Artificial neural networks that grow when they learn and shrink when they forget
Öğrenince büyüyen, unutunca küçülen yapay sinir ağları

Ethem Alpaydın

Bilgisayar Mühendisliği Bölümü, Boğaziçi Üniversitesi, TR-80815 Istanbul
alpaydin@trboun.bitnet

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. The reader is referred to (Alpaydın, 1991) for the author's own contribution to the field.

Özet

Eğer öğrenme sadece bazı parametrelerin değerlerini değiştirebilmek ise kullanım alanı kısıtlı kalır; öğrenilebileceklerin olabildiği kadar geniş olması için sistemin yapısının da değiştirilebilir olması gerekir. Yapay sinir ağlarında sadece bağlantı ağırlıklarının ayarlanması ile gerçekleşen öğrenmenin başarılı olabilmesi ancak ağ yapısının, yani saklı katman ve saklı ünite sayıları ve bunların önceki ve sonraki katmanlara bağlantı şeklinin, kullanım alanına uygun olması ile gerçekleşebilir. Bu tebliğ, ağ yapısının belirlenmesinin sıkça yapıldığı gibi deneme-yanılma ile değilde, yine öğrenme yordamının kendisi tarafından yapılması gerektiğini savunur. Bu tip yordamlarda ağ yapısı, örneğin hata geri yayma yordamında olduğu gibi durağan değildir; gerektiğinde yeni ünite ve bağlantılar eklenebilir veya olanlar çıkarılabilir. Bu tebliğ, bu konuda yapılmış çalışmaların bir özetini verimektedir; yazarın bu alana katkısı için, okuyucu (Alpaydın, 1991)'e başvurmaya davet edilir.

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:

[1] *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.

[2] *Network complexity* by itself can be very difficult to assess but two important factors are the *network size* and the *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.

[3] *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 (Alpaydın, 1991) 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 +

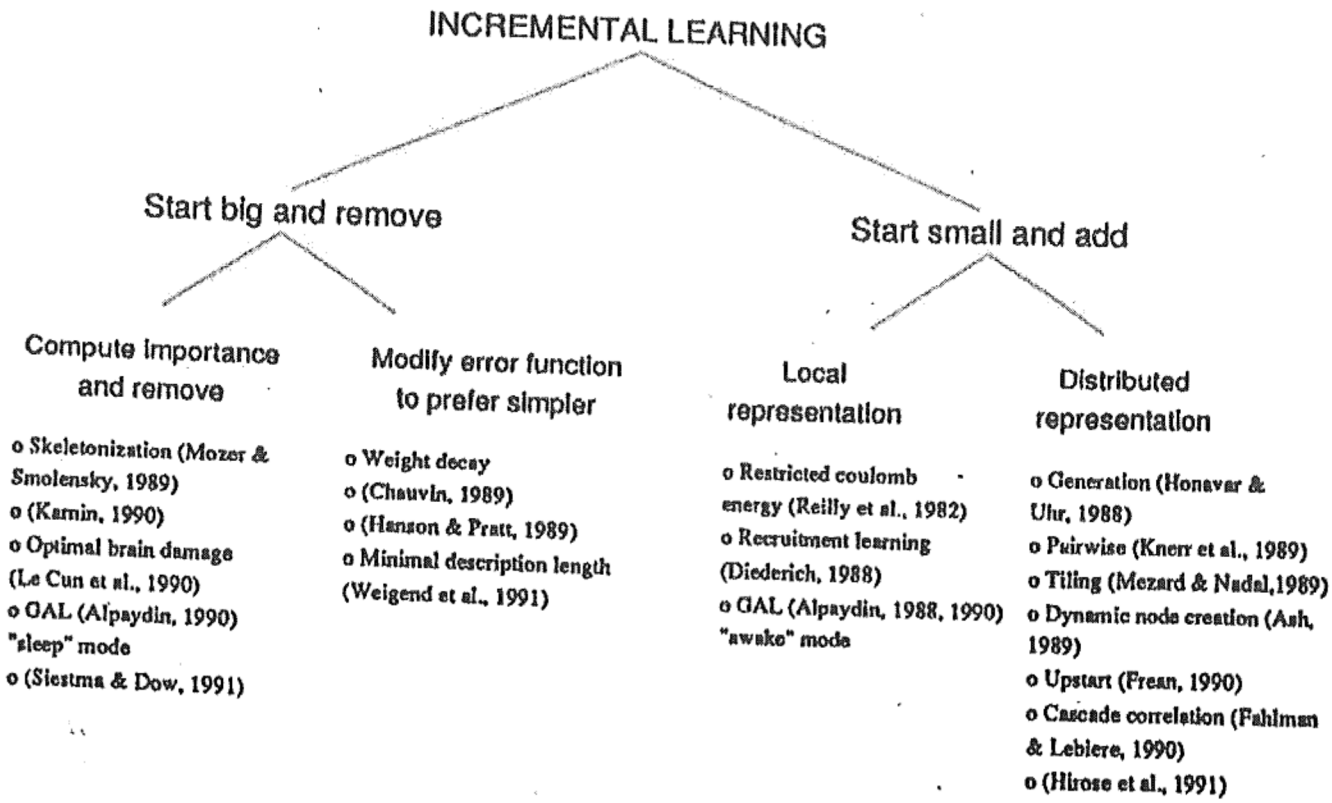

Fig. 1. Taxonomy of incremental learning.

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

---

+ 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.

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 (Alpaydın, 1990a), 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), 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 (Mezard & 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.

REFERENCES

[1] Alpaydın, E. (1988) "Grow and Learn" *Internal Note, Lami--EPFL,* Switzerland.

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

[3] Alpaydın, E. (1991) "GAL: Networks that grow when they learn and shrink when they forget" *International Computer Science Institute,* TR 91-032, Berkeley, CA.

[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, *NIPS,* D.S. Touretzky (ed.), **1**, 519--526, Morgan Kaufmann

[7] 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

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

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

[10] Fahlman, S.E., Lebiere, C. (1990) "The cascade-correlation architecture, *NIPS,* D.S. Touretzky (ed.), **2**, 524--532, Morgan Kaufmann

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

[12] Frean, M. (1990) "The upstart algorithm: A method for constructing and training

feedforward neural networks" *Neural Computation*, 2, 198--209
[13] Hanson, S.J., Pratt, L.Y. (1989) "Comparing biases for minimal network construction with back-propagation," *NIPS*, D.S. Touretzky (ed.), 1, 177--185, Morgan Kaufmann
[14] Hanson, S.J., Burr, D.J.(1990) "What connectionist models learn: Learning and representation in connectionist networks" *Behavioral and Brain Sciences*, 13, 471--518
[15] Harp, S.A., Samad, T., Guha, A.(1990) "Designing application-specific neural networks using the genetic algorithm" *NIPS*, D.S. Touretzky (ed.), 2, 447--454, Morgan Kaufmann
[16] Hertz, J., Krogh, A., Palmer, R.G. (1991) *Introduction to the theory of neural computation*, Addison Wesley
[17] Hirose, Y., Yamashita, K., Hijiya, S. (1991) "Back-propagation algorithm which varies the number of hidden units" *Neural Networks*, 4, 61--66
[18] 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 Schod*, D. Touretzky, G. Hinton, T. Sejnowski (eds.), Morgan Kaufmann
[19] Karnin, E.D. (1990) "A simple procedure for pruning back-propagation trained neural networks" *IEEE trans. on neural networks*, 1, 239--242
[20] Knerr, S., Personnaz, L., Dreyfus, G. (1989) "Single layer learning revisited: A stepwise procedure for building and training a neural network" *Neurocomputing: Algorithms, architectures, and applications*, F. Fogelman-Soulie, J. Herault (eds.), NATO ASI Series, Springer
[21] 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
[22] Le Cun, Y., Denker, J.S., Solla, S.A. (1990) "Optimal brain damage, *NIPS*, D.S. Touretzky (ed.), 2, Morgan Kaufmann, 598--605
[23] Leveit, W.J.M. (1990) "On learnability, empirical foundations, and naturalness" *Behavioral and Brain Sciences*, 13, 501
[24] Lippman, R.P. (1987) "An introduction to computing with neural nets" *IEEE ASSP magazine*, 4, 4--22
[25] Mezard, M., Nadal, J.-P. (1989) "Learning in feedforward layered networks: The tiling algorithm" *Journal of Physics A*, 22, 2191--2204
[26] McCarthy, J. (1990) "Interview: Approaches to artificial intelligence" *IEEE Expert*, 5(3), 87--89
[27] Mozer, M.C., Smolensky, P. (1989) "Skeletonization: A technique for trimming the fat from a network via relevance assessment" *Connection Science*, 1, 3--26
[28] Müller, B., Reinhardt, J. (1990) *Neural networks: An introduction*, Springer
[29] Reilly, D.L., Cooper, L.N., Elbaum, C. (1982) "A neural model for category learning" *Biological Cybernetics*, 45, 35--41
[30] Rissanen, J. (1987) "Stochastic complexity" *Journal of Royal Statistical Society B*, 49, 223--239 and 252--265
[31] Rumelhart, D.E., Hinton, G.E., Williams, R.J. (1986) "Learning internal representations by error propagation" *PDP*, D.E. Rumelhart, J.L. McClelland (eds.), 1, MIT Press, 151--193
[32] Siestma, J., Dow, R.J.F. (1991) "Creating artificial neural networks that generalize" *Neural Networks*, 4, 67--79
[33] Weigend, A.S., Rumelhart, D.E., Huberman, B.A. (1991) "Generalization by weight-elimination with application to forecasting, *NIPS*, R.P. Lippman, J. Moody, D.S. Touretzky (eds.), 3, Morgan Kaufmann