GAL:

Networks that grow when they learn and shrink when they forget

Ethem Alpaydın

International Computer Science Institute

May 1991

TR 91-032

Abstract. Learning when limited to modification of some parameters has a limited scope; the capability to modify the system structure is also needed to get a wider range of the learnable. In the case of artificial neural networks, learning by iterative adjustment of synaptic weights can only succeed if the network designer predefines an appropriate network structure, i.e., number of hidden layers, units, and the size and shape of their receptive and projective fields. This paper advocates the view that the network structure should not, as usually done, be determined by trial-and-error but should be computed by the learning algorithm. Incremental learning algorithms can modify the network structure by addition and/or removal of units and/or links. A survey of current connectionist literature is given on this line of thought. "Grow and Learn" (GAL) is a new algorithm that learns an association at one-shot due to being incremental and using a local representation. During the so-called "sleep" phase, units that were previously stored but which are no longer necessary due to recent modifications are removed to minimize network complexity. The incrementally constructed network can later be finetuned off-line to improve performance. Another method proposed that greatly increases recognition accuracy is to train a number of networks and vote over their responses. The algorithm and its variants are tested on recognition of handwritten numerals and seem promising especially in terms of learning speed. This makes the algorithm attractive for on-line learning tasks, e.g., in robotics. The biological plausibility of incremental learning is also discussed briefly.

Keywords. Incremental learning, supervised learning, classification, pruning, destructive methods, growth, constructive methods, nearest neighbor.

Earlier part of this work was realized at the Laboratoire de Microinformatique of Ecole Polytechnique Fédérale de Lausanne and was supported by the Fonds National Suisse de la Recherche Scientifique. Later part was realized at and supported by the International Computer Science Institute. A number of people helped by guiding, stimulating discussions or questions: Subutai Ahmad, Peter Clarke, Jerry Feldman, Christian Jutten, Pierre Marchal, Jean Daniel Nicoud, Steve Omohondro and Leon Personnaz.

For contact, write to E. Alpaydın, ICSI, 1947 Center Street Suite 600, Berkeley CA 94704-1105 USA. Email: ethem@icsi.berkeley.edu

1. INTRODUCTION

1.1. Assessing the quality of a neural network solution

There are three factors that affect the quality of a neural network solution:

- *Success achieved on test data* indicates how well the network generalizes to data unseen during training which one wants to maximize. This generally is taken as the only performance criterion.
- *Network complexity* by itself can be very difficult to assess but two important factors are *network size* and *processing complexity of each unit.* Network size gives the memory required which is the product of the number of connections and the number of bits required to store each connection weight. Processing complexity depends on how costly it is to implement processing occurring in each unit, e.g., sigmoid vs. threshold non-linearity, fan-in, fan-out properties, precision in storage and computation, etc. This has a negative effect on the quality as one prefers smaller and cheaper networks.
- *Learning time* is the time required to learn the given training data till one gets a reasonable amount of performance. This is to be minimized also.

In the ideal case, learning algorithms where a certain cost function is minimized should take into account not only success but the whole quality measure including success, network complexity, and learning time. However the actual relative importances of these three factors depend on the application and the implementation constraints. In tasks like optical character recognition where the environment does not change and thus learning is done only once, learning time is not a critical factor. On the other hand, when a hardware implementation is envisaged, network complexity is important and a smaller but less successful network can be preferred over a more complex but very successful one. In tasks like robotics where rapid adaptation to the environment is necessary, learning time has crucial importance. The best neural network for a given application is one having the highest quality and thus it does not make sense to say that one algorithm is better than another one per se; only based on a certain application and a set of implementation constraints can solutions be compared among themselves. This implies that with different hardware and environmental constraints, for the same training set, different networks may be required. The learning system may have a repertoire of learning algorithms and depending on the current constraints, one is chosen and employed. For example, when rapid adaptation is necessary, a one-shot learning method may be used to quickly learn encountered associations. When the system later has time to spare, an iterative fine-tuning process may be employed to improve performance.

1.2. Why smaller and simpler is better

In the case of feed-forward layered networks, the mapping capability of a network depends on its structure, i.e., the number of layers, and the number of hidden units (Lippman, 1987; Hanson & Burr, 1990; Hertz et al., 1991). Given a certain application and training data, the network structure should be pre-determined as algorithms like the back-propagation (Rumelhart et al., 1986) can modify only the synaptic weights but not the net structure.

Networks with more layers and hidden units can perform more complicated mappings however better performance on unseen data, i.e., generalization ability, implies lower order mappings. Given a certain training set, there are very many possible generalizations and one is interested in the simplest possible generalization. One reason for this is that simpler explanations of a phenomenon, i.e., those that require a shorter description, are more plausible and have a higher probability of occurrence (Rissanen, 1987). By having a smaller network, one also decreases the network size and thus less memory is required to store the connection weights, and the computational cost of each iteration decreases. However note that although one iteration takes less in a smaller network, the number of iterations to learn a certain training set can be more.

Frequently an analogy is made between learning and curve fitting (Duda & Hart, 1973). There are two problems in curve fitting: finding out the *order* of the polynomial and finding out the *coefficients* of the polynomial once the order is determined. For example given a certain data set, one first decides on that the curve is second order thus has the form $f(x) = ax^2 + bx + c$ and then computes somehow values of a, b, and c, e.g., to minimize sum of squared differences between required and predicted $f(x_i)$ for x_i in the training set. Once the coefficients are computed, $f(x_i)$ value can be computed for any x_i even for x_i that are not in the training set. Orders smaller than the good one risk not to lead to good approximations even for points in the data set. On the other hand, choosing a larger order implies fitting a high order polynomial to low order data and although one hopes that the high order terms will have zero coefficients to have their effect cancelled, this practically is not the case; it leads to perfect fit to points in the data set but very bad $f(x_i)$ values may be computed for x_i not in the training data, i.e., the system will not generalize well.

Similarly a network having a structure simpler than necessary cannot give good approximations even to patterns in the training set and a structure more complicated structure than necessary, i.e., with many hidden units, "overfits" in that it leads to nice fit to patterns in the training set performing poorly on patterns unseen. Bigger networks also need larger data samples for training; it was pointed out (Müller & Reinhardt, 1990) based on an information theoretic measure that the required number of patterns in the training set grows almost linearly with the number of hidden units.

As currently there is no formal way by which the network structure can be computed given a certain training set or application, the usual approach is trial-and-error, i.e., a series of attempts are made each one involving deciding on a more complicated network structure and iterating the learning algorithm a considerable number of times until one is content with the performance, which can be assessed by cross-validation. In determining the structure, the network designer is only guided by his/her intuition and rather limited knowledge of the application and the learning algorithm. Any knowledge related to the problem concerning the geometry or the topology of the input should be introduced to the network as help (Denker et al., 1987). When the input is an image for example, most of the constraints are local, i.e., nearby pixels have correlated output, thus it makes more sense to define local receptive fields than completely connected layers (Le Cun et al., 1989). A recent approach is to use a genetic algorithm to be able to "produce" better structures (Harp et al., 1990). The problem however is that "parent" networks should be trained for their fitness to be assessed and in tasks where training set is large or many generations are necessary, this turns out to be not very practical.

1.3. One-shot on-line learning

The time it takes to learn a given training set is crucial in many applications. Iterative algorithms based on gradient descent require very many iterations to converge and thus one is compelled to learn *off-line*. Another reason for off-line learning besides learning time is that, network models in which associations are distributed over a set of connections need to be introduced patterns in an unbiased fashion which cannot be guaranteed in a real world operational environment. One cannot for example add a certain association to network's memory by training with one pattern only; as weights are distributed, the whole training set should be re-learned together with the new pattern. However in an *on-line* learning system, one does not have time to do this and neither there is memory to store the whole training set. This is the case in many robotics applications where rapid adaptation to environment is a must. Iterative algorithms or networks using a distributed representation thus cannot learn at *one-shot* on-line. This fact led to the belief that neural network models cannot learn one-shot on-line and this became a frequent point on which learning limits of neural models are negatively judged (McCarthy, 1990; Leveit, 1990). To be able to learn on-line, addition of a new association should be done very quickly, i.e., one-shot, and without affecting the past existing knowledge of the network for other inputs. GAL algorithm explained in this paper using a local representation and based on an incremental approach has both of these properties and is a connectionist method that learns at one-shot.

2. INCREMENTAL LEARNING

The idea of incremental learning implies starting from the simplest possible network and adding units and/or connections whenever necessary to decrease error (Alpaydın, 1990a). To be able to decrease network size and increase generalization ability, one also wants to be able to get rid of units/connections whose absence will not degrade significantly system's performance. In both cases, as opposed to a static network structure, small modifications to a dynamic network structure during learning is envisaged. Determination of the network structure and computation of connection weights are not done separately but together, both by the learning algorithm.

Approaches given in the connectionist literature leading to network structure modification can be divided into two classes. There are those that start with a big network and eliminate the unnecessary and there are others that start from small and add whatever is necessary (Fig. 1).[1]

[1] *Note that there are also incremental* **unsupervised** *learning algorithms like ART (Carpenter & Grossberg, 1987) and GAR (Alpaydın, 1990a) which are beyond the scope of this paper. In unsupervised incremental learning, one adds a new cluster index whenever the current input is not similar to any of the existing clusters. The similarity measure is thus done in the input space regardless of the class to which the input patterns belong.*

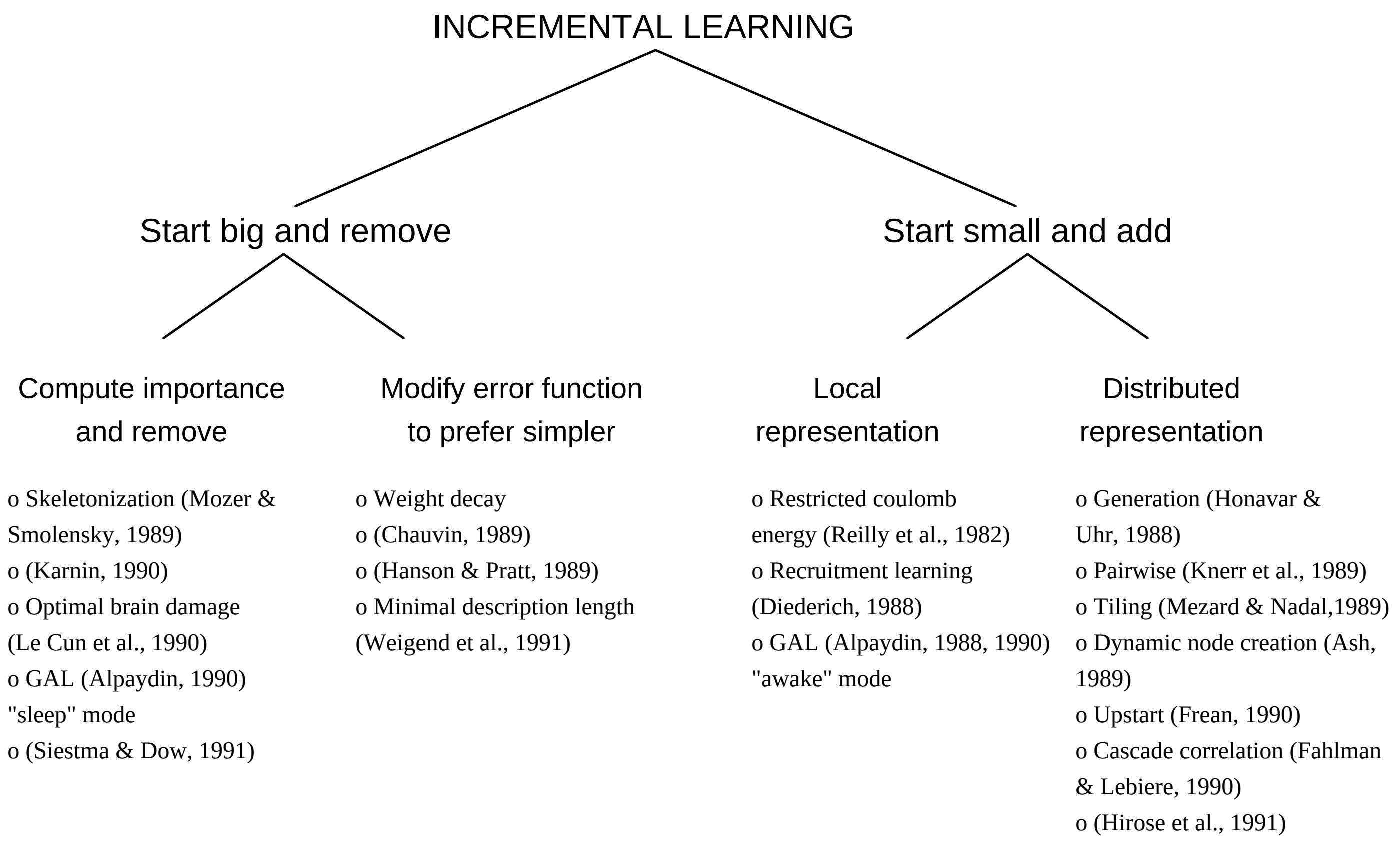
INCREMENTAL LEARNING
Start big and remove
Start small and add
Compute importance and remove
Modify error function to prefer simpler
Local representation
Distributed representation
o Skeletonization (Mozer & Smolensky, 1989)
o (Karnin, 1990)
o Optimal brain damage (Le Cun et al., 1990)
o GAL (Alpaydin, 1990) "sleep" mode
o (Siestma & Dow, 1991)
o Weight decay
o (Chauvin, 1989)
o (Hanson & Pratt, 1989)
o Minimal description length (Weigend et al., 1991)
o Restricted coulomb energy (Reilly et al., 1982)
o Recruitment learning (Diederich, 1988)
o GAL (Alpaydin, 1988, 1990) "awake" mode
o Generation (Honavar & Uhr, 1988)
o Pairwise (Knerr et al., 1989)
o Tiling (Mezard & Nadal,1989)
o Dynamic node creation (Ash, 1989)
o Upstart (Frean, 1990)
o Cascade correlation (Fahlman & Lebiere, 1990)
o (Hirose et al., 1991)

Figure 1. Taxonomy of incremental learning.

2.1. Start big and remove

In the context of polynomial curve fitting the "start big and remove" approach implies starting from a high order polynomial and eliminating those high order terms which do not contribute significantly to success. Such methods are also called *pruning* or *destructive*. If one starts with a large network and if the problem in fact requires a simpler network, one likes to have the weights of all unnecessary connections and the output of all unnecessary units equal to zero. There are two approaches in achieving this:

[1] One may explicitly try to compute how important is the existence of a connection/unit in keeping the error low after the network has been trained and a number of the least important may then be deleted. The remaining network needs to continue to be trained. In the ideal case, understanding the importance of a connection/unit requires training two networks one with the connection/unit and one without. As this is not practical for large networks, heuristical approaches have been proposed with the back-propagation algorithm where the sensitivity of the error function to the elimination of a connection/unit is estimated.

- In the "skeletonization" procedure (Mozer & Smolensky, 1989), the network is trained till a certain performance criterion is met. The "relevance" of each connection is then computed which is given as the partial derivative of error with respect to the connection. However this value tends to zero when error decreases thus a poor relevance is computed when error is low. Using a linear error function for computation of relevances, i.e., the sum of the absolute value of the differences of required and actual values, leads to better relevance values.

- (Karnin, 1990) computes the "sensitivity" in the same way but sums the values computed throughout learning instead of computing only once at the end. More memory and computation is required but the usual quadratic error measure can be used.
- "Optimal brain damage" (Le Cun et al., 1990) uses an information theoretic measure to compute the "saliency" of a connection using the second derivative of the error function. Training proceeds till error reaches down to a certain value at which point saliencies are computed and a number of the least salient are deleted and the remaining network is re-trained.
- Grow and Learn (GAL) algorithm, as explained in the next section, has a "sleep" mode during which the network is closed to the environment, the inputs are generated by the system itself, and units that are no longer necessary due to recent additions are removed.
- Siestma and Dow (1991) examine the behavior of units under the presentation of the entire training data and decide to prune accordingly. From "broad" networks with few layers and many units on each layer, after training, they trim as many units as possible and by adding extra layers, generate "long narrow" networks with many layers but few units on each layer; they discover however that networks of the latter type generalize poorly.

[2] Instead of approximating how much the error will change if the unit/connection is eliminated, one may also modify the learning algorithm so that after training, the unnecessary connections/units will have zero weight/output.

- One may build a tendency in the learning algorithm to have those weights that are not relevant decay to zero by decrementing them by a certain factor at each weight update (see review in Hertz et al., 1991). Weights that are necessary to store associations will be moved away from zero but those that are not needed will not be increased and will finally be close to zero.
- This decay can be done also implicitly by modifying the error function. Terms can be added to the error function to penalize large weights (Chauvin, 1989) and hidden units that have small outputs (Hanson & Pratt, 1989).
- Another possibility is to use the information theoretic idea of "minimum description length" and add a term to the cost function that penalizes network complexity, i.e., number of connections (Weigend et al., 1991). Thus during gradient descent, the algorithm will settle to the network that has the best trade-off between error and complexity. Such a cost function is similar to the quality measure proposed in the first section; however the network complexity is defined very simply as the number of connections.

2.2. Start small and add

The other approach in dynamic modification of network structure during learning, which can be named "start small and add," implies starting from a simple network and adding units and/or connections to decrease error. These methods are also called *growth* or *constructive*. In the context of curve fitting, it implies starting with a low order polynomial and adding higher order terms whenever the polynomial of current order cannot give a good fit for any set of coefficients. Note that this cannot be done in a straightforward manner especially in networks where associations are distributed over a number of shared connections; the whole training should be re-done in such a case. One needs a certain mechanism whereby addition of a new unit improves success instead of corrupting the harmony as one would normally expect. There are two possibilities:

[1] If one can make sure that when the new unit gets activated, none of the ancient units get activated, there will be no problem. The units should thus somehow be able to suppress other units when they get control. This implies a competitive strategy and a local representation.

- The first incremental neural learning algorithm is the Restricted Coulomb Energy (RCE) model (Reilly et al., 1982) which is an incremental version of Parzen windows. Associated with each unit is a number of prototypes where a prototype gets activated only if the input falls into its domination region, determined by a distance computation followed by a thresholding. If an input does not activate any prototype, a new prototype unit is created at that position with an initially large domination region. Prototypes that get activated for inputs that belong to different classes are penalized by having their regions decreased which is done by modifying the threshold. The input space is thus divided into zones dominated by prototype units. A number of sweeps is necessary to finetune the thresholds where units closer to class boundaries have small zones and units interior have larger domination zones.

- Recruitment learning (Diederich, 1988) is used in the case of structured connectionist networks where a previously free unit is committed to represent a new concept and required connections built up dynamically (Feldman, 1982). This is a one-shot learning algorithm, i.e., one iteration is sufficient to learn a new concept.
- In the first version of Grow-and-Learn (Alpaydın, 1988), weights in a single layer were learned by Hebbian learning at one shot. However if an association could not be learned or if addition of this association corrupted the previously learned associations, a new hidden unit was added with input weights equal to the input vector. The output weight was computed in such a manner to compensate for the effect of the input layer and thus impose any output. The problem was that as Hebbian learning was used, orthogonality of input patterns were necessary and as this is rarely the case, many units were allocated. However Hebbian learning made the algorithm a one-shot learning one.
- The current version of Grow-and-Learn (GAL) algorithm (Alpaydın, 1990a, 1990b), explained in the next section, uses also a local representation by having a number of exemplars associated with each class. It learns at one-shot but orthogonality of patterns is no longer required.

[2] Another possibility is to divide the network into separately trained subnetworks where such subnetworks can be added in an incremental manner. One approach is to have subnets that have competition between subnets, another is to have each subnet as another hidden layer.

- The "generation" method proposed by Honavar and Uhr (1988) enables a "recognition cone" to modify its own topology by growing links and recruiting units whenever performance ceases to improve during learning by weight adjustment using back-propagation.
- The "stepwise procedure" uses subnets of different conceptual interpretations (Knerr et al., 1989). In this method, one first trains a one layer network with the Perceptron learning algorithm assuming that classes are linearly separable. For a class where this is not satisfied, one adds a subnet to separate classes in a pairwise manner. For cases where this does not work either, one performs a piecewise approximation of boundaries using logical functions by additional subnets. As linear separability is rarely the case, one generally is obliged to separate classes in a pairwise manner two by two. The major drawback of this is that the number of hidden units increase exponentially with the number of class units.
- Another approach named the "tiling" algorithm adds a new hidden layer whenever the required mapping cannot be done with the existing network (Mézard & Nadal, 1989; explained also in Hertz et al., 1991). There is a "master" unit which is trained to be the output unit by the pocket algorithm—a variant of the Perceptron learning algorithm. If this unit cannot learn all the required associations, additional "ancillary" units are added to learn the rest and another layer is created with a master unit and learning proceeds till the master unit can learn to behave like the output unit.
- The "dynamic node creation" method (Ash, 1989; explained also in Müller & Reinhardt, 1990) trains networks with one hidden layer only. Given a certain net that is being trained, if the rate of decrease of error falls down a certain value, a new hidden unit is added and training is resumed when all connections are continued to be modified.
- The "upstart" algorithm (Frean, 1990) uses binary units. Like the "tiling" algorithm, first one unit is trained to learn the required associations using the pocket algorithm. If this is not successful, "daughter" units are created to correct the output of this "parent" unit, for "wrongly on" and "wrongly off" cases. This is repeated in a recursive manner to lead to a binary tree which can then be "squashed" into one hidden layer.
- In the "cascade correlation" algorithm (Fahlman & Lebiere, 1990), if the required mapping cannot be learned by one layer, a hidden unit is added and trained while the previously trained

weights are "frozen." If this does not work either, another hidden unit is added as another hidden layer and so on. A hidden layer has only one hidden unit but connections skip layers, i.e., a unit has connections to all the following layers.

- Method proposed by (Hirose et al., 1991) is quite similar to that proposed by Ash (1989), namely, using a network with only one hidden layer, if the rate of decrease for error becomes small, additional hidden units are added. Their contribution is that, once the network converges, the most recently added hidden unit is removed and the network is checked to determine whether the same function can be achieved by fewer hidden units. If the network cannot converge when a hidden unit is removed, the last network that converged is chosen as the final network.

3. GROW AND LEARN (GAL)

3.1. GAL network structure

This section explains the "Grow and Learn" (GAL) algorithm used to learn categories in an incremental manner (Alpaydın, 1990a, 1990b). Class definitions are extended if need arises. The network grows when it learns class definitions, thus the name.

A GAL network has the structure seen in Fig. 2. The first layer is the layer of input units. The second layer is that of exemplars (prototypes) and the third is the layer of class units. The weight vector of unit e is denoted as W_e. T_{ec} denotes the connection from exemplar e to class c. When P is the input vector, the activation of an exemplar unit e, A_e, is the distance between P and the weight vector of e, W_e, computed using a suitable metric denoted $D()$, e.g., Euclidean distance. It is assumed here that $D()$ is normalized such that $D(A, A)$ is zero and $D(A, B)$ increases as A and B gets further apart. A winner-take-all type non-linearity then chooses the closest, i.e., minimum. The class units' activations are simply computed by a dot product. T_{ec} values are 1 or 0 depending on whether e is an exemplar of class c or not.

$$\forall e,\ A_e = D(P, W_e).$$

$$E_e = \begin{cases} 1, & \text{if } A_e = \min_i(A_i); \\ 0, & \text{otherwise.} \end{cases}$$

$$T_{ec} = \begin{cases} 1, & \text{if } e \text{ is an exemplar of class } c; \\ 0, & \text{otherwise.} \end{cases}$$

$$C_c = \sum_e E_e * T_{ec}. \tag{1}$$

This structure is what one would normally have to implement nearest-neighbor using a neural network formalism. The input space is divided in the form of a Voronoi tesselation where exemplar units' domination regions are bounded by hyperplanes that pass through the medians of the two closest exemplar units. The domination region of a class is the juxtaposition of its exemplars' domination regions. Class boundaries are thus approximated in a piecewise manner with no restriction on shapes, e.g., linear separability, convexity.

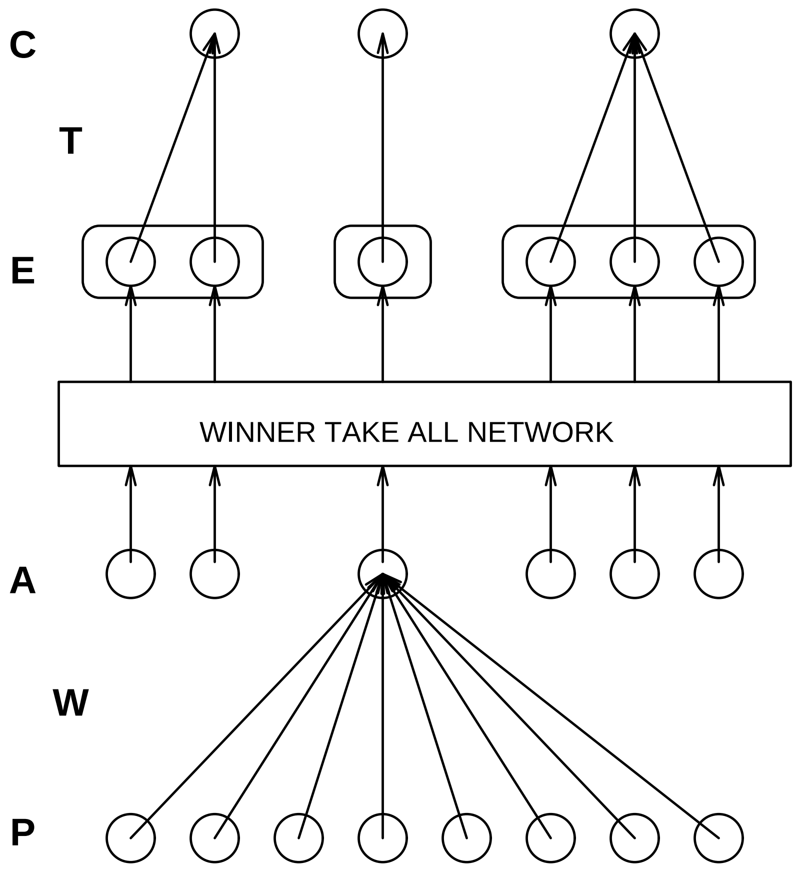

Figure 2. GAL network structure. P is the input vector. W_e is the weight vector of exemplar unit e. A_e is equal to the distance between input vector P and weight vector of exemplar e. Only one of E_i is active, namely that whose weight vector is closest to the input vector which in turn activates the corresponding class unit. C is the layer of class units.

3.2. Learning in GAL

What makes GAL incremental is that to start with, there are neither exemplar nor class units. They are added and W and T values are set as follows: When a new input P is given during training as being a member of class c, the response of the network is computed as given in equation (1) above and checked if it is already correct. If it is, no modification is done. If it is not, an exemplar unit is created and the weight vector of the newly created exemplar unit is set equal to the input vector for it to be the closest exemplar if that input is encountered once more. The exemplar unit is then connected to the correct class unit. Of course a class unit needs also be created if that class is encountered for the first time.

$$W_e = P.$$
$$\forall o,\ T_{eo} = \begin{cases} 1, & \text{if } o = c; \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

3.3. Example

As a didactic example to show how GAL works, a two dimensional hypothetical signal is chosen. The complete test data and 10% of the test data chosen at random taken as the training data is given in Fig. 3. Textures denote classes and blank points are not associated with any class. Euclidean distance is used as the metric.

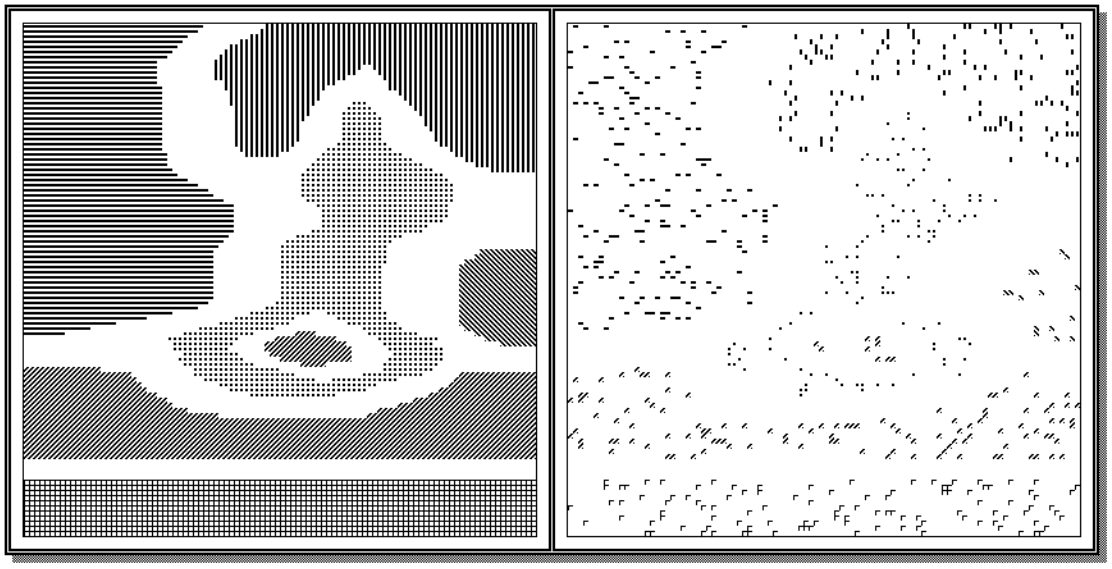

Figure 3. To the left, complete test data and to the right, 10% of the test data that make up the training data are shown. Different textures denote different classes and blanks are points that are not associated with any class.

The evolution of the response of GAL network after having seen 10, 50, 100, and 200% (2 sweeps) of the training set is given in figures 4 to 7.

The number of exemplars stored and classification error as a function of the learning iterations are given in figures 8 and 9 respectively. The states shown in figures 4 to 7 are marked on charts with letters A to D. In the beginning of training, many erroneous classifications are made and exemplars are added to correct. As more and more iterations are made, network's response tend to get right more frequently and thus less additions are needed. Two or three sweeps are generally sufficient to get 100% success on the training data.

3.4. Two spirals problem

The two spirals problem originally proposed by Alexis Wieland of MITRE Corp. is difficult unless attempted by an incremental learning algorithm that can also modify the network structure (Fahlman & Lebiere, 1990). GAL can learn this problem in two sweeps over the training set which make only 384 iterations ! The problem and GAL's response is shown in Fig. 10. Out of 192 patterns in the training set, 79 are stored as exemplars.

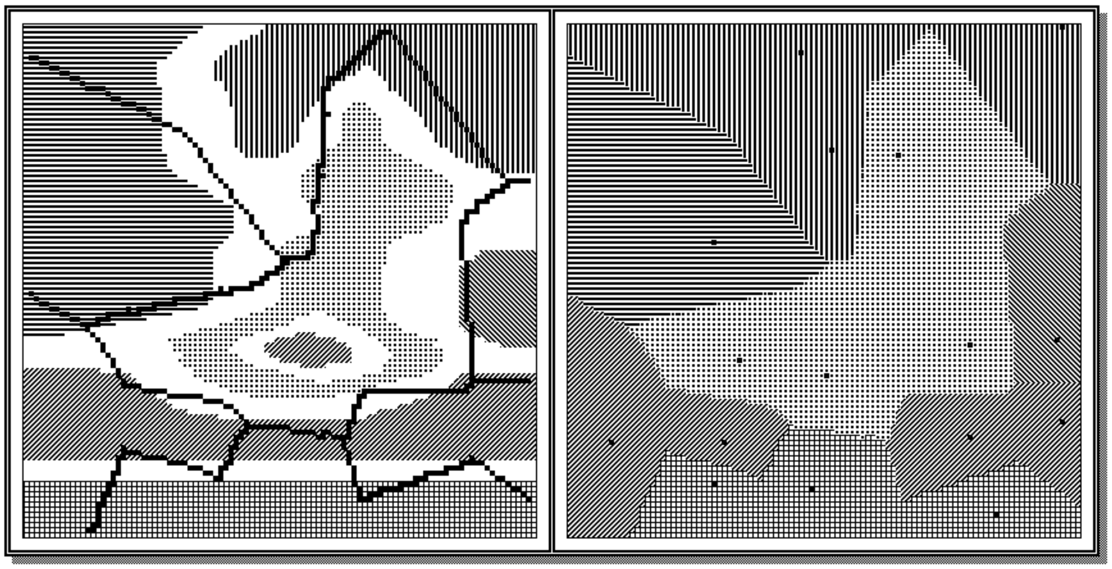

Figure 4. After having seen 10% of the training data, 16 units are stored and error on test data is 1234 out of 7074.

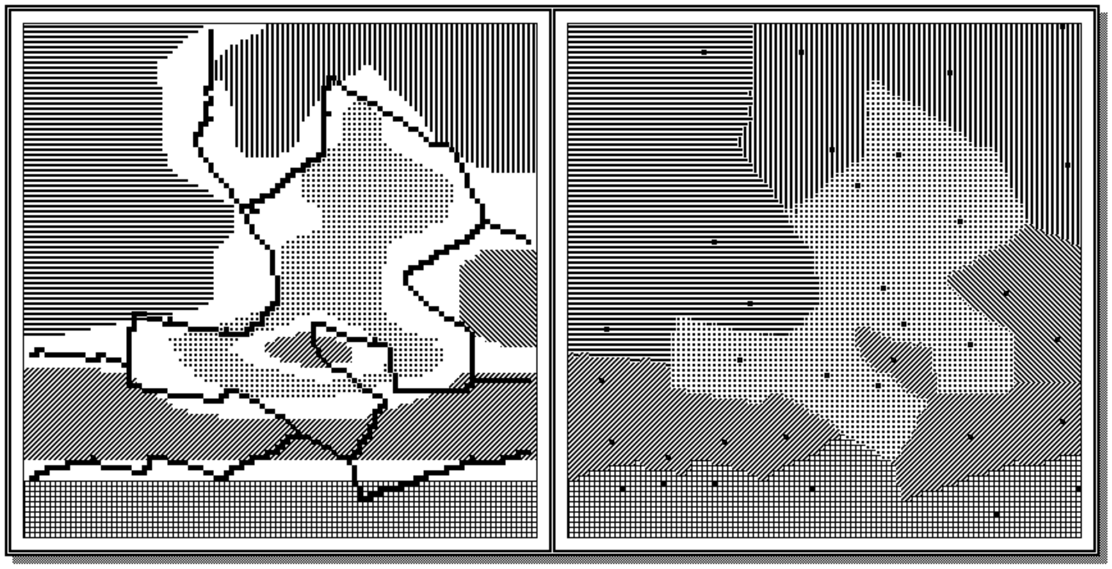

Figure 5. After 50%, 34 units are stored and error is 310.

3.5. Forgetting in GAL

The algorithm tends to store those input patterns in the training set that are closest to boundaries for finer approximation of the class boundaries. Unfortunately in a learning scheme as GAL's, the actual patterns stored as exemplars and the number of them, depend on the order in which patterns are encountered during learning. An exemplar previously stored, when another closer to a boundary is added, becomes useless (Fig. 11). Such exemplar units, as now they are in the domination region

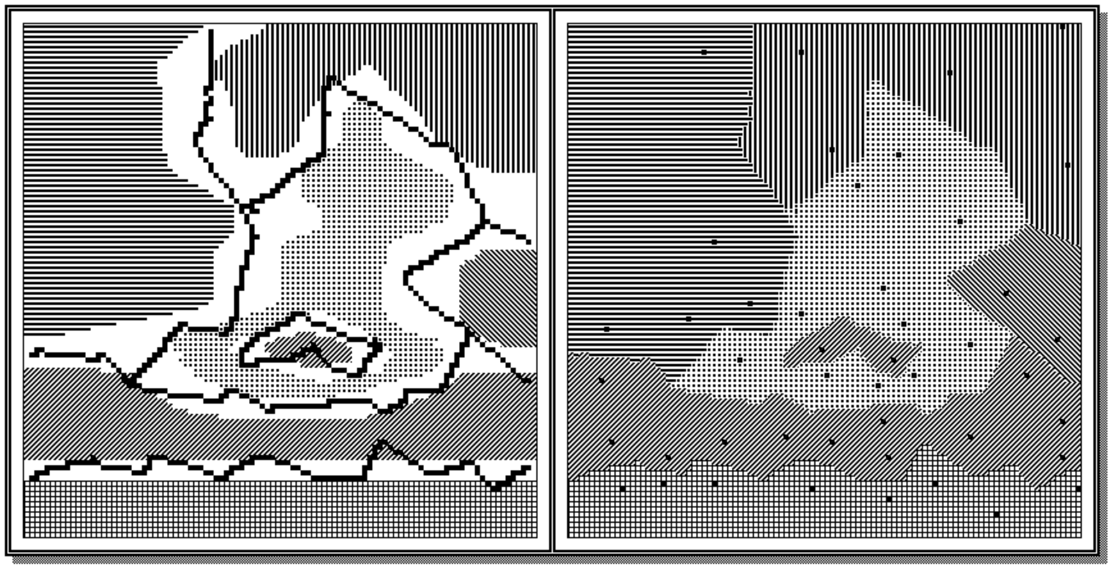

Figure 6. After 100% (one sweep), 45 units are stored and error is 67.

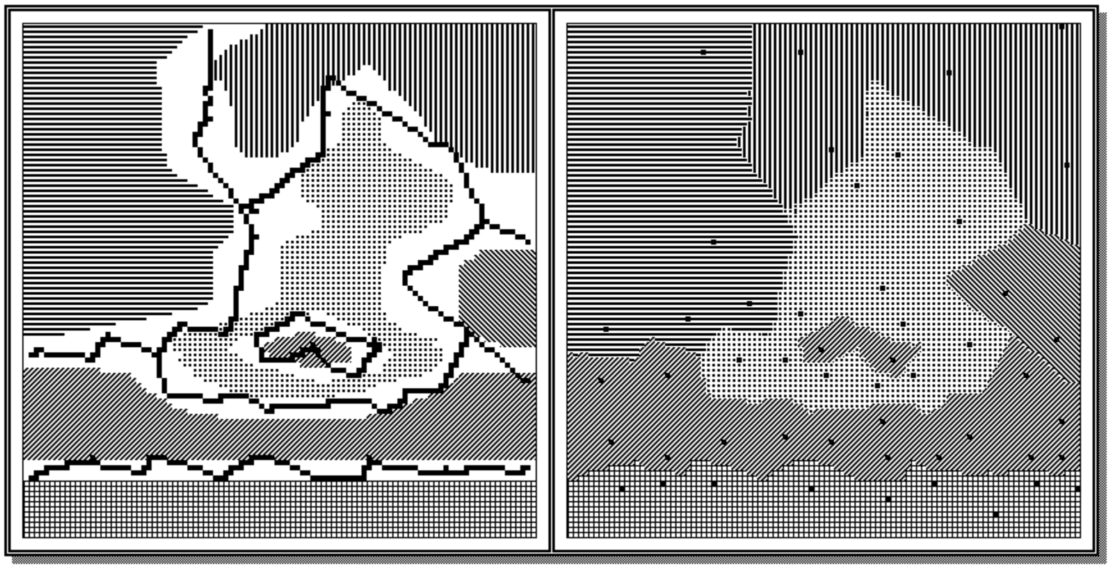

Figure 7. After 200% (two sweeps), 50 units are stored and error on test data is 37. Error on training data is 0 so no more learning iterations are necessary.

of another unit of the same class, can be thought of as useless and may be eliminated to decrease network complexity.

To get rid of such units, a phase called *sleep* is introduced where the following operation is done. One of the exemplar units is chosen at random. The input vector is set equal to the weight vector of the chosen exemplar—the "dream"— and the response of the network is computed by equation (1). The response is the class to which that exemplar is connected. Then that particular exemplar is disabled and the response is computed once more, this time without that unit. If the classes found

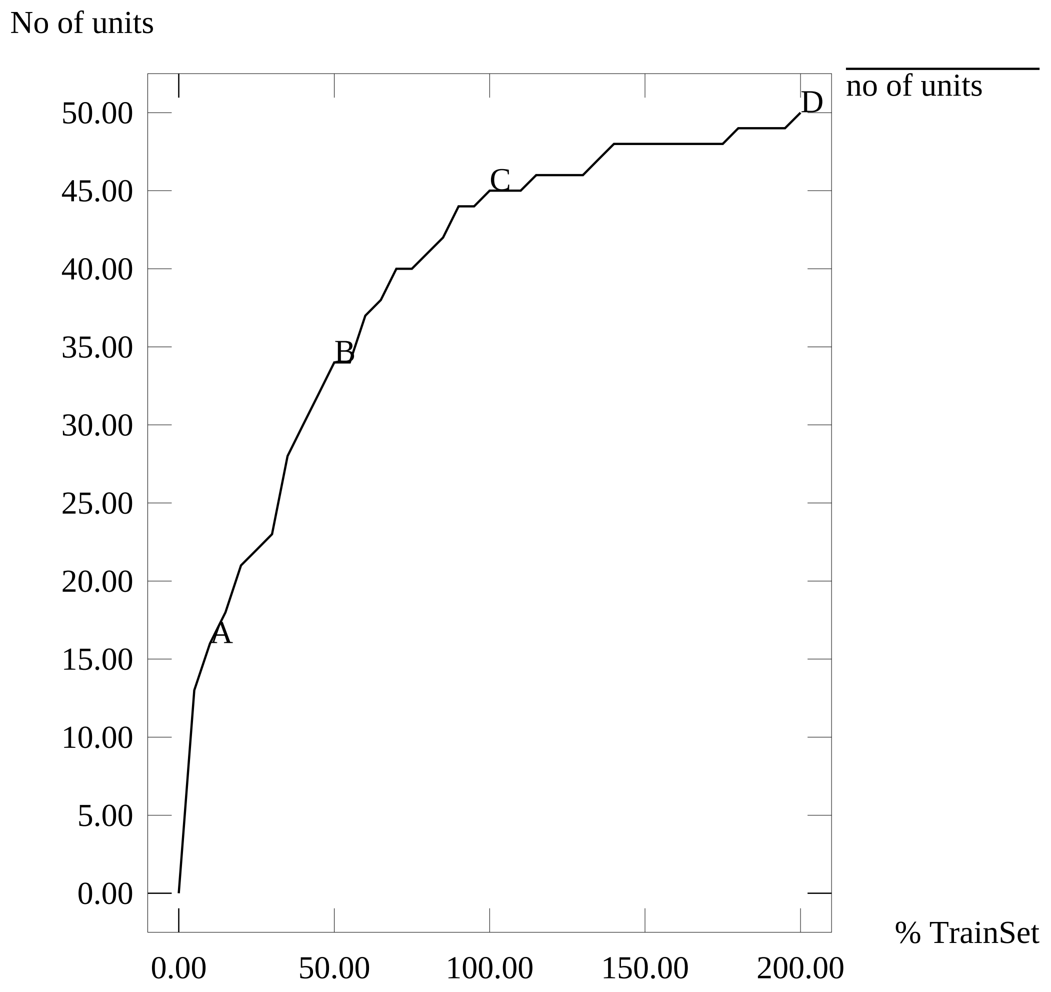

Figure 8. Number of exemplar units as a function of learning iterations. Letters A to D mark states shown in figures 4 to 7.

in the two cases are the same then that exemplar is removed. Note that such a strategy may cause error to increase as we check for the exemplar unit only, not by all the points that it dominates.

When the network sleeps after its state in figure 7, 20 units out of 50 are removed but error increases from 37 to 337 (Fig. 12). To remedy increasing error during sleep, one uses alternating "awake" and "sleep" phases. A number of "awake" passes are made during which units are added and error decreased, which are followed by a "sleep" phase that gets rid of some redundant exemplars. After a few such alternations, the GAL network settles down to a set of units where no longer additions are necessary as one has already 100% on the training data and where all units are closest to boundaries thus no removals are possible either. Of course success on test data depends on how well the training data reflects it. One should also note that there may be several subsets of the training data that lead to 100% on the training data and different success values on the test data. It is for this reason, the same training data ordered differently may lead to different GAL networks. In section 3.8, we will see how we can take advantage of this.

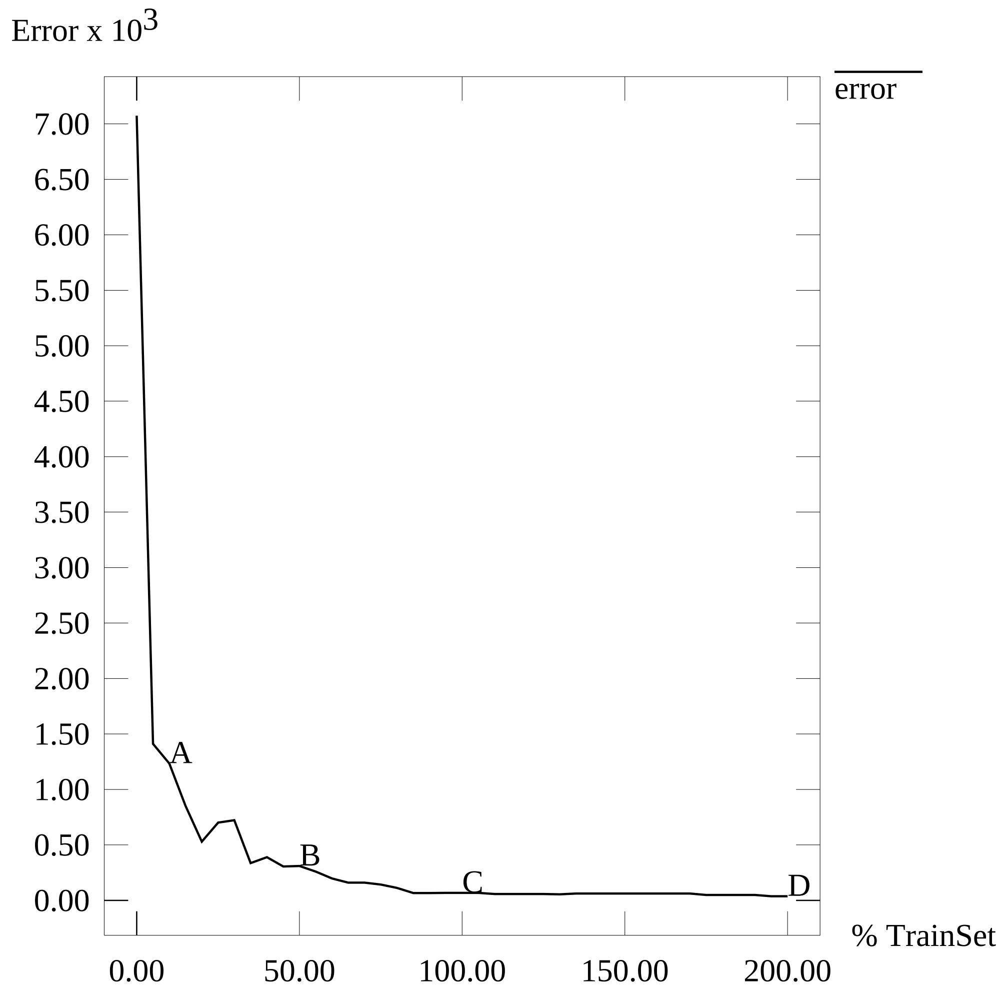

Figure 9. Error as a function of learning iterations. Letters A to D mark states shown in figures 4 to 7.

3.6. GAL with reject

GAL as explained hitherto, always gives a class code as response and never rejects. When the risk of misclassification is high, one needs a mechanism to be able to reject when the network is not "sure." The most straightforward way is to check the closest and next closest exemplars (named e and f respectively):

$$A_e = \min_i (A_i)$$

$$A_f = \min_{i \neq e} (A_i)$$

and reject if they belong to different classes, and if there is no great difference between their activations:

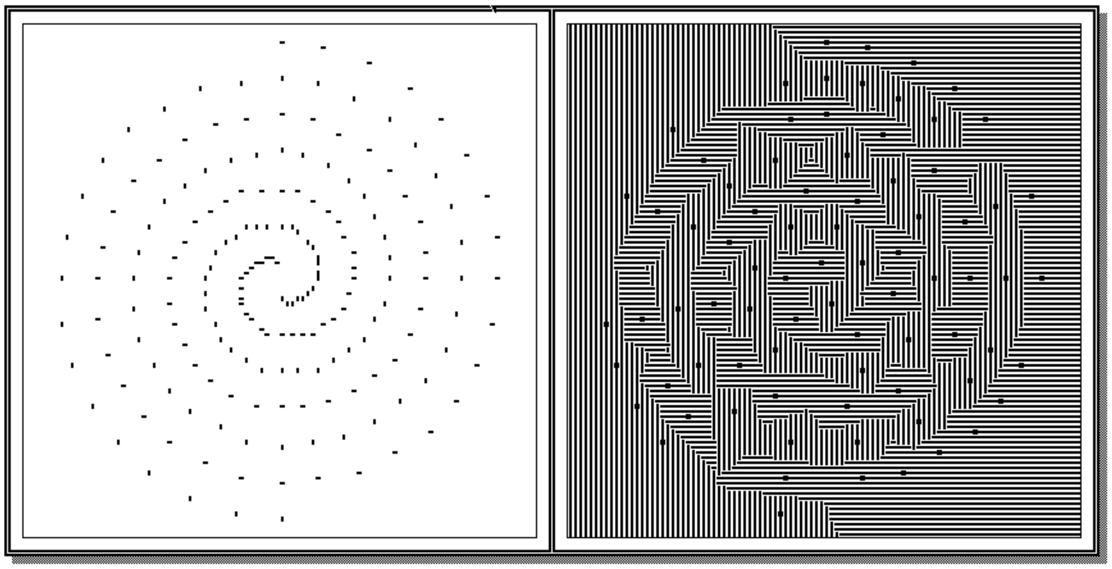

Figure 10. *Training points for the two spirals problem and GAL's output after having learned the problem in 384 iterations (2 sweeps) with 100% success storing 79 patterns out of 192. Horizontal and vertical lines denote points that are members of the two classes.*

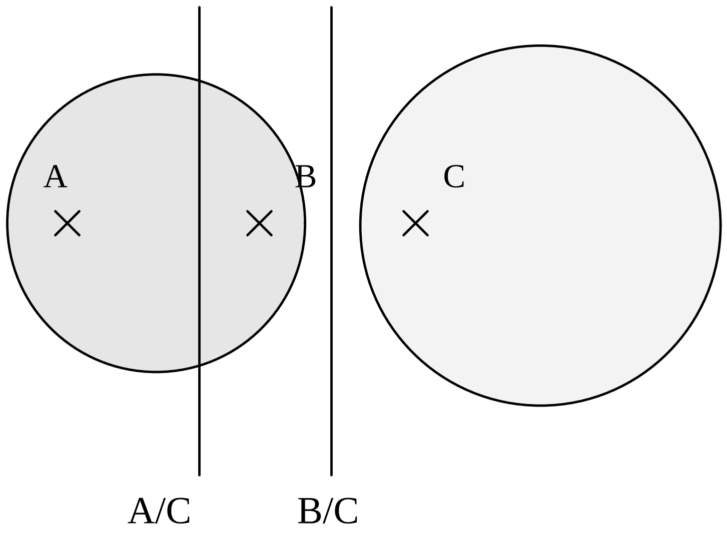

Figure 11. *A and B are associated with one class and C with another. If A is seen before B, both A and B are stored; if B is seen first, A is not stored.*

$$class(e) \neq class(f)\ AND\ A_f - A_e < \theta$$

When the two closest exemplars belong to the same class, the network is rather sure, otherwise certainty decreases as input gets closer to their median. The user defined threshold θ defines the size of this reject region around the median.

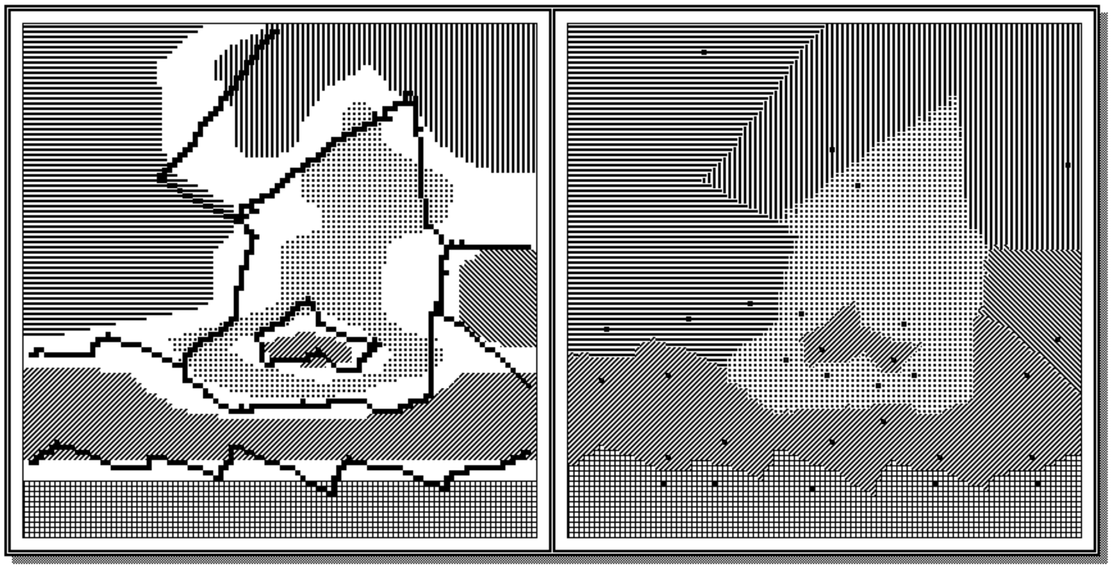

Figure 12. Following figure 7, a "sleep" phase takes place and 20 exemplars out of 50 are removed with a parallel increase of error from 37 to 337.

3.7. Finetuning a GAL network

In the first section it was mentioned that a learning system needs to have a set of learning algorithms and, depending on the current constraints, the most appropriate one may be chosen and used. One constraint may be the learning time. The time it takes to learn a given training set hardly ever has been taken into account as a quality measure in "neural" applications. The reason for this may be that there were simply not algorithms around that could learn very fast, i.e., in the rate of actual signal from the environment. Thus learning has been done off-line from a previously collected sample. In real working environment, e.g., in closed-loop systems in robotics, the system may need to learn and act accordingly very quickly. Under such circumstances, a fast algorithm is necessary even if it may not learn optimally. The system's knowledge may then later be re-organized when there is enough time. Thus a two level learning system may be appropriate where the first learns very fast but not optimally, and a second that "finetunes" the network to improve performance but on a much slower basis.

The "sleep" mode of GAL is one such attempt to improve system quality using an off-line process by eliminating unnecessary units. Hopfield et al. (1983) learned random vectors using a Hopfield network with a negative factor to increase the energy of such states which may be parasitic confabulation stable states. A similar idea was also used with the Boltzmann machine by Hinton and Sejnowski (1986) which they call the $phase^{-}$. These schemes are on the line of the proposal of Crick and Mitchison (1983) related to the function of "REM sleep" which will be discussed in the section on the biological view of incremental learning.

In GAL the only way to modify the network is by adding a new exemplar and once an exemplar is added, the only way it is modified afterwards is by being removed during "sleep" if necessary. Other than those, the connection weights are not modified. The advantages are:

[1] Learning is very fast as network modification is very simple.
[2] The precision required to store the connection weight values need not be more precise than that is required to store the input pattern.[2]

[2] *For example, when input vectors are binary, binary weights and Hamming distance measure may be*

The disadvantages are:

[1] The method is not immune to noise. If there is considerable amount of noise in input patterns or if there is teacher noise, this causes a big degradation in the response of the network.

[2] Even if there is no noise, the piecewise boundaries found by GAL may be rather different from boundaries in the Bayesian sense as densities are not taken into account.

To alleviate these problems, an optional "finetuning" process may be employed with a GAL network to have an averaging process near boundaries, to be able to have medians nearer to Bayes optimal boundaries. It works as follows. When the closest exemplar and the next closest exemplar belong to different classes, the weight vector of the closest exemplar e is moved a little bit towards the input vector.

$$W_e = W_e + \alpha(t) * (P - W_e). \tag{3}$$

Note that because this is a statistical averaging process, a large number of iterations will be necessary starting from a small $\alpha(0)$ and decreasing it very slowly towards zero as t, training iteration, increases.

When learning time and network complexity is critical but the performance is not, GAL in its basic form can be used. When there is time and machinery available to perform more precise computation, GAL network can be finetuned off-line.

3.8. Multiple GAL networks

Another possible way to improve the success is to use a number of GAL networks and "vote" over their responses. It was mentioned before that the same training set ordered in different ways may lead to GAL networks with different patterns stored as exemplars. Although all give 100% on the training set, they may lead to different success percentages with the test set. Instead of using one GAL network, one may use a number of these and decide based on their responses:

[1] One possibility is to count how many of the networks give a certain class code as output and set response r equal to the class having the maximum count.

$$Net_{pc} = \begin{cases} 1, & \text{if the } p^{th} \text{ net gives class } c \text{ as output;} \\ 0, & \text{otherwise.} \end{cases}$$

$$\forall c,\ R_c = \sum_p Net_{pc}.$$

$$R_r = \max_c (R_c). \tag{4}$$

[2] One may use a more sophisticated voting scheme by weighting the response of each network by its "certainty" that its response is correct, computed in the same way as in the case of GAL with reject. If the two closest exemplars belong to the same class, the network is quite certain; if they belong to different classes, the certainty decreases as the input gets closer to their median. When e is the closest and f is the next closest, "certainty" of net p, C_p, and the response is computed as follows:

$$C_p = \begin{cases} A_e, & \text{if class(e)=class(f);} \\ A_f - A_e, & \text{otherwise.} \end{cases}$$

$$\forall c,\ R_c = \sum_p C_p * Net_{pc}.$$

used as shown in section 4 where GAL is used for recognition of handwritten numerals.

$$R_r = \max_c (R_c) \,. \tag{5}$$

This scheme is a faster alternative to "finetuning" in improving success and may even require less memory and simpler implementation.

4. RECOGNITION OF HANDWRITTEN NUMERALS

To test and assess utility of GAL in a real-world application, it is applied to the task of recognizing handwritten numerals. The database is made up of 1200 examples of 10 digits written by 12 people where each example is a bitmap normalized to the size of 16 by 16 pixels (Guyon et al., 1989). This is a relatively easy database as writers all followed a given writing style. It is divided into two parts. One half is used to train the network and the second to test how well the network generalizes. No preprocessing is done. The distance measure is the Euclidean distance — equivalent to Hamming distance when patterns are binary. Table 13 summarizes the results obtained as an average of 30 runs by variants of GAL, the nearest neighbor method, restricted coulomb energy (RCE) model and the learning vector quantization (LVQ) (Kohonen, 1988). Note that LVQ and GAL finetuned require more than one bit for each connection. The reason why the number of units do not increase as rapidly as expected when one uses multiple GAL nets is that, the same vector generally needs to act as an exemplar in different GAL nets, but instead of storing it many times, one can just store it once performing the distance computation only once and feed the output A_i to different winner-take-all nets.

	no of	Average			
Network type	sweeps	No.units	Success	Error	Reject
Nearest neighbor	1	600	93.2	6.8	0.0
RCE	6	142.0	74.1	2.9	23.0
LVQ	50	100.0	92.9	7.2	0.0
GAL	3	118.4	90.2	9.8	0.0
GAL with reject	4	148.2	86.3	2.5	11.2
GAL finetuned	50	118.4	91.4	8.2	0.0
3 GAL nets S	3	247.5	92.3	7.7	0.0
5 GAL nets S	3	321.8	93.3	6.7	0.0
3 GAL nets W	3	247.5	93.4	6.6	0.0
5 GAL nets W	3	321.8	94.2	5.8	0.0
7 GAL nets W	3	371.7	94.6	5.4	0.0

Table 13. Comparison of different algorithms for recognition of handwritten numerals. In the case of multiple GAL nets, 'S' and 'W' denote simple and weighted voting schemes respectively.

In Fig. 14, results obtained in different simulations for various GAL variants are given for the 30 runs made. Note that when one uses multiple GAL nets, a finer pointed distribution with less variance and a higher success mean is achieved which is an indication of better generalization. Note that despite the fact that the dimensionality of the input is very high, i.e., 256, GAL still performs rather well. It is also rather interesting to note that by using multiple GAL nets, one can have a success higher than that is achieved by the nearest neighbor method although only a *part* of the training set is stored !

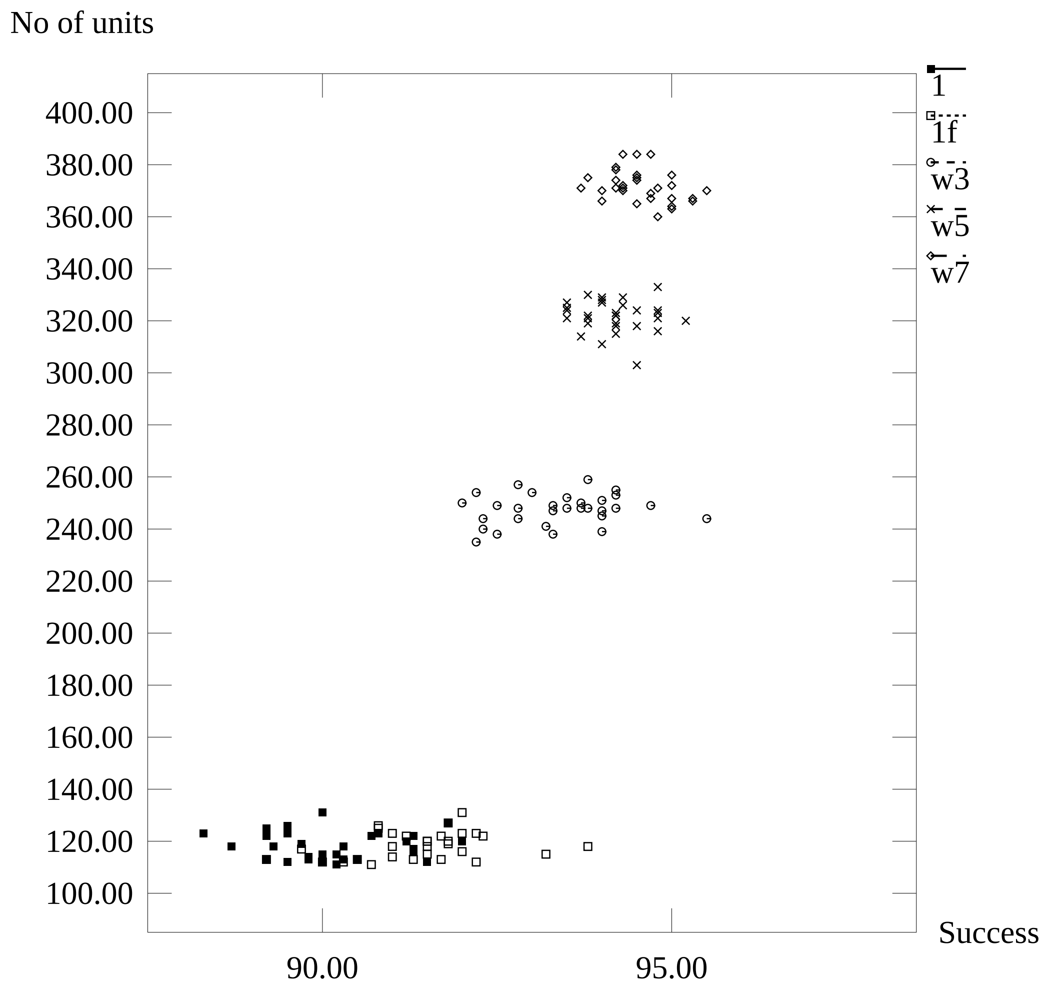

Figure 13. Distribution of simulation results for different GAL network types. (1) basic GAL network, (1f) GAL finetuned, (wn) n *GAL nets with weighted voting.*

5. CRITIQUE OF GAL

GAL basically is a variant of the nearest neighbor method where instead of storing all the patterns in the training set, one stores only a subset. Successive "awake" and "sleep" phases allows the system to choose a good subset, namely those patterns that are closest to class boundaries to be able to approximate these boundaries in a piecewise manner. One should note that as generally training sets tend to be big and highly redundant, the advantage of GAL in terms of minimizing memory and computational requirements cannot be underestimated. The problem of "curse of dimensionality" in the case of methods based on nearest neighbor also applies to GAL, namely, the percentage of the training set that needs to be stored by GAL increases as the dimensionality of the input increases. Results achieved in section 4 however shows that despite a very high dimensional input, one still can achieve a good performance by GAL.

One problem is that the final set of exemplars stored depends on the order of patterns in the training set. For different orderings, although all give 100%, the number of exemplars stored and the success on the test set may be different. One may use cross-validation to choose one of the GAL nets, namely that which generalizes best to the unseen part of the training data. A GAL net may be learned and then "finetuned" taking into account class distributions thus averaging over vectors in the training set. Or a number of GAL nets may be employed with the result computed by voting over them, thus averaging over the responses of a number of different GAL nets.

GAL is very simple to understand and use; there are no parameters for which optimal values need be searched.[3] No a priori knowledge of the task is required to tune network parameters simply because there is nothing that can be tuned.

GAL does not extract any features; there are no hidden units trained to extract features common to many classes. One just assigns patches of input space to classes. There is no limitation on the shape of classes that can be learned, however if class boundaries are low order, GAL may not do a good job trying to piecewise approximate with a lot of small hyperplanes. Although it is for categorization, GAL may also be extended to learn any mapping when the range is discretized and each discrete part (values within a certain user defined tolerance value) is taken as a different class, i.e., piecewise constant approximation. It leads to very quick learning of a mapping. Using smaller tolerance values lead to finer mappings at the expense of more units. Omohondro (1989) proposed independently the same method to quickly learn function mappings. Moody and Darken (1990) pointed out the advantages of local receptive fields in terms of decreasing learning time.

Learning incrementally by adding exemplar units is a very fast method. Actually GAL learns at one shot and thus can learn effectively on-line. Relatively low success achieved can be improved in a number of ways:

- The connection weights can be "finetuned," this makes the network also immune to noise.
- A number of networks can be trained and response computed by voting over their responses. In this way as shown in section 4, one may get a success higher than that of nearest neighbor by actually storing only a subset of the training set.
- An application dependent preprocessing technique may be employed before input patterns are fed to GAL.
- The GAL network may be preceded by an unsupervised learning method to extract statistically important characteristics of the input signal. The Grow and Represent (GAR) algorithm (Alpaydın, 1990a) is an incremental unsupervised learning algorithm which has recently been proposed for this purpose. Reduction of dimensionality by feature extraction also alleviates the problem of "curse of dimensionality."

[3] *Except the threshold value in the case of GAL with reject that needs to be tuned to get the required reject rate that guarantees a certain maximum error percentage.*

6. BIOLOGICAL VIEW

There is no evidence that neurons are generated post-natally, so if there is any form of dynamic allocation of previously unused units, it should have the form of recruitment learning. Instead of dynamic addition of units, one would expect recruitment of a previously unused unit and growth of required synapses dynamically for the necessary task. The brain is built according to some genetic program with an abundant number of cells. The neurons are generated before birth followed by a a migration process where young neurons migrate from one part of the brain to another. Finally, they settle down, maturate, specialize, and form synapses (Cowan, 1979) (Kandel & Schwartz, 1985).

This initial redundant structure loses then between 15% and 85% of its components. This phenomenon of "cell death" takes place both during embryonic and post-natal days. Cell death before birth is an intrinsic phenomenon and the criteria are genetic. Post-natal cell death depends on experience where the structure during a *critical development period* is tested against the environment and "sculptured" (Cowan, 1979) to better match the environment.

Before cell death starts, the majority of the axons have reached their target fields and have just started establishing connections. The fact that these two phenemona overlap, suggests that there is some sort of a feedback process "back-propagated" from the axons to the soma—a retrograde transport of a "trophic," i.e., nourishing, substance which probably is glia-derived when the axons are growing and driven by the activity of the target cells once the synapses are formed (Clarke, 1985).

In the case of post-natal development, the utility criterion by which relevance of neurones is assessed is related to the functional activity of the cells in the target field on which synapses were formed—retrograde maintenance modulated by activity (Clarke, 1985). If the target field is destroyed, the cell death increases to around 100% and if it is artificially extended, death proportion decreases (Cowan et al., 1984). Although the dendritic branching of a neuron is determined genetically, most neurons seem to generate many more synapses than are needed or than they are subsequently able to maintain. It was proposed that it is not the actitivity level per se but the correlation of the activity of the pre- and post-synaptic cells that leads to synaptic stabilization (Schmidt & Tieman, 1989). There is a phase of synapse elimination during which many (and in some cases all but one) of the initial group of connections are withdrawn (Cowan, 1979). The relationship between activity dependent synapse elimination and cell death is not clearly known but it was proposed (Schmidt & Tieman, 1989) that synapse elimination might occur first and lead to cell death when the number of synapses falls below a critical number.

It was initially proposed by Crick and Mitchison (1983) that some sort of a "reverse learning" to get rid of parasitic memory traces occurs during Rapid Eye Movement (REM) sleep when dreaming occurs. The idea basically is that, the system is closed to its environment, e.g., sleep, inputs are generated by the system itself, e.g., dreaming, and unwanted modes of behaviour emerging due to accumulation of experience are eliminated during an active process of unlearning. It is said that "we dream in order to forget." Jouvet (1983; Kandel & Schwartz, 1985) suggested that "species-specific behaviours are rehearsed during sleep governed by a genetical preprogram." It is like simulating real life during sleep; the aim is rather similar to having military manoeuvres which can also be thought of as a sort of off-line learning. The idea in such proposals is similar to what is proposed in the first section, namely, having a two level learning system where the first level strategy can learn very quickly on-line but probably not optimally, and a second level strategy that is employed whenever there is time available to improve the memory in an off-line manner by re-structuring the previously acquired knowledge.

REFERENCES

[1] Alpaydın, E. (1988) "Grow and Learn" *Internal Note*, Laμi–EPF Lausanne, Switzerland.

[2] Alpaydın, E. (1990a) *Neural models of incremental supervised and unsupervised learning*, PhD dissertation, Ecole Polytechnique Fédérale de Lausanne, Switzerland.

[3] Alpaydın, E. (1990b) "Grow and Learn: An incremental method for category learning" *Int. Neural Network Conf.*, Paris, France.

[4] Ash, T. (1989) "Dynamic node creation in backpropagation networks," *Connection Science*, **1**, 365–375.

[5] Carpenter, G.A., Grossberg, S. (1987) "ART2: Self-organization of stable category recognition codes for analog input patterns," *Applied Optics*, **26**, 4919–4930.

[6] Chauvin, Y. (1989) "A back-propagation algorithm with optimal use of hidden units," in *Advances in neural information processing systems*, D.S. Touretzky (ed.), **1**, 519–526, Morgan Kaufman.

[7] Clarke, P.G.H. (1985) "Neuronal death in the development of the vertebrate nervous system," *Trends in Neuroscience*, **8**, 345–349.

[8] Cowan, W.M. (1979) "The development of the brain," *Scientific American*, **241(3)**, 106–117.

[9] Cowan, W.M., Fawcett, J.W., O'Leary, D.D.M., Stanfield, B.B. (1984) "Regressive events in neurogenesis," *Science*, **225**, 1258–1265.

[10] Crick, F., Mitchison, G. (1983) "The function of dream sleep," *Nature*, **304**, 111–114.

[11] Denker, J., Schwartz, D., Wittner, B., Solla, S., Howard, R., Jackel, L., Hopfield, J. (1987) "Large automatic learning, rule extraction, and generalization," *Complex Systems*, **1**, 877–922.

[12] Diederich, J. (1988) "Connectionist recruitment learning" *Proc. of the 8th European conf. on Artificial Intelligence*, London, UK.

[13] Duda, R.O., Hart, P.E. (1973) *Pattern classification and scene analysis*, John Wiley and sons.

[14] Fahlman, S.E., Lebiere, C. (1990) "The cascade-correlation architecture," in *Advances in neural information processing systems*, D.S. Touretzky (ed.), **2**, 524–532, Morgan Kaufman.

[15] Feldman, J. (1982) "Dynamic connections in neural networks," *Biological Cybernetics*, **46**, 27–39.

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

[17] Guyon, I., Poujoud, I., Personnaz, L., Dreyfus, G., Denker, J., le Cun, Y. (1989) "Comparing different neural network architectures for classifying handwritten digits" *Int. Joint Conf. on Neural Networks*, Washington, USA.

[18] Hanson, S.J., Pratt, L.Y. (1989) "Comparing biases for minimal network construction with back-propagation," in *Advances in neural information processing systems*, D.S. Touretzky (ed.), **1**, 177–185, Morgan Kaufmann.

[19] Hanson, S.J., Burr, D.J. (1990) "What connectionist models learn: Learning and representation in connectionist networks," *Behavioral and Brain Sciences*, **13**, 471–518.

[20] Harp, S.A., Samad, T., Guha, A. (1990) "Designing application-specific neural networks using the genetic algorithm," in *Advances in neural information processing systems*, D.S. Touretzky (ed.), **2**, Morgan Kaufmann, 447–454.

[21] Hertz, J., Krogh, A., Palmer, R.G. (1991) *Introduction to the theory of neural computation*, Addison Wesley.

[22] Hinton, G.E., Sejnowski, T.J. (1986) "Learning and relearning in Boltzmann machines," in *Parallel distributed processing*, D.E. Rumelhart, J.L. McClelland (eds.), **1**, MIT Press, 282–317.

[23] Hirose, Y., Yamashita, K., Hijiya, S. (1991) "Back-propagation algorithm which varies the number of hidden units," *Neural Networks*, **4**, 61–66.

[24] Honavar, V., Uhr, L. (1988) "A network of neuron-like units that learns to perceive by generation as well as reweighting of its links" *Proc. of the 1988 Connectionist Summer School*, D. Touretzky, G. Hinton, T. Sejnowski (eds.), Morgan Kaufman.

[25] Hopfield, J.J., Feinstein, D.I., Palmer, R.G. (1983) "'Unlearning' has a stabilizing effect in collective memories," *Nature*, **304**, 158–159.

[26] Jouvet, M. (1983) "Neurophysiology of dreaming," in *Functions of the nervous system*, M. Monnier, M. Meulders (eds.), **4**, *Psycho–Neurobiology*, Elsevier, 227–248.

[27] Kandel, E.R., Schwartz, J.H. (1985) *Principles of neural science*, 2nd edition, Elsevier.

[28] Karnin, E.D. (1990) "A simple procedure for pruning back-propagation trained neural networks," *IEEE trans. on neural networks*, **1**, 239–242.

[29] Knerr, S., Personnaz, L., Dreyfus, G. (1989) "Single layer learning revisited: A stepwise procedure for building and training a neural network," in *Neurocomputing: Algorithms, architectures, and applications*, F. Fogelman-Soulié, J. Hérault (eds.), NATO ASI Series, Springer, in print.

[30] Kohonen, T. (1988) *Self organization and associative memory*, 2nd edition, Springer.

[31] Le Cun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jeckel, L.D. (1989) "Backpropagation applied to handwritten zip recognition," *Neural Computation*, **1**, 541–551.

[32] Le Cun, Y., Denker, J.S., Solla, S.A. (1990) "Optimal brain damage," in *Advances in neural information processing systems*, D.S. Touretzky (ed.), **2**, Morgan Kaufman, 598–605.

[33] Leveit, W.J.M. (1990) "On learnability, empirical foundations, and naturalness," *Behavioral and Brain Sciences*, **13**, 501.

[34] Lippman, R.P. (1987) "An introduction to computing with neural nets," *IEEE ASSP magazine*, **4**, 4–22.

[35] Mézard, M., Nadal, J.-P. (1989) "Learning in feedforward layered networks: The tiling algorithm," *Journal of Physics A*, **22**, 2191–2204.

[36] McCarthy, J. (1990) "Interview: Approaches to artificial intelligence," *IEEE Expert*, **5(3)**, 87–89.

[37] Mozer, M.C., Smolensky, P. (1989) "Skeletonization: A technique for trimming the fat from a network via relevance assessment," *Connection Science*, **1**, 3–26.

[38] Müller, B., Reinhardt, J. (1990) *Neural networks: An introduction*, Springer Verlag.

[39] Omohondro, S. (1989) *Geometric learning algorithms*, ICSI Technical Report **89-041**.

[40] Reilly, D.L., Cooper, L.N., Elbaum, C. (1982) "A neural model for category learning," *Biological Cybernetics*, **45**, 35–41.

[41] Rissanen, J. (1987) "Stochastic complexity," *Journal of Royal Statistical Society B*, **q**, 49.223–239 and 252–265

[42] Rumelhart, D.E., Hinton, G.E., Williams, R.J. (1986) "Learning internal representations by error propagation," in *Parallel distributed processing*, D.E. Rumelhart, J.L. McClelland (eds.), MIT Press, 151–193.

[43] Schmidt, J., Tieman, S.B. (1989) "Activity, growth cones and the selectivity of visual connections," *Comments on Developmental Neurobiology*, **1**, 11–28.

[44] Siestma, J., Dow, R.J.F. (1991) "Creating artificial neural networks that generalize," *Neural Networks*, **4**, 67–79.

[45] Weigend, A.S., Rumelhart, D.E., Huberman, B.A. (1991) "Generalization by weight-elimination with application to forecasting," in *Advances in neural information processing systems*, R.P. Lippman, J. Moody, D.S. Touretzky (eds.), Morgan Kaufmann, in print.