}_n, n = 1, \cdots, N-1$:
    For each symbol, $\mathbf{S}_m, m = n+1, \cdots, N$:
      begin
      a. *If* $D(\mathbf{S}_n) = D(\mathbf{S}_m)$ *set* $\mathbf{S}^\star = \mathbf{S}_n,\quad \mathbf{S}^+ = \mathbf{S}_m$
        elseif $D(\mathbf{S}_n) < D(\mathbf{S}_m)$ *set* $\mathbf{S}^\star = \mathbf{S}_{n\to m}, \mathbf{S}^+ = \mathbf{S}_m$
        else *set* $\mathbf{S}^\star = \mathbf{S}_{m\to n}, \mathbf{S}^+ = \mathbf{S}_n$
      b. *If* $C_?^{(w)}(\mathbf{S}^\star, \mathbf{S}^+) = 0$ abort the check for the wth unit and *goto* Step 1
      end
  1.B *Mark* the wth neuron as 'pruned'
2) *Eliminate* pruned neurons
3) *Recompute* the symbol table.

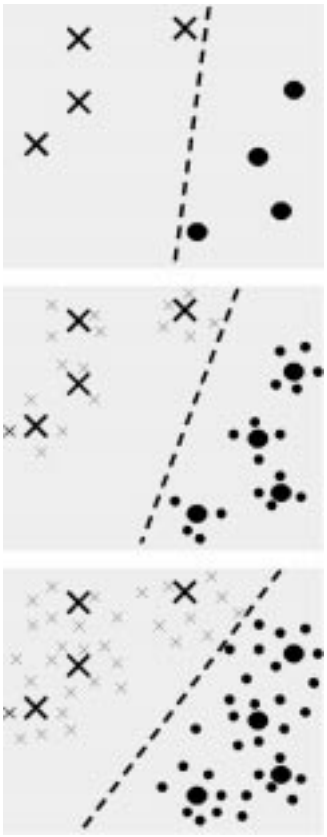


Fig. 4. The effect of noise injection on the separating boundaries. (a) Rendom-search training on the original data set may lead to boundaries with poor error margin. (b) and (c) Increasing noise injection improves the margin quality of the separating boundary.

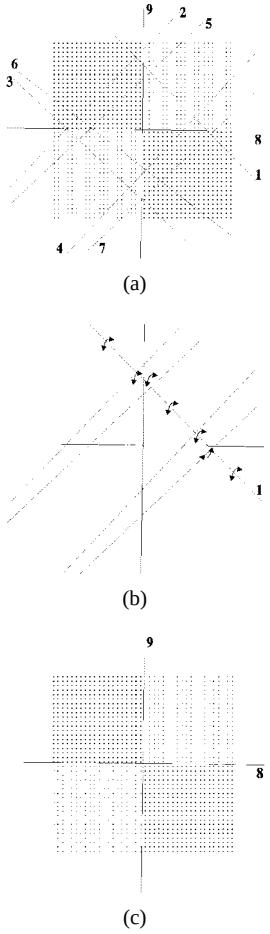


Fig. 5. Operation of the pruning system for the generalized-XOR problem. (a) Situation before pruning. (b) Pruning context for Neuron 1. (c) Final situation.

significance for classification. Therefore, the inspection starts from the collection of elementary partitions labeled by a trained CCE-network, and leads to complex structures that might help understand the underlying problem nature.

Such operation is here referred to as "clustering," and takes full advantage of the possibility of directly observing neighboring regions. In the present context, a cluster is defined as a nonempty set of pairwise-contiguous partitions that are labeled by the same class; the cluster is labeled accordingly.

The principle of operation of clustering is that two elementary partitions (as well as entire clusters) may merge when 1) they are labeled with the same class and 2) they face across a segment of a common boundary. This definition again involves the concept of neighborhood among partitions, whereas the constraint on equal classes requires strong compatibility. As opposed to pruning, the clustering process intentionally does not adopt weak neighborhood, so that undecided space regions may be preserved in the eventual structural representation.

Thanks to its most general definition, a cluster may span a quite complex portion of the data space, which typically cannot be represented by elementary boundaries easily. On the other hand, the number of compact subregions shrinks drastically, hence an analyst may get some hint about the number and displacement of significant zones in the data distribution. Besides, if a cluster is regarded as a collection of digital symbols, any classical reduction algorithm from binary logic can apply to the bit configurations to sinthesize out simplified expressions. This approach to simplification resembles a rule-extraction process. It does not guarantee, however, a decrease in a classifier's complexity and VC-dimension, since intercluster boundaries may be so intricate that no reduction is possible in their representation. Therefore, one might say that clustering is useful in simplifying interpretation rather than representation.

In the following outline of the clustering algorithm, Π_u will denote the uth cluster, and $C(\Pi_u)$ its associated class. By obvious extension, two clusters are mergeable when they include two partitions that are mergeable as shown in the algorithm at the bottom of the next page. It is easy to prove that, in force of the condition imposed at step 1.B.b, all symbols and related space regions included in a cluster will belong to a same class. In addition, thank to the neighborhood consistency, there exists a finite and uninterrupted walk connecting all points within a cluster; in other words, the space portion defined by a cluster is compact.

The clustering algorithm has a limited computational cost (since it operates at the symbol level); in addition, it applies directly to multiclass problems and possibily quite complex data distributions. It is worth noting that the clustering algorithm

is totally independent of the pruning algorithm previously discussed; indeed, the clustering action converges to the same solution when it is applied either before or after network pruning. In the generalized-XOR problem, for example, the algorithm invariantly yields four clusters for both the situation in Fig. 5(a) and that in Fig. 5(c). The operation of the clustering algorithm is exemplified in Fig. 6, where the CCE network faces an artificial classification problem spanned by five random lines in the two-dimensional (2-D) plane. Fig. 6(a) presents a sample distribution of training data (the test was repeated several times successfully on different data sets). Fig. 6(b) displays the space tessellation as obtained by the trained network, which includes seven neurons; symbols at different hierarchical levels are apparent. Fig. 6(c) presents the result of the clustering algorithm, which converges to four clusters, and demonstrates the possibly complex nature of the aggregated regions.

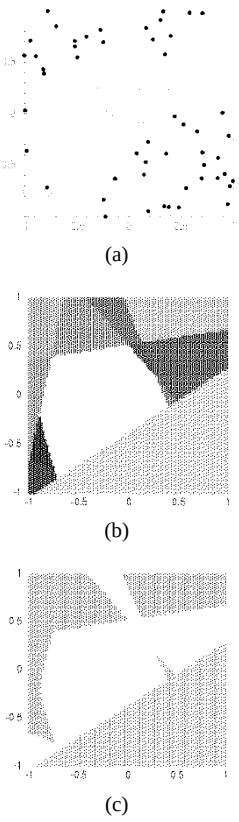


Fig. 6. Sample operation of the clustering algorithm on a random-net learning test.

IV. Experimental Results

The experimental evaluation of the described methods and algorithm aims to verify the consistency of the derived theoretical properties, and the efficacy of the various criteria proposed for network construction. To this end, the empirical analysis took into account several different domain problems, and tested each of them under several different conditions, mainly to remove any bias from estimated results.

The problems considered include both artificial domains, which can drive an easier (typically visual) interpretation of the methods' performances, and real-world testbeds, which allow significant comparisons with related achievements in the literature. The empirical approach conveyed a remarkable experimental effort, hence a summary of obtained results will be here presented; the presentation is arranged according to the various domains considered.

---

0) *Input*: a trained network, including W units and N symbols; $U = 0$;
1) *For each* symbol $\mathbf{S}_n,\ n = 1, \cdots, N$
Begin
1.A *If* does not exist $\Pi_u\ u = 1, \cdots, U$ such that $\mathbf{S}_n \in \Pi_u$ (create new cluster if necessary)
Set $U = U + 1;\ \Pi_U = \{\mathbf{S}_n\};\ C(\Pi_U) = C(\mathbf{S}_n)$;
1.B *For each* symbol $\mathbf{S}_m,\ m = n+1, \cdots, N$: (match symbol with table)
Begin
1.B.a. *If* $D(\mathbf{S}_n) = D(\mathbf{S}_m)$ (expand higher symbol if necessary)
Set $\mathbf{S}^\star = \mathbf{S}_n, \quad \mathbf{S}^+ = \mathbf{S}_m$
Elseif $D(\mathbf{S}_n) < D(\mathbf{S}_m)$
Set $\mathbf{S}^\star = \mathbf{S}_{n \to m}, \mathbf{S}^+ = \mathbf{S}_m$
Else
Set $\mathbf{S}^\star = \mathbf{S}_{m \to n}, \mathbf{S}^+ = \mathbf{S}_n$
1.B.b. *If* $C_!(\mathbf{S}^\star, \mathbf{S}^+) = 1$ (Allow only strong compatibility)
Begin
If does not exist Π_v such that $\mathbf{S}_m \in \Pi_v$ (If the symbol has not been assigned yet)
Set $\Pi_u = \Pi_u \cup \{\mathbf{S}_m\}$ (join $\mathbf{S}_m$ to Π_u)
Else
Begin (merge clusters)
Set $\Pi_u = \Pi_u \cup \Pi_v$
Delete Π_v; *Set* $U = U - 1$;
End
End
End
End

A. Generalized XOR

This testbed [22] conveys a classification problem that most stresses a CCE network's representation power. The optimal solution requires only two units whose boundaries coincide with the coordinate axes. As far as incremental CCE training is concerned, the problem is especially interesting because any plastic optimization-based approach would fail to catch the problem's nature and would necessarily miss the simplest solution. Indeed, to attain optimality, the first neuron must draw a boundary along one axis, accepting a singular partitioning that does not decrease uncertainty: however, when using incremental network construction, any optimization approach will reject the (correct) solution at the first step, and will choose one of the many partitionings along nonsingular directions that decrease CCE.

A sample solution found by CCE training involving linear units has already been shown in Section III-C. The problem's difficult topology has been used in that situation to demonstrate the efficacy of the pruning algorithm. In this specific testbed, the circular model does not bring in a significant advantage over linear units; however, in both cases the pruning algorithm ends up with two neurons and the clustering algorithm converges to the correct solution of one cluster for each quadrant.

B. Nested Spirals

This quite famous testbed [7] involves a highly nonlinear class distribution. A backpropagation network requires three hidden layers of sigmoidal neurons [35] to represent the problem with a considerable optimization effort. Including circular neurons in multilayer perceptrons makes it possible to classify samples correctly with one layer of at most six units [6], convergence failures are most rare.

When applied to the nested-spirals testbed, CCE networks with linear units require more than 30 neurons and the resulting representation may appear unsatisfactory with respect to the data distribution. The use of circular units improves the quality of problem solution in both training and generalization. The number of neurons decreases to a range of 8–10, and Fig. 7 (left, middle) allows a visual comparison of the resulting problem representations. The advantage derives from the fact that the circular model is a superset of conventional perceptrons and simply covers a wider class of problems with a limited increase in complexity.

The nested-spiral problem can demonstrate visually the enhancements in generalization performance brought about by the techniques described in Section III. Fig. 7(c) displays the final representation resulting from a joint application of 1) noise injection for network training and 2) the clustering algorithm. To obatin the result displayed in the picture, the original data set (including 97 points $\times$ 2 classes) has been artificially inflated tenfold, using a noise level estimated according to theoretical prediction (28). The network's number of neurons has been limited to 30 for stopping training. The clustering process has been then applied to the space partitioning spanned by the trained network, yielding to the three elementary clusters distinguished by the different colors. The experimental evidence presented is merely a sample of the several tests performed. All experiments on this problem led to similar results, and are not displayed here for brevity.

(a) (b) (c)

Fig. 7. Sample solutions for the nested-spirals problem. (a) and (b) Network mappings using linear and circular neurons (without noise injection). (c) The three clusters identified by the clustering algorithm.

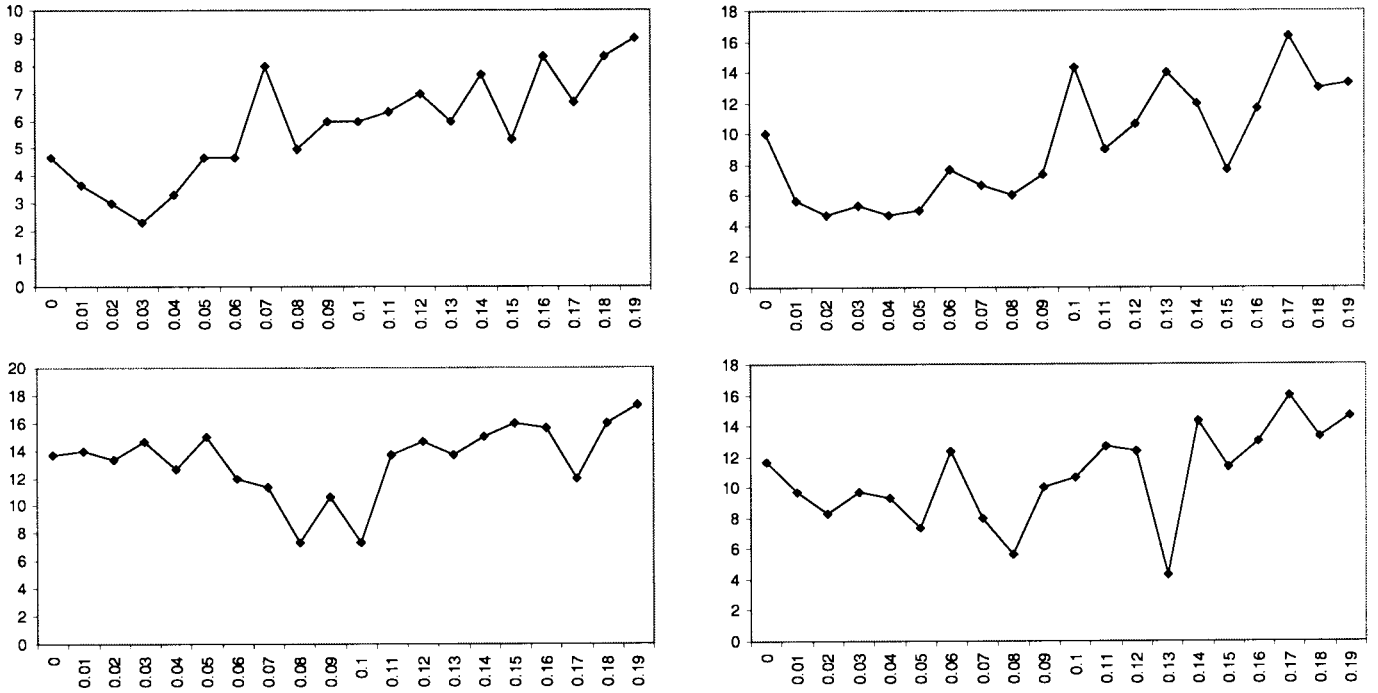
The results on such a complex testbed are significant mainly because they confirm theoretical expectations about the noise-injection criterion. In addition, the outcome of the clustering algorithm witnesses the power of the data-inspection tool that we obtain from joining complexity control (noise injection) with posttraining simplification (clustering).

C. Random Teacher Network

These tests investigate the performance of CCE networks when learning the classification schema spanned by a set of random linear dichotomies. Choosing a two-dimensional data space facilitates interpretation of results; a sample solution for one of the several runs performed has already been given in Section III-D—Fig. 6.

Such experiments involved a massive computational effort, involving problems at increasing complexity: several runs used an increasing number of random dichotomies over the unit square $[-1, +1]^2$ (typ. 15 runs for each schema, ranging from two to seven dichotomies). For each random schema, a training data set has been sampled and used to build a CCE-based classifier; due to the hyperplane-based schema, only linear units have been considered. Network training has involved noise-injection at increasing noise levels; generalization has been evaluated by measuring classification accuracy over the entire square interval.

The graphs in Fig. 8 give the results obtained for a subset of such experimental cases; similar achievements have been obtained in all tests, and are not shown here for brevity. In the



(a) (b)

Fig. 8. Classification results at increasing noise levels for the random-network problem. (a) Five-dichotomies case. (b) Seven-dichotomies case.

graphs, each point represents a network-training result: the x-axis gives the noise level adopted to inflate the basic training set, whereas the y-axis measures the overall generalization error measured for the trained network. The results confirm the validity of noise-injection: classification error has a minimum at a nonnull noise level, which depends on the specific problem considered, and that matches theoretical expectation satisfactorily, as will be statistically supported in the next sections.

D. Sonar

Training runs on this testbed served to verify the advantage of circular units over linear ones in a real and very well-known domain [9], that involves a two-class problem and a 60-dimensional data space; the training set includes 104 samples. The classification problem is known to be linearly separable [36], and both approaches actually converged to the optimal solution after a considerable optimization effort. In this sense, the reported results relate to suboptimal solutions with more than one neuron, yet allow a comparison between the two neuron models.

To ensure statistical significance, the comparative analysis involved forty pairs of training runs, including the linear and the circular neuron models under equal optimization conditions and with equal parameters. Thus any test sample consisted in the number of units required for convergence on the training set; in practice, this number always kept in the range $2 \leq H \leq 4$.

Table I gives the empirical distribution of number of units for the two neuron models, showing a marked preference for the circular model, which typically requires fewer neurons. Table II gives some statistical parameters summarizing the convergence results. The superior performance of the circular model is confirmed by such measures, for which Student's t test indicates that the two means are significantly different $(P < 0.05)$. Indeed, the 95% confidence intervals for the given average and variance values are $(2.42, 2.83)$ and $(2.76, 3.14)$ for the circular and the linear neuron models, respectively.

TABLE I
DISTRIBUTION OF UNIT NUMBERS

Neuron model	Number of runs		
	w/ 2 neur.	w/ 3 neur.	w/ 4 neur.
Linear	8	26	6
Circular	18	19	3

TABLE II
STATISTICAL SUMMARY

(40 runs)	Avg. neurons	Variance
Linear	2.950	0.356
Circular	2.625	0.394

The graph in Fig. 9 plots test error percentage versus increasing noise level for both linear and circular neurons, and demonstrates the efficacy of noise-based generalization control. Remarkably, best generalization is attained by the predicted noise level. A final remark concerns the optimization effort required for network training (for $H \geq 2$), which appears impressively smaller (less than 70 000 random search steps) than the equivalent effort for backpropagation training.

E. BCancer

Wisconsin's Breast Cancer database [10] collects 699 cases for such diagnostic samples. This testbed involves a two-class discrimination problem, and the two categories are almost balanced in the original patient set. The data dimensionality is 10, but patient's ID can be removed for the present purposes, hence a nine-dimensional domain was considered. In addition,

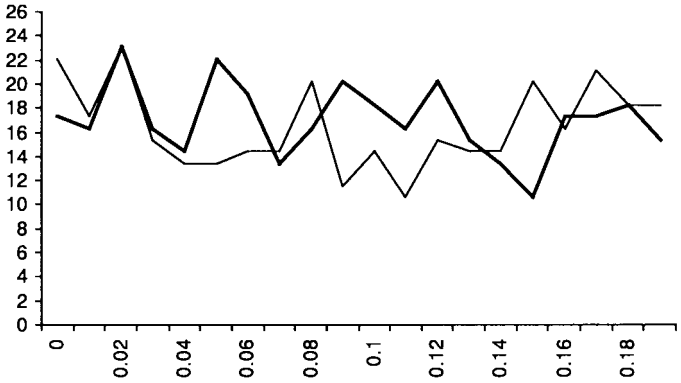


Fig. 9. Generalization perf. versus noise level—"sonar." Thin line: linear units; thick line: circular units.

the 16 sample cases including "missing" values were removed from the data set, whose cardinality eventually amounts to 683. Since the numerical representation of features for each patient spans a quite wide numerical range, the data were normalized along each coordinate axis.

In order to assess generalization, the sample set was split randomly into a training and a test set for cross-validation; the training set included 400 samples, whereas the test set included the remaining 283. This procedure was repeated several times to remove possible bias deriving from the random sampling. The graphs in Fig. 10 presents a sample result of the generalization performance for the linear-neuron classifier; the average advantage of using noise-injection is apparent and also matches theoretical predictions, as will be further detailed in the following.

F. Phoneme

The experiments on the Phoneme database [11] consider network performance from a twofold perspective: first, to compare the classification accuracy resulting from either circular or linear units; second, to verify experimentally the validity of the noise-injection procedure on a complex, real-world problem. The Phoneme problem is interesting because it demonstrates the general applicability of the CCE-based approach in a multiclass classification problem, in which the ability of segregating space portions becomes crucial to the overall classifier's performance.

In each run, the original database has been half-split at random into a training and a test set. The experiments followed the same procedure adopted for the previous testbed. Likewise, the graphs in Fig. 11 plot generalization error versus noise level; two curves are given, one for each kind of unit. The results obtained support the validity of the noise-injection method for controlling generalization performance, no significant difference has been detected between linear and circular boundaries.

G. Vowel

The "vowel" testbed [37] provides a challenging 11-class case study involving speech identification. The database includes a training set and a test set, made up of 528 and 462 samples, respectively; the data-space dimensionality is 10. The cited work on this testbed [12] allows a comparative

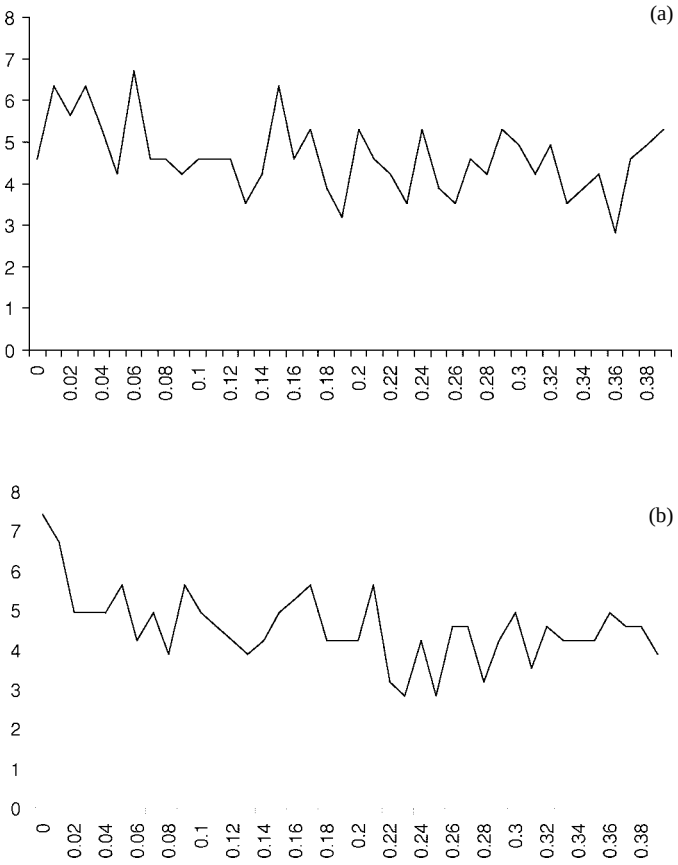


Fig. 10. Generalization perf. versus noise level—"BCancer." (a) linear units; (b) circular units.

analysis of a method's accuracy and generalization ability. In a first set of experiments, for simplicity of implementation and without loss of generality, the classification task was divided into several twin-class problems. A network was trained to discriminate each class from the remaining ones, which were in turn unified under a unique label; this mapped one 11-class problem into 11 two-class subtasks. Those experiments considered the different behaviors of linear and circular units. For each class, five networks were brought to convergence on training data, and the resulting average number of neurons was worked out. The same process, under equal optimization conditions, was performed using both the linear and the circular neuron models. The total number of trained networks (110) witnesses the massive experimental effort involved. Table III gives the average number of units required for each class, and confirms the overall advantage of circular units over linear ones; the odd behavior for class 5 is a consequence of imperfect optimization. Interestingly, evidence also shows that Class 1 can be separated from the other classes by one circular unit.

An additional set of experiments studied the consistency of noise-injection for generalization control. Therefore, no exact training was pursued, but rather the number of neurons in a network was *a priori* limited to a maximum of 30; classification on the test set gave the actual evaluation parameter; all categories were considered simultaneously in a multiclass network-building problem. Fig. 12 reports the classification-error curves (using the conventions previously adopted), and

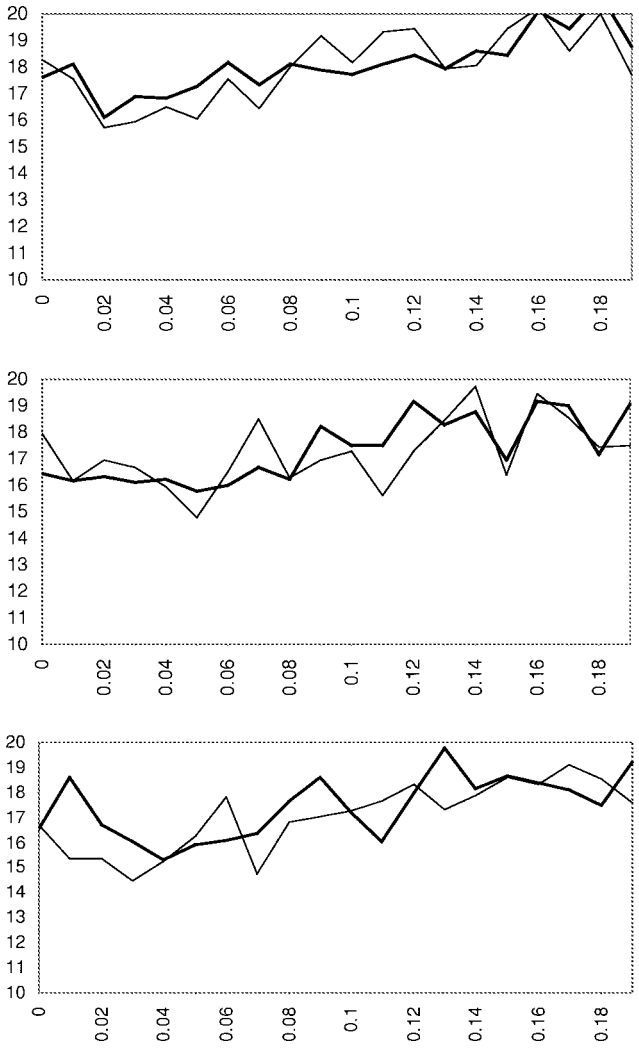


Fig. 11. Generalization error versus noise level for three sample runs on the phoneme database. Thin line: linear units; thick line: circular units.

TABLE III
"VOWEL" TRAINING

Class	Avg. No. of Units	
	Linear	Circular
1	2	1
2	4.2	4.2
3	4.6	3.2
4	4.6	4
5	5.2	5.7
6	6.4	6.3
7	6.6	5.3
8	4	2.8
9	6.8	6.1
10	3.8	3.4
11	5.6	5.6

proves both the benefit of noise injection to generalization and the advantage of circular units over linear ones.

H. Vowel2D

The "Vowel2D" database [13] resembles the previous one since it relates to a very similar problem (i.e., speech recognition). This testbed is interesting since it is defined over a two-dimensional space, hence some visual inspection of the classifier's behavious is feasible [13]. Vowel2D is a multiclass

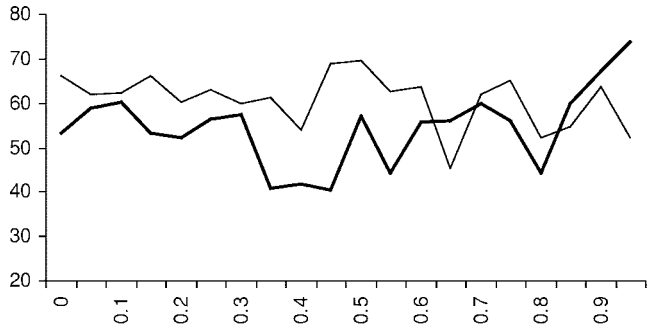


Fig. 12. Classification error % versus noise level for "vowel." Thin line: linear units; thick line: circular units.

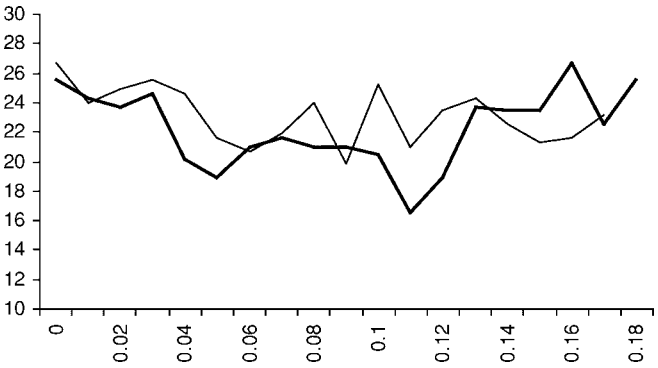


Fig. 13. Gereralization error % versus noise level for the "Vowel2D" testbed.

(a) (b)

Fig. 14. Domain partitioning for the "Vowel2D" testbed. (a) linear units; (b) circular units.

problem $(C = 10)$ of peculiar difficulty, since classes are quite intermixed within one another; the database is split into a training and a test set, including 338 and 333 samples, respectively.

The difficulty of this real-world classification problem give the experiments some additional validity in confirming the generalization-control mechanisms. In particular, the difference in behavior between linear and circular units is apparent, and the consistency of the noise-injection criterion is strongly validated. As usual, Fig. 13 compares the related classification error curves at increasing noise level, whereas Fig. 14-left) and right) show the domain partitioning when using linear and circular units, respectively.

I. Numerical Validation of the Noise-Injection Criterion

This section reconsiders the noise-injection criterion and gives statistical evidence of the theoretical prediction given in (28), exploiting the experimental results from the testbeds previously illustrated. Table IV summarizes the relevant quantities for the analytical computation of the optimal noise

TABLE IV
THEORETICAL ($K = 1.5$) AND EXPERIMENTAL NOISE LEVELS AT OPTIMAL GENERALIZATION

Testbed			Noise Level	
Name	d_0	δ^2	Theor.	Actual
XOR	2	0.05	0.075	0.03
Spirals	2	0.14	0.176	0.1
RndNet(5) I	2	0.047	0.0705	0.09
RndNet(5) II	2	0.07	0.1050	0.07
RndNet97) I	2	0.047	0.0705	0.05
RndNet(7) II	2	0.03	0.0450	0.09
Sonar	60	0.507	0.1388	0.11
BCancer	9	0.529	0.37	0.38
Phoneme I	5	0.045	0.0427	0.03
Phoneme II	5	0.031	0.0294	0.04
Phoneme III	5	0.048	0.0455	0.05
Vowel	10	0.427	0.2864	0.35
Vowel 2D	2	0.0095	0.013	0.05

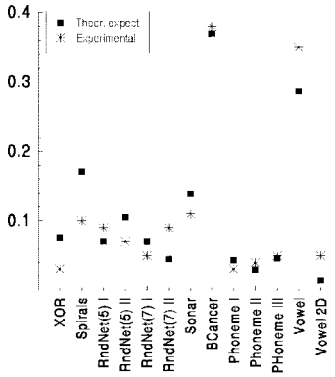


Fig. 15. Comparison of theoretical ($K = 1.5$) and experimental noise levels at optimal generalization.

level, and compares theoretical expectations with the actually measured values for the various testbeds. The comparison is also illustrated graphically in Fig. 15.

From experimental measures it was established that the optimal value for parameter K in expression (28) is $K = 1.5$, which implies that the artificial cloud of patterns around an original sample may extend beyond its maximum limit. Such adjustment is mostly general and is not tuned to any specific testbed. The increase in K is explained by considering that the theoretical expectation for noise level has been obtained from the *worst* case, i.e., the pair of closest heterogenous samples; in fact, the statistics of intersample distances over the whole training set is wider and the related constraint may be relaxed.

The data reported give evidence of the consitency of the noise-injection criterion; Pearson's correlation index among the two sets of values, amounting to $r = 0.9349$ $(P < 10^{-6})$, strongly validates the noise-injection estimator on a statistical basis. It is worth stressing that a very similar result $(r = 0.9350, P < 10^{-6})$ was obtained when using $K = 1$, so that one may conclude that the above-mentioned correction is not essential to the overall method's validity.

V. CONCLUSION

The major apparent difference between CCE and backpropagation networks for classification is the entropy-based cost function, which determines a different behavior in setting sepration boundaries, and imposes a different strategy in optimizing a network's parameters. However, a crucial feature of CCE models is that they proceed by partitioning and labeling regions of the data space explicitly. Such ability to represent the empirical data distribution at the local level is a major advantage of the overall approach, which widely makes up for the (possibly) cumbersome use of nongradient-based algorithms for training.

In this regard, the adoption of a plastic model for network building aims to limit the huge computational cost involved by random-search optimization. The consequent danger of combinatorial explosion in the network's complexity (involving poor generalization) is compensated by posttraining algorithms, which remove the inefficiencies conveyed by the plastic approach in the early phases of the training process. The pruning and the clustering algorithms reduce a network's complexity by cutting inessential parameters and by simplifying representation, respectively. Conversely, artificially inflating the training set by controlled noise injection enhances generalization performance, since the resulting separation boundaries exhibit a wider margin of classification error.

The techniques described in this paper for improving generalization should be therefore regarded as an integrated approach to network construction. In particular, noise injection improves boundary placement, whereas joining plastic growth with pruning trades off a lighter computational load for a wider memory occupation. This seems to place the overall approach in a memory-based paradigm for network construction that is recently getting increasing interest, especially in view of the constantly decreasing cost of hardware implementation.

The experimental evidence collected in the reported research also indicated that the CCE-based approach never suffered (or even came close to suffer) from the above-mentioned combinatorial explosion in symbol labeling. This empirical result holds for both artificial and (especially) real-world domains, and directly confirms the equivalent observations preliminary reported in [1].

From this viewpoint, the current lines of research in this field include the design of analog VLSI architectures supporting the presented paradigm, which should greatly reduce development costs in terms of optimization time. At the same time, the development of specific and more sophisticated training algorithms is also being investigated, thus leading to an integrated paradigm unifying CCE-based and classical backpropagation-based neural models.

ACKNOWLEDGMENT

The authors wish to thank the reviewers, whose comments were very stimulating and greatly helped improve the quality of the paper contents and overall presentation.

REFERENCES

[1] M. Bichsel and P. Seitz, "Minimum class entropy: A maximum information approach to layered networks," *Neural Networks*, vol. 2, pp. 133–141, 1989.
[2] R. Linsker, "Self-organization in a perceptual neural network," *Computer*, vol. 21, pp. 105–117, 1988.
[3] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, no. 220, p. 671, 1983.
[4] A. Corana, M. Marchesi, C. Martini, and S. Ridella, "Minimizing multimodal functions of continuous variables with the 'simulated annealing' algorithm," *ACM Trans. Math. Soft.*, no. 51, pp. 262–280, 1987.

[5] N. Baba, "A new approach for finding the global minimum of error function of neural networks," *Neural Networks*, vol. 2, pp. 367–373, 1989.
[6] S. Ridella, S. Rovetta, and R. Zunino, "Circular backpropagation networks for classification," *IEEE Trans. Neural Networks*, vol. 8, pp. 84–97, 1997.
[7] K. Lang and M. Witbrock, "Learning to tell two spirals apart," in *Proc. Connectionist Models Summer School*, 1989.
[8] S. Ridella, S. Rovetta, and R. Zunino, "Adaptive internal representation in circular backpropagation networks," *Neural Comput. Applicat.*, no. 3, pp. 222–233, 1995.
[9] R. P. Gorman and T. J. Sejnowski, "Analysis of hidden units in a layered network trained to classify sonar targets," *Neural Networks*, vol. 1, pp. 75–89, 1988.
[10] W. H. Wolberg and O. L. Mangasarian, "Multisurface method of pattern separation for medical diagnosis applied to breast cytology," in *Proc. Nat. Academy Sci.*, Dec. 1990, vol. 87, pp. 9193–9196.
[11] A. Guerin-Dugue and others, "Enhanced learning for evolutive neural architecture," ESPRIT-Basic Research Project Number 6891 (ROARS), Elena-NervesII, Tech. Rep., Deliverable R3-B4-P—Task B4: Benchmarks, June 1995.
[12] D. H. Deterding, "Speaker normalization for automatic speech recognition," Ph.D. dissertation, Univ. Cambridge, U.K., 1989.
[13] N. B. Karayannis and G. W. Mi, "Growing radial basis neural networks: Merging supervised and unsupervised learning with network growth techniques," *IEEE Trans. Neural Networks*, vol. 8, pp. 1492–1506, 1997.
[14] R. Hecht-Nielsen, *Neurocomputing*. Reading, MA: Addison-Wesley, 1990.
[15] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Pacific Grove, CA: Wadsworth and Brooks, 1984.
[16] M. Golea and M. Marchand, "A growth algorithm for neural-network decision trees," *Europhys. Lett.*, vol. 12, pp. 205–210, 1990.
[17] J. A. Sinat and J. P. Nadal, "Neural trees: A new tool for classification," *Network: Comput. Neural Syst.*, vol. 1, no. 4, pp. 423–438, 1990.
[18] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. XXVII, no. 3, pp. 379–423, pp. 624–656, July 1948.
[19] G. Mirchandani and W. Cao, "On hidden nodes for neural nets," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 661–664, 1989.
[20] J. Nadal, "Study of a growth algorithm for neural networks," *Int. J. Neural Syst.*, vol. 1, pp. 55–59, 1989.
[21] A. Roy, S. Govil, and R. Miranda, "An algorithm to generate radial basis function (rbf)-likenets for classification," *Neural Networks*, vol. 8, no. 2, pp. 179–202, 1995.
[22] S. Ridella, G. L. Speroni, P. Trebino, and R. Zunino, "Class-entropy minimization networks for domain analysis and rule extraction," *Neural Comput. Applicat.*, vol. 2, no. 1, pp. 40–52, 1994.
[23] M. Gori and F. Scarselli, "Are multilayer perceptrons adequate for pattern recognition and verification?," in *Neural Nets WIRN*, M. Marinaro and R. Tagliaferri, Eds. London, U.K.: Springer, 1996.
[24] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. New York: Wiley, 1991.
[25] S. I. Gallant, *Neural Network Learning and Expert Systems*. Cambridge, MA: MIT Press, 1993.
[26] S. A. J. Keibek, G. T. Borkema, H. M. A. Andree, M. H. F. Savenije, and A. Taal, "A fast partitioning algorithm and a comparison of binary feedforward neural networks," *Europhys. Lett.*, vol. 18, pp. 555–559, 1992.
[27] S. I. Gallant, "Perceptron-based learning algorithms," *IEEE Trans. Neural Networks*, vol. 1, pp. 179–191, June 1990.
[28] M. Mezard and J. P. Nadal, "Learning in feedforward neural networks: The tiling algorithm," *J. Phys. A: Meth. Gen.*, vol. 22, pp. 2191–2203, 1989.
[29] S.-C. Huang and Y.-F. Huang, "Bounds on the number of hidden neurons in multilayer perceptrons," *IEEE Trans. Neural Networks*, vol. 2, pp. 47–55, Jan. 1991.
[30] P. Rujan and M. Marchand, "Learning by minimizing resources in neural networks," *Complex Syst.*, vol. 3, pp. 229–242, 1989.
[31] M. Marchand, M. Golea, and P. Rujan, "Convergence theorem for sequential learning in two-layer perceptrons," *Europhys. Lett.*, vol. 11, pp. 487–492, 1990.
[32] L. Holmström and P. Koistinen, "Using additive noise in backpropagation training," *IEEE Trans. Neural Networks*, vol. 3, pp. 24–38, 1992.
[33] K. Matsuoka, "Noise injection into inputs in backpropagation learning," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, pp. 436–440, 1992.
[34] R. Setiono and H. Liu, "Neural-network feature selector," *IEEE Trans. Neural Networks*, vol. 8, pp. 654–662, 1997.
[35] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," *Advances in NIPS 2*. San Mateo, CA: Morgan Kaufman, 1990, pp. 524–532.
[36] M. Hoehfeld and S. E. Fahlman, "Learning with limited numerical precision using the cascade-correlation algorithm," *IEEE Trans. Neural Networks*, vol. 3, pp. 602–611, July 1992.
[37] The vowel database, available via anonymous ftp: ics.uci.edu/pub/machine-learning-databases

Sandro Ridella (M'93) received the Laurea degree in electronic engineering from the University of Genova, Italy, in 1966.

He is a full Professor in the Department of Biophysical and Electronic Engineering, University of Genova, Italy, where he teaches Circuits and Algorithms for Signal Processing. In the last seven years his scientific activity has been mainly focused in the field of neural networks.

Stefano Rovetta (M'98) received the Laurea degree in electronic engineering in 1993 and the Ph.D. degree in models, methods and tools for electronic and electromagnetic systems in 1997, both from the University of Genoa, Italy.

He is a Postdoctoral Researcher with the Electronic Systems and Networking Group of the Department of Biophysical and Electronic Engineering, University of Genoa. He is Invited Professor of Operating Systems at the University of Siena, and teaches Electronics at Genoa University. His research interests include electronic circuits and systems, and neural-network theory, implementation, and applications.

Rodolfo Zunino (S'90–M'90) received the Laurea degree in electronic engineering from Genoa University, Italy.

From 1986 to 1995 he was a Research Consultant with the Department of Biophysical and Electronic Engineering of Genoa University. He is currently with the same department as an Assistant Professor, where he teaches industrial electronics. His main scientific interests include electronic systems for neural networks, distributed control, and methods for data representation and processing.