Salzburg Interest Group on Integrated Systems

siGis-001-1997 January 5, 1998

Is Evolution a good Teacher? Genetic Algorithms Create Training Data Sets for Artificial Neural Networks

The Third Nordic Meeting on Genetic Algorithms and their Applications
August 18–20, 1997, Helsinki, Finland

Roland Schwaiger
rschwaig@cosy.sbg.ac.at

Helmut A. Mayer
helmut@cosy.sbg.ac.at

Department of Computer Science
University of Salzburg

Correspondence to:

Universität Salzburg
Institut für Computerwissenschaften und Systemanalyse
Jakob–Haringer–Straße 2
A–5020 Salzburg
Austria

Is Evolution a Good Teacher? Genetic Algorithms Create Training Data Sets for Artificial Neural Networks

Roland Schwaiger and Helmut A. Mayer
Department of Computer Science
University of Salzburg
A–5020 Salzburg, Austria
rschwaig@cosy.sbg.ac.at, helmut@cosy.sbg.ac.at

Abstract

For supervised training of an artificial neural network (ANN) the size and composition of the training data set (TDS) is one of the most important prerequisites for an ANN with good generalization abilities. The exisiting methods to construct training data sets (TDSs) vary from random sampling to incrementally increasing the number of samples in the TDS until a certain quality measure is reached. In our approach we employ a genetic algorithm (GA) for the parallel selection of appropriate input patterns for the TDS. The parallel netGEN system that uses a GA to generate problem–adapted generalized multi–layer perceptrons being trained by error–back–propagation has been extended to evolve (sub)optimal TDSs adapted to an (evolved or user–defined) ANN of fixed topology. Empirical results on a benchmark problem are presented.

1 Introduction

Considering real world problems we often face the problem of huge amounts of possible training data patterns, e.g. a robot collecting sensor data from its environment, or satellite image pixels to be classified. Although, the importance of the composition, i.e. the selection of the training data, has been noted by some researchers, mainly rules of thumb of how to design TDSs have been provided by the community [Hay94]. With most ANN applications, either training data patterns are selected by human intuition, or a random sample is drawn to constitute a TDS reflecting the distribution of the original data set. Whereas these methods are simple, robust, and enable approximation of the training data, they might decrease the generalization properties of the ANN, e.g. *Overfitting* [RF95], and unnecessarily increase training time by using too many training data.

In order to construct TDSs so that the ANN can extract the most useful information – based on certain quality measures – of, two main methods have been proposed in literature (see Section 1.1). *Active Selection* builds a small TDS out of all available data patterns by selecting the most useful. *Active Sampling* generates new data patterns based on the data in the initial TDS and adds them to the TDS. Both methods operate sequentially and are dependent on the initial TDS. In this paper we present *Evolutionary TDSs*, a GA–based active selection method, where the patterns of the (sub)optimal TDSs are selected in parallel.

1.1 Related Work

Zhang [Zha94] proposed *Incremental Learning* where a small number of training examples (seed examples) is randomly selected from a candidate set C to build the training set D. The ANN is then trained to a specified performance level and λ new examples are drawn from C based on their *Criticality*. This is a measure used to determine the examples triggering maximum error at the output of the partially trained ANN. This procedure is repeated iteratively and is terminated when a desired ANN performance on a validation set is reached. As the total number of training examples can be reduced by this method, ANN learning is accelerated [Zha94].

In [ZV91] the initial training set D, again, contains only a few seed examples which are processed by a GA to construct new and more examples, or more exactly input vectors. Hence, the corresponding output vectors must be deducible from the generated vectors. As the learner provides novel examples and the teacher the corresponding result, this method is called *Supervised Creative Learning*.

Röbel [Röb94] introduced *Dynamic Pattern Selection*, a method closely related to incremental learning. Here the point in time when a new pattern is added to the training set is determined by the *Generalization Factor* $\rho = \frac{\epsilon(V)}{\epsilon(D)}$, $\epsilon(V)$ being the error on the validation set V and $\epsilon(D)$ on the training set. When network generalization reaches (increasing ρ) a certain threshold, a pattern is added which also gives a means to analyze the relationship between ANN approximation and generalization. Plutowski's [Plu94] active selection method sequentially adds a new input pattern (or *Exemplar*) to the TDS based on the selection criterion ΔISB (*Integrated Squared Bias*) which evaluates the ability of examplars to maximize the decrement in squared error that would result from adding the new exemplar to the training set and training upon the resulting set. Theoretically, an optimal TDS with regard to the ΔISB criterion could be constructed in parallel, but is *"computationally impractical"*[Plu94].

2 System Overview

The netGEN system [HMS95] has been designed for an arbitrary amount of machines/processors employing the *Parallel Virtual Machine (PVM)* library[GBD$^+$94] in order to train evolutionarily generated ANNs in parallel.

Basically, it consists of two architecture components which are incorporated into one system: the ANN generating component and the TDS generating component.

The ANN generating part of the system consists of three main components, the (extended) Simple Genetic Algorithm (netSGA)[SGE91], the Genotype/Phenotype Mapper (netGPM), and the Neural Network Manager (hyperNNM [1]) as shown in Figure 1. For the evolution of TDSs two processes have been added to the system, namely dataSGA and dataGPM.

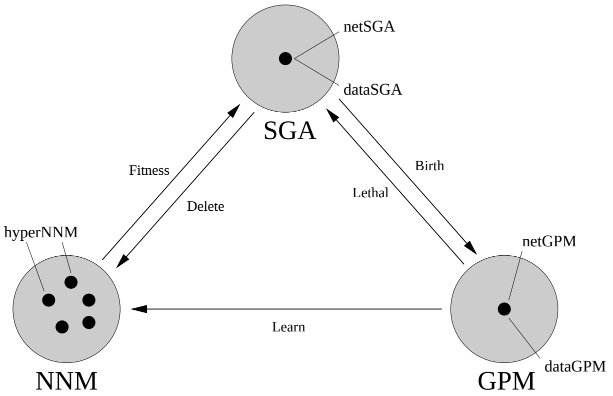

Figure 1: The netGEN system with the Simple Genetic Algorithms (netSGA, dataSGA), the Genotype/Phenotype Mappers (netGPM, dataGPM), and a group of Neural Network Managers (hyperNNM).

netSGA generates a genotype for each ANN which is processed by the netGPM module mapping the ANN genotype to a phenotype adapted to the Neural Network Simulator [ZMV$^+$94]. The mapping module's task is to prepare valid configuration files and to dismiss ANN topologies which cannot be processed by netGEN. The hyperNNM interfaces to the ANN simulator, the network is trained on a training set for the specific number of training epochs, and the fitness of each ANN is evaluated over a validation set and passed back to netSGA. With the parallel implementation a set of hyperNNM processes runs on different machines and/or different processors.

[1] The term *hyper* reflects the use of the same NNM process for net and data evolution.

netGEN may be used in various ways to evolve TDSs. At the time we use a fixed topology ANN, whose architecture does not change during TDS evolution. The ANN is trained via hyperNNM on the basis of different TDSs, whose genotypes are evolved by dataSGA and are mapped to a phenotype, i.e. a TDS file, by dataGPM (netSGA and netGPM are not used in this mode). The fitness of a TDS is determined by the performance of the ANN on a validation set.

2.1 ANN and TDS Fitness Functions

The ANN fitness function comprises a complexity regularization term $\mathcal{E}_c$ suggested by Rumelhart [HP89]

$$\mathcal{E}_c = \sum_{i \in C_{total}} \frac{w_i^2}{1 + w_i^2}, \tag{1}$$

where w_i is the weight of a connection and C_{total} is the set of all network connections. The *Composite Fitness Function* $\mathcal{F}$ is given by a weighted sum of *Model Fitness* and *Complexity Fitness*

$$\mathcal{F} = \alpha_1(1 - \mathcal{E}_m) + \alpha_2 \frac{1}{1 + \mathcal{E}_c} \tag{2}$$

with $\alpha_1 + \alpha_2 = 1.0$, and $\mathcal{E}_m$ being the model error . Earlier work [HSM96] showed that $\alpha 2$ in the range of $0.001 - 0.01$ is sufficient to guide the evolution towards ANNs of low complexity. Thus, we set $\alpha 2 = 0.01$. The model error for classification tasks is simply given by

$$\mathcal{E}_m = \frac{e_v}{n_v}, \tag{3}$$

where e_v is the number of misclassifications on the validation set, and n_v its respective size.

The TDS fitness function is given by setting $\alpha 1 = 1.0$ in Equation 2, thus, the performance of the given ANN trained by the evolved TDS gives the TDS fitness. Additionally, a term evaluating the size (complexity) of the TDS could be incorporated, but this has not been done with experiments in this paper.

3 ANN and TDS Encoding

The main enhancement in our *Modified–Miller–Matrix (MMM)* direct encoding scheme [HMS95] based on Miller et al. [MTH89] are the *Neuron Markers* which are encoded in a separate section on the ANN chromosome (the two other sections contain learning parameters and all possible connections, respectively). Technically, each neuron marker (one bit) is transferred to the main diagonal (i, i) of the MMM adjacency matrix, indicating the absence or presence of a particular neuron i and its connections (Fig. 2). As a consequence, most ANN chromosomes contain non–coding regions [2], as all connections associated with a specific neuron become obsolete, if the corresponding neuron marker is zero. The non–coding regions (*Introns*) reduce the disruptive effects of crossover [Lev91] [Wu96].

The maximum number of hidden neurons (neuron markers) has to be set in advance with this evolution scheme, hence, it could be labeled as *Evolutionary Pruning*, since the system imposes an upper bound on the complexity of the network. One of the measures to avoid overfitting is the encoding of the number of training epochs with the additional benefit of another ANN parameter being set automatically. Two learning parameters are also encoded in the genotype, namely the *Resilient Propagation* (RPROP [RB93]) learning parameters Δ_0 and Δ_{max} defining the initial and maximum stepsize, respectively.

In addition to our modifications to the MM encoding scheme, the GA does not operate on the matrix representation, but on the concatenation of the rows of the non–zero part of the MMM's lower triangle matrix representing the possible connections. All MMM entries clamped to zero need not be processed by the GA, hence, they are not encoded in the genotype.

The TDSs are encoded as a mapping of single bits to each item in a data store file containing all available training patterns (Figure 3). Thus, a 1 for the corresponding item indicates the selection of the pattern for the TDS, whereas a 0 dismisses the specific item. This method ensures that all patterns in the data store

[2] Non–coding DNA regions or *Junk DNA* can be found in abundance in biological systems, i.e. *Eukaryotes*.

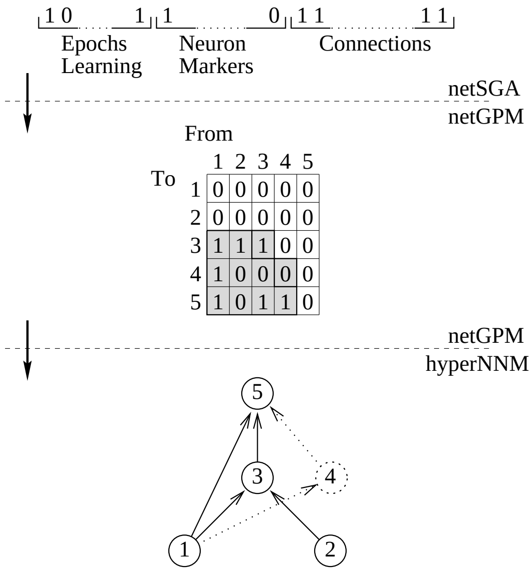

Figure 2: ANN encoding scheme.

can be chosen, and no duplicate patterns can be contained in the constructed TDS. Moreover, the size of the TDS can vary from 0 to N, the number of patterns in the data store, hence, the TDS size is evolved as well.

Both GAs, netSGA and dataSGA, use the same basic genetic operators which are given below. For the GA selection phase we chose *Binary Tournament Selection*, for it showed mostly better results than *Proportionate Selection Methods*. First analytical steps, possibly confirming this observation, have been reported [BT95].

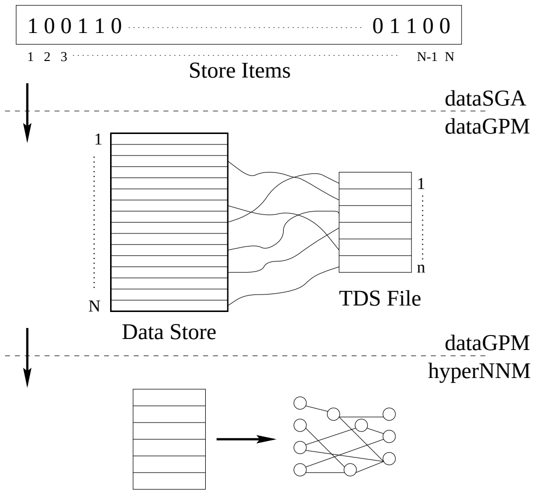

Figure 3: TDS encoding scheme.

3.1 GA and ANN Parameters

The following GA and ANN parameters have been used with all the experiments in this paper:
GA Parameters: Population Size = 50, Generations = 50 (10), Crossover Probability $p_c = 0.6$, Mutation Probability $p_m = 0.005$, Crossover = 2–Point, Selection Method = Binary Tournament.

ANN Parameters: Network Topology = Generalized Multi–Layer Perceptron, Activation Function (all neurons) = Sigmoid, Output Function (all neurons) = Identity, Training = RPROP, Hidden Neurons (max 10), Learning Parameters Δ_0 (max 1.0), Δ_{max} (max 50.0), Number of Training Epochs (max 1500) = Evolutionary.

4 Benchmark Problem

From the PROBEN1 benchmark suite [Pre94] of real-world data a problem with a rather large number of patterns in the TDS was chosen. This should simulate a problem, where a large quantity of data has been collected, and (sub)optimal TDSs should be constructed of these data. However, it should be noted that the TDSs of these benchmark problems have been optimized by many previous experiments, i.e. they already are a result of an evolutionary search process. Table 1 shows the parameters of the discussed data sets of problem *Thyroid* and the initial number of hidden units.

The problem is to determine, whether a patient has abnormal thyroid function based on 21 patient query and examination data used as input. The 3 ANN output neurons should give the following diagnosis: normal (class 2), overfunction (*hyperthyroid*, class 1), and underfunction (*hypothyroid*, class 0). Because app. 92% of the patients have normal thyroid function, a good classifier must be significantly above this value. In [Pre94] three different compositions of training-, validation and test sets are given. We chose *thyroid1*, where Prechelt reported a validation set accuracy $v_{acc} = 0.9883$, and a test set accuracy $t_{acc} = 0.9800$ with a $21 - 16 - 8 - 3$ fully connected architecture yielding 488 connections. Due to the netGEN complexity regularization, we were able to set the maximum number of hidden neurons to 10, because first netGEN runs with 24 initial hidden neurons generated networks with 4–7 hidden neurons.

Data Set Name	Input	Output	Hidden	Training	Test	Validation
Thyroid	21	3	10	3600	1800	1800

Table 1: Parameters for the data sets and the network topology.

5 Experimental Results

In order to find a problem–adapted ANN as the fixed topology environment for the TDS evolution, netGEN has been run in the ANN toplogy search mode. Due to the large thyroid TDS, we employed *Time–Dependent Epoch Compression* to speed up the evolutionary search. The (evolved) number of epochs is compressed by a factor which is derived from a polynomial compression function $C(g) = (\frac{g}{g_{max}})^{s-1}$, where g is the current generation, g_{max} the maximum number of generations, and s the *Evolutionary Speedup*. This method will be investigated in more detail in future work, but preliminary results indicate that ANNs found with substantial speedup vary only slightly with respect to classification accuracy, when compared to a full search ($s = 1.0$). All ANN evolutions in this paper have been run with an evolutionary speedup of $s = 8.0$, whereas TDS evolutions have been given full time ($s = 1.0$).

The ANN found with the original thyroid1 TDS consisted of 2 hidden neurons with 63 connections. Training the network with the evolved number of 1373 epochs yielded $v_{acc} = 0.9917$, and $t_{acc} = 0.9889$. This evolved ANN has been used as a fixed topology environment for the evolution of TDSs. As the best TDSs have been found in the very first generations in runs with 50 generations, we reduced the maximum number of generations to 10 and ran five runs with different random seeds (evolved TDSs $A - E$). The ANN specific results are given in Table 2.

Four of the five evolved TDSs improved the classification accuracy on the test set (t_{acc}), however, the increase is very small considering that $t_{acc} = 0.9900$ corresponds to two additional correct classifications compared to the original TDS. These two patterns were mostly in class 0 (hyperthyroid).

Table 3 gives the details on the original and evolved TDSs.

The decrease of evolved TDS size is in a modest range of 3%, and the relative class distribution $h_{ci}, i \in \{0, 1, 2\}$ of training patterns is very similar. However, two facts are quite interesting. The size of the TDSs

ANN Results					
Patterns	1800	1800	40	91	1669
TDS	v_{acc}	t_{acc}	$c0_{acc}$	$c1_{acc}$	$c2_{acc}$
O	0.9917	0.9889	0.9000	0.9890	0.9910
A	0.9922	0.9894	0.9500	0.9780	0.9910
B	0.9922	0.9900	0.9500	0.9890	0.9910
C	0.9922	0.9900	0.9500	0.9890	0.9910
D	0.9922	0.9900	0.9500	0.9890	0.9910
E	0.9911	0.9889	0.9250	0.9780	0.9910

Table 2: Validation , test set, and test class accuracies for original TDS (O) and evolved TDSs ($A - E$).

TDS Results						
TDS	*Size*	*Identical*	h_{c0}	h_{c1}	h_{c2}	Reduction
O	3600	62	0.0253	0.0503	0.9244	0.0000
A	3489	61	0.0244	0.0499	0.9258	0.0308
B	3486	61	0.0244	0.0499	0.9257	0.0317
C	3488	61	0.0244	0.0499	0.9258	0.0311
D	3487	61	0.0244	0.0499	0.9257	0.0314
E	3563	62	0.0250	0.0494	0.9256	0.0103

Table 3: Original (O) and evolved TDS ($A - E$) sizes.

yielding best ANN performance is nearly identical. TDS E is the only set with a larger number of patterns (close to the original TDS size), but does not improve classification accuracy. Even more surprising is that a comparison of TDSs B and D – differing by only one pattern in size – revealed that all patterns of TDS B are in D. Though, the observed performance increase is very small, it seems that there is a pronounced optimum of TDS size and contents.

Moreover, earlier results and conclusions [MS97] are backed up by experiments with this benchmark problem. The complexity of the evolved ANN is an indicator for the quality of the TDS. ANNs with few (or even no) hidden neurons correspond to a "good" TDS, whereas non–optimal TDSs induce the evolution of rather complex ANNs. In [MS97] for a simple pattern recognition problem the number of connections of the evolved ANN could be reduced by a factor of 7.09 by using an evolved TDS instead of a TDS of equal size created by humans. With the thyroid problem the complexity could not be further reduced by using an evolved TDS indicating the quality of the original TDS. Thus, the question posed in the title can be answered positively considering that the original TDS has already been evolved by researchers experimenting with the data. The additional artificial evolution could still reduce TDS size and slightly improve ANN performance with an evolved network of substantial lower complexity (a decrease by a factor of 7.75 with respect to the number of connections compared to [Pre94]).

6 Outlook

Though, the evolution of TDSs of the thyroid benchmark problem resulted only in small improvements of ANN performance and TDS size, it is a promising method for real world problems with a huge amount of possible training data. Thus, we will further investigate TDS evolution with remotely sensed imageries, specifically, *Landsat TM* data, and their use for ANN *Land Cover Classification*, where no generally accepted guidelines for the construction of TDSs exist. Another future research topic will be the coevolution (i.e. simultaneous evolution) of ANN and TDS populations. With this scheme ANN generalization might be further improved, as (sub)optimal ANNs have to survive even "bad" TDSs they are trained with (vice versa for TDSs).

7 Acknowledgements

We wish to thank the *Austrian Center for Parallel Computation* (ACPC), Group Salzburg, for making available a cluster of DEC AXP workstations where the netGEN system has been ported to. We very much appreciate the support of our student Marc Strapetz, who helped to finish this paper in time. Many thanks to Ari Nissinen (Helsinki University of Technology) and an anonymous referee for very constructive remarks.

References

[BT95] Tobias Blickle and Lothar Thiele. A Mathematical Analysis of Tournament Selection. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the 6th International Conference (ICGA95)*, San Mateo, California, 1995. Morgan Kaufman Publishers, Inc.

[GBD+94] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM 3 User's Guide and Reference Manual.* Oak Ridge National Laboratory, 1994.

[Hay94] Simon Haykin. *Neural Networks. A Comprehensive Foundation.* MacMillan, 1994.

[HMS95] Reinhold Huber, Helmut A. Mayer, and Roland Schwaiger. netGEN - A Parallel System Generating Problem-Adapted Topologies of Artificial Neural Networks by means of Genetic Algorithms. In *Beiträge zum 7. Fachgruppentreffen Maschinelles Lernen der GI-Fachgruppe 1.1.3, Forschungsbericht Nr. 580, Dortmund*, August 1995.

[HP89] S.J. Hanson and L.Y. Pratt. Comparing biases for minimal network construction with backpropagation. In D. Touretzky, editor, *Advances in Neural Information Processing Systems, Vol. 1*, volume 1, pages 177–185. Morgan Kaufmann, 1989.

[HSM96] Reinhold Huber, Roland Schwaiger, and Helmut A. Mayer. On the Role of Regularization Parameters in Fitness Functions for Evolutionary Designed Artificial Neural Networks. In *World Congress on Neural Networks*, pages 1063–1066. International Neural Network Society, Lawrence Erlbaum Associates, Inc., 1996.

[Lev91] James R. Levenick. Inserting Introns Improves Genetic Algorithm Sucess Rate: Taking a Cue from Biology. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 123–127. University of California, San Diego, Morgan Kaufmann, 1991.

[MS97] Helmut A. Mayer and Roland Schwaiger. Towards the Evolution of Training Data Sets for Artificial Neural Networks. In *Proceedings of the 4th IEEE International Conference on Evolutionary Computation*, pages 663–666. IEEE Press, 1997.

[MTH89] Geoffrey F. Miller, Peter M. Todd, and Shailesh U. Hegde. Designing Neural Networks using Genetic Algorithms. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 379–384, San Mateo, California, 1989. Philips Laboratories, Morgan Kaufmann Publishers, Inc.

[Plu94] Mark Plutowski. *Selecting Training Examplars for Neural Network Learning.* PhD thesis, University of California, San Diego, 1994.

[Pre94] Lutz Prechelt. PROBEN1 - A Set of Neural Network Benchmark Problems and Benchmarking Rules. Technical Report TR 21/94, Universität Karlsruhe, Fakultät für Informatik, 76128 Karlsruhe, Germany, September 1994.

[RB93] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Proceedings of the IEEE International Conference on Neural Networks*, San Francisco, CA, April 1993.

[RF95] Paul L. Rosin and Freddy Fierens. Improving Neural Network Generalisation. In *Proceedings of IGARSS'95, Firenze, Italy*, July 1995.

[Röb94] A. Röbel. The Dynamic Pattern Selection: Effective Training and Controlled Generalization of Backpropagation Neural Networks. Technical report, Technical University Berlin, 1994.

[SGE91] Robert E. Smith, David E. Goldberg, and Jeff A. Earickson. SGA-C: A C–Language Implementation of a Simple Genetic Algorithm. TCGA Report 91002, The Clearinghouse for Genetic Algorithms, The University of Alabama, Department of Engineering Mechanics, Tuscaloosa, AL 35487, May 1991.

[Wu96] Annie Siahung Wu. *Non–Coding DNA and Floating Building Blocks for the Genetic Algortihm.* PhD thesis, University of Michigan, 1996.

[Zha94] Byoung-Tak Zhang. Accelerated Learning by Active Example Selection. *International Journal of Neural Systems*, 5(1):67–75, 1994.

[ZMV+94] Andreas Zell, Guenter Mamier, Michael Vogt, Niels Mach, Ralf Huebner, Kai-Uwe Herrmann, Tobias Soyez, Michael Schmalzl, Tilman Sommer, Artemis Hatzigeogiou, Sven Doering, and Dietmar Posselt. *SNNS Stuttgart Neural Network Simulator, User Manual.* University of Stuttgart, 1994.

[ZV91] Byoung-Tak Zhang and Gerd Veenker. Neural Networks that Teach Themselves through Genetic Discovery of Novel Examples. In *Proceedings International Joint Conference on Neural Networks (IJCNN '91)*, pages 690–695. IEEE Press, 1991.