Salzburg Interest Group on Integrated Systems

siGis-004-1996 January 5, 1998

Towards the Evolution of Training Data Sets for Artificial Neural Networks

IEEE International Conference on Evolutionary Computation
April 14–17, 1997, Indianapolis

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

Roland Schwaiger
rschwaig@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

Towards the Evolution of Training Data Sets for Artificial Neural Networks

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

While most efforts in artificial neural network (ANN) research have been put into the investigation of network types, network topologies, various types of neurons and training algorithms, work on training data sets (TDSs) for ANNs has been comparably small. There are some approximations for the size of ANN TDSs, but little is known about the quality of TDSs, i.e. selecting data sets from which the ANN can draw the most information. As a matter of fact, with most real world applications not even human experts being familiar with the problem can give accurate guide lines for the construction of the TDS. In order to automatize this process we investigate the use of a genetic algorithm (GA) for the selection of appropriate input patterns for the TDS. The parallel netGEN system which 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. Empirical results on a simple example 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, at most 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 ensure good approximation of the training data, they might decrease the generalization properties of the ANN, i.e. *Overfitting* [RF95], and unnecessarily increase training time by using too many training data.

In order to construct TDSs 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* which 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. 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 where 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 workstations employing the *Parallel Virtual Machine (PVM)* library [GBD+94] and is currently running on a network cluster of DEC AXP workstations (10 machines) in order to train individual ANNs in parallel. The network 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.

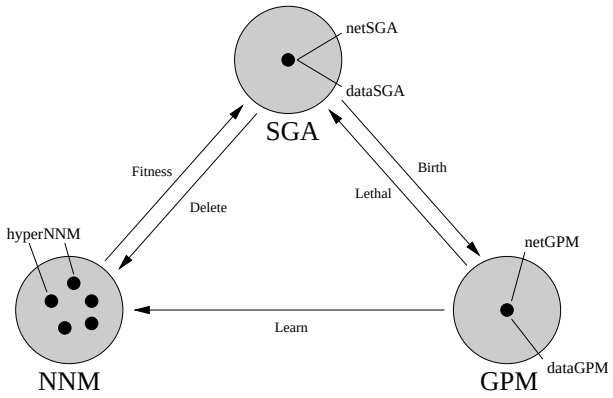

Figure 1: The netGEN system with the Simple Genetic Algorithms (netSGA, dataSGA), the Genotype/Phenotype Mappers (netGPM, dataGPM), and a group of Neural Network Managers (hyperNNM).

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

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.

Employing netGEN for the evolution of TDSs we use a fixed topology ANN which is trained via hyperNNM by 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.

Furthermore, we achieve *Implicit Load Balancing* which means that more ANNs are passed to the faster machines within the PVM. An adaptive time–out mechanism prevents the system from deadlock. Performance reducing situations at the end of a generation due to overloaded machines are handled by *Bench Penalties* (i.e. the overloaded machine(s) take(s) a rest for a number of generations).

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 *Evolu-*

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

tionary 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. The standard back–propagation learning rate η, and a parameter δ_ϵ [3] which may also reduce overfitting, as the ANN can often be trained with substantial fewer epochs.

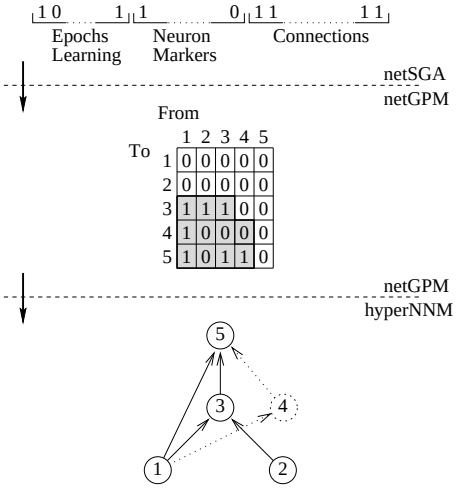

Figure 2: ANN encoding scheme.

In addition to our modifications of 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 concatenation of the desired number of training patterns (genes) which are represented as binary coded indexes to a data store file containing all available training patterns (Fig. 3). Clearly, the number of bits used to index an item in the data store is dependent on the size of the store. By now, the number of training patterns has to be set by the user, but a similar procedure as outlined above (i.e. the use of *Data Markers*) could additionally evolve TDS sizes. 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 recently [BT95].

[3] δ_ϵ specifies an error threshold for all output neurons . If the error is below that threshold, the output neuron is considered to have zero error.

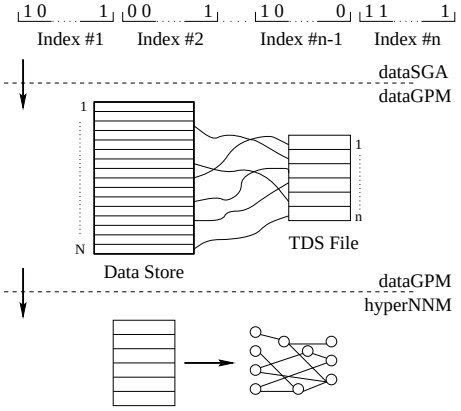

Figure 3: TDS encoding scheme.

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, 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 = Error–Back–Propagation, Hidden Neurons (max 10), Learning Parameters η (max 1.0), δ_ϵ (max 0.4), Number of Training Epochs (max 2000) = Evolutionary.

4 TDS Evolution Results

A simple test problem, the detection of lines and their directions, has been adopted from [HMS95]. The input to the ANN is a block of 3×3 black or white pixels, each corresponding to an input neuron. The output layer consists of four neurons activated depending on the line(s) contained in the input pattern. Figure 4 shows the input patterns with the corresponding output layer, i.e. the four directions of lines the ANN should detect, of a 22 pattern TDS from [HMS95] which has been created by human intuition (provided the authors have some).

First, the human TDS has been used to evolve an ANN topology where the ANN fitness is calculated from the classification performance on the validation set which is identical with the data store, i.e. all 512 possible patterns. The *Model Fitness Function* is given by $\mathcal{F}_m = \frac{N(V)-E(V)}{N(V)}$, with $N(V)$ being the

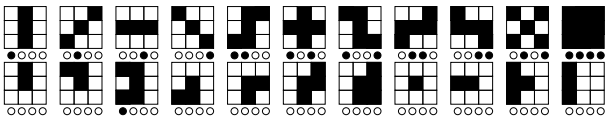

Figure 4: The human TDS, 22 input patterns and the desired output activations.

size of, and $E(V)$ the number of misclassifications over the validation set. Using a *Composite Fitness Function* by building a weighted sum of model and *Complexity Fitness* [HSM96] so as to decrease network complexity the evolved architecture has 10 hidden neurons and 163 connections with a fitness of 0.6992 (i.e. about 70% of the validation set has been classified correctly, Fig. 5).

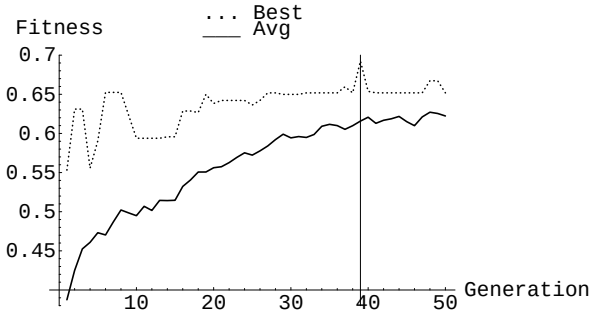

Figure 5: ANN evolution with human 22 pattern TDS.

Second, a 22 pattern TDS has been evolved (Fig. 6) in the environment of this ANN topology and the TDS fitness being identical to the ANN model fitness reached 1.00. The patterns in the evolved TDS are irregular and more complex than in the human TDS and contain no explicit examples of specific lines.

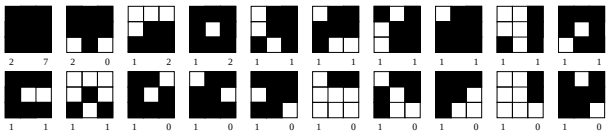

Figure 6: The evolved TDS, 22 input patterns, the lower left number indicates the occurrences of the pattern within the TDS, the lower right number indicates the occurrences in all other evolved TDSs (21–1 patterns).

Third, TDSs with decreasing number of patterns (from 21 to 1) have been evolved by netGEN (Fig. 7) resulting in a TDS fitness of 1.00 for a pattern number above 17. The 7 pattern TDS with a fitness of 0.6934 still performed comparably to the human 22 pattern TDS .

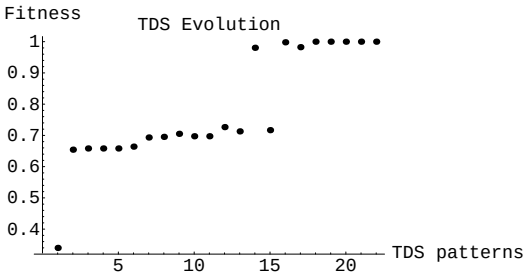

Figure 7: The fitness values of the best evolved TDSs with varying number of training data (1–22 input patterns).

Fourth, the evolved 22 pattern TDS has been used as an environment for another net evolution. The generated ANN had a classification performance of 100% and consisted of 0 (zero) hidden neurons and 23 connections, again, using complexity regularization (Fig. 8). While the classification accuracy of the ANN has been increased substantially with the evolved TDS, the complexity of the network could be decreased by a factor of 7.09 (with respect to the number of connections).

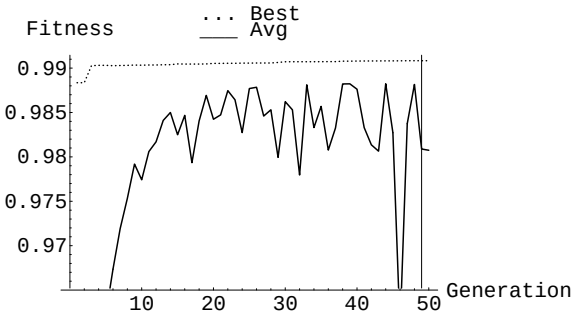

Figure 8: ANN evolution with evolved 22 pattern TDS, all other netGEN parameters unchanged.

5 Outlook

Though, the results of this work are promising, the evolution of TDSs remains to be studied with more difficult problems (benchmark and real world problems), but it could be a method to substantially decrease the size of often huge TDSs, at the same time improving ANN performance by selecting input patterns in parallel, i.e. complete TDSs, the ANN can generalize best from. A future research topic will be the co–evolution (i.e. simultaneous evolution) of ANNs and TDSs.

6 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 and like to express our gratitude to our colleague Reinhold Huber for very productive past and future collaborations. Last but not least, we like to thank two anonymous referees for their 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.

[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.

[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 Kaufman Publishers, Inc.

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

[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.