MULTI–CHROMOSOMAL REPRESENTATIONS IN NEUROEVOLUTION

August Mayer and Helmut A. Mayer
Department of Computer Sciences
University of Salzburg
Jakob–Haringer–Straße 2
A-5020 Salzburg, Austria
email: {amayer,helmut}@cosy.sbg.ac.at

ABSTRACT
Contemplating the development of the field of evolutionary computation (EC), where most basic concepts are borrowed from nature, it is remarkable that multi–chromosomal representations present in all complex organisms have rarely been studied in the artificial domain. Evidently, the addition of such an additional layer of genetic code must prove to possess certain benefits before it is introduced. Thus, we present experiments with multi–chromosomal evolution of artificial neural networks (ANNs) on three benchmark problems with the goal to investigate potential advantages of genotypes with multiple chromosomes. Besides the technical benefit of using different encodings, genetic operators, and parameters for specific chromosomes, we hypothesize that the generalization capabilities of evolved networks (or other structures) increase, when their genotype is of multi–chromosomal organization.

KEY WORDS
Neuroevolution, multi–chromosomal representation, ANN classification, ANN prediction

1 Introduction

Multi–chromosomal genotype representation is an approach not only stressing the biological roots of artificial evolution, but also offering some potential technical advantages. As the genetic information is divided and packaged on several chromosomes, a new hierarchical level of genotype representation is introduced, which gives rise to an additional recombination operator, i.e. *Chromosome Shuffling*. In this process, complete chromosomes are exchanged between parents. At a certain stage of evolution, the chromosomes may represent useful solutions for a specific sub-problem, which could be of immediate benefit in another genome. We hypothesize that chromosomes are evolved towards general usability in a genotype, as specialized chromosomes relying on other chromosomes to generate a phenotype of high fitness will be weeded out by the shuffling procedure. Consequently, evolved solutions based on multi–chromosomal representations might be more general than those employing conventional techniques with single chromosomes.

To paint these theoretical considerations with some color, imagine components of a car with the car body (chromosome A), a motor (chromosome B), and wheels (chromosome C). If in a car design X of high fitness, wheels and motor are so specialized that they only fit to the specific car body, the exchange (shuffling) of chromosomes (e.g., wheels) with another car design Y will leave X without wheels. If the car body of Y accepts a variety of wheels, Y will still represent a valid car design. Thus, in the long run only car bodies will survive that can cope with a variety of motors and wheels (and vice versa), which in the end makes it likely that the evolved car can also take on wheels never tried during evolution, i.e., a general car design has been evolved.

The above hypothesis could be confirmed in multi–chromosomal evolution of fuzzy controllers [1]. Controllers evolved to solve the inverted pendulum problem performed better on unknown test cases: they had better generalization capabilities when their genotype was split into multiple chromosomes.

Another interesting property of multi–chromosomal representations is the potential to use different encodings in a genotype. For example, with ANN evolution, the network structure can be encoded by bit strings (direct representation), while the connection weights may be encoded by real values in separate chromosomes.

In this work we investigate the validity of the hypothesis (improved generalization induced by multi–chromosomal representations) in the realm of neuroevolution. Both network structure, and connection weights of an *Artificial Neural Network* (ANN) are evolved without need for conventional training. Three benchmark problems, namely the two classification problems *Glass* and *Two Spirals*, and the prediction problem *Mackey–Glass* are utilized for evolution experiments. For each of these, the performance of the resulting neural networks (with different encodings) is compared on training and test sets, along with the size (number of neurons and links) of the evolved network structures.

Simultaneous evolution of network structure and connection weights is often the method of choice when explicit training data are not available. For example, in *Evolutionary Robotics* the behavior (fitness) of a robot controlled by an ANN can be assessed, but it is mostly impossible to

provide coherent sensor–motor mappings for each situation the robot may experience. *NeuroEvolution of Augmenting Topologies* (NEAT) is a recent development in the evolution of ANN structure and weights, which has been effectively applied to a pole–balancing task in [2]. However, if training data are available, most systems incorporate conventional training so as to systematically exploit the knowledge contained in the teaching examples. Another system evolving structure and weights is *EPNet*, which has been applied to real–world benchmark problems, but employs conventional (partial) training as the mutation operator of highest priority [3]. In this work we evolve ANN structure and weights for well–known benchmark problems without utilizing any conventional training methods.

This paper first introduces the concepts of multi–chromosomal neuroevolution in section 2. Section 3 describes the experimental setup and the parameters used for each of the benchmark problems. Section 4 presents the results of the various experiments, and section 5 concludes with a summary and potential future work.

2 Multi–chromosomal ANN Evolution

With multi–chromosomal representations, the genetic operators, evolution parameters, and encoding of each chromosome may be different. As an example, fig. 1 displays the ANN genotype representations used in the experiments below.

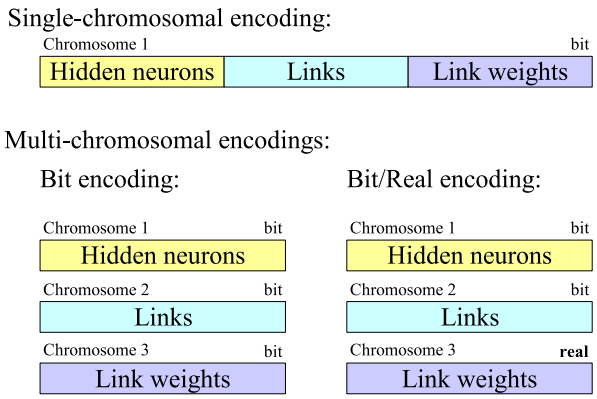

Figure 1: Single- and multi-chromosomal ANN genotype representations.

The conventional genetic operators can be extended in a straight-forward manner. Mutation is performed separately on each chromosome, and crossover is restricted to exchange only parts of corresponding chromosomes (such as hidden neurons; see [4] for a different crossover approach). The additional chromosome shuffling operator [5] enables the exchange of genetic material between individuals on the level of partial solutions. Chromosome shuffling is motivated by the biological process of *Meiosis*, where after crossover the recombined chromosomes of mother or father separate randomly to different cell areas to form a new (shuffled) set of chromosomes [6]. Similarly, in the artificial domain chromosome shuffling is implemented by exchanging complete chromosomes of two parents with a shuffle rate of p_s. Assume that in fig. 1 the two multi–chromosomal encodings represent two individuals (each having three chromosomes) of a population. During shuffling each chromosome of a parent is exchanged with the corresponding chromosome (for example, links) of the other parent with probability p_s.

Evolutionary computation techniques allow to automate the process of ANN design (e.g., structure, weights) and may optimize other system parameters (e.g., learn rate, composition of training data). Usually, the evolution of the network structure is combined with conventional ANN training, if appropriate training data sets are available. Though the latter is obviously true for the benchmark problems utilized in this work, we also evolve the connection weights in our experiments, which is known to require a rather large number of individuals and generations. However, having in mind the investigation of potential benefits of multi–chromosomal representations on the generalization capabilities of ANNs, we wanted to employ a "pure" evolution system, as the main hypothesis (improved generalization induced by chromosome shuffling) is grounded on evolutionary arguments.

So far, multi–chromosomal evolution has not been in the spot light of evolutionary computation research. One of the earliest papers using the term is by Hinterding and Juliff (1993) describing the application on a stock cutting problem [7]. Ronald et al. (1997) presented multi–chromosomal experiments solving a modified traveling salesperson problem [8]. Another application of multi–chromosomal encoding is described by Bhatia and Basu (2004) [9]. Cavill et al. (2005) explore the concepts of polyploidy and dominance with multi–chromosomal evolutionary algorithms [4].

3 Experimental Setup

In our experiments we compare ANN representations using a single bit (string) chromosome (called *single* in the following) with two multi–chromosomal representations, namely, three bit chromosomes (*multi–bit*), and two bit chromosomes and a real-valued chromosome (*multi–real*). The corresponding encoded ANN parameters are the network structure (hidden neurons, links) and the link weights. The direct encoding of network structure, where each hidden neuron and link is represented by a single bit, results in a *Generalized Multi-Layer Perceptron* (GMLP), which allows any connection in the forward direction, e.g., direct connections from input to output.

The links are encoded in a (linearized) adjacency matrix describing the network structure. If a neuron is turned off (the corresponding bit is 0), all associated connections become obsolete. If any input or output neuron does not have a connection to a hidden neuron, the network is rendered invalid by assigning a fitness of 0.

The connection weights are encoded using 16 bits (bit chromosome) mapped into the interval $[-100,100]$, or a real value (real chromosome), which adds a flavor of *Evolution Strategies* to the genotype (fig. 1).

The mutation operator for the binary chromosomes and the real number chromosome is the standard bit flip mutation, and σ-self-adaption (σ-mutation) [10], respectively. With σ-mutation each object parameter x_i (here a connection weight) has an associated strategy parameter σ_i controlling mutation of the object parameter as given by

$$x_i' = x_i + \sigma_i' \cdot N(0,1), \tag{1}$$

where x_i' is the mutated object parameter, and $N(0,1)$ the normal distribution. The strategy parameters σ_i are mutated according to

$$\sigma_i' = \sigma_i \cdot e^{(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1))}, \tag{2}$$

with $\tau' = \left(\sqrt{2n}\right)^{-1}$, $\tau = \left(\sqrt{2\sqrt{n}}\right)^{-1}$, n being the number of object parameters, and $N_i(0,1)$ indicating that a new random number is drawn from the distribution for each strategy parameter.

The recombination operator for all chromosomes is 2–point crossover (occurring separately on each chromosome), and the selection method of choice is *Binary Tournament* selection with replacement.

For a fair comparison of the various representations, we need to match the crossover rates. This is done by considering the probability p_{NOR} that an individual is not altered under crossover and shuffling:

$$p_{NOR} = [p_s^c + (1-p_s)^c](1-p_c)^c \tag{3}$$

where p_s is the shuffle rate, p_c the crossover rate, and $c > 0$ is the number of chromosomes. By equating p_{NOR} for single- and multi–chromosomal representations, we obtain the matched crossover rate for multiple chromosomes $p_{c,M}$:

$$p_{c,M} = 1 - \sqrt[c]{\frac{1-p_{c,S}}{p_s^c + (1-p_s)^c}} \tag{4}$$

with $p_{c,S}$ being the crossover rate in the single-chromosomal case. Note that the matched crossover rates $p_{c,M}$ and $p_{c,S}$ are different even without shuffling ($p_s = 0$), as crossover is restricted to single chromosomes (the number of potential crossover events is equal to the number of chromosomes).

Table 1 shows the various evolution parameters used for the experiments. The values of the matched crossover rates seem to be unusual, but a more conventional setting of $p_{c,S} = 0.6$ would result in a negative $p_{c,M}$ (Equation 4).

The (problem–specific) fitness function $\mathscr{F}_m$ (model fitness) assesses the performance of the evolved ANN. The model fitness is evaluated on a training set, while the final performance of an evolved ANN is determined by using a test set with data not presented during evolution.

Runs:	30
Generations:	1,000
Individuals:	150
Mutation Rate:	$\frac{5}{l}$ (l . . . genotype length)
Crossover Rate:	$p_{c,S} = 0.817$
	$p_{c,M} = 0.1$
Shuffle Rate:	$p_s = 0.5$
Selection:	Binary Tournament

Table 1: General ANN evolution parameters.

The experiments have been conducted using the Java framework *netJEN*, an ANN evolution environment, developed by the authors and colleagues. The following three benchmark problems have been adopted.

3.1 Glass

Glass is a classification problem from the PROBEN1 neural benchmark suite [11]. The problem stems from criminology, where glass splinters need to be analyzed and categorized into different types of glass, e.g., window glass. An input pattern is made of values for the refractive index, and concentrations of eight chemical elements in the analyzed splinters (nine parameters). The six output classes are float- and non–float processed building window glass, vehicle window glass, container, tableware, and headlamp glass. For the experiments, the *glass2* data set with 161 training and 53 test patterns is used. The maximal number of hidden neurons is set to 20.

The fitness function $\mathscr{F}_m$ used for the Glass (and the Two Spirals) problem is simply the classification accuracy $\mathscr{F}_m = 1 - \frac{N_{err}}{N_{pat}}$ with N_{pat} being the number of patterns in the training set, and N_{err} the number of incorrectly classified patterns.

3.2 Two Spirals

The *Two Spirals* problem is a well–known classification benchmark from the CMU neural benchmark repository [12], which is known to be hard to solve for neural networks. The goal is to discriminate between two spirals coiled inside each other inside the unit square. The 388 training and 194 test patterns with two inputs (for x– and y–coordinates of a point) and two outputs (one for each spiral) are generated by adding constant (small) noise terms to the points on the spirals. The maximal number of hidden neurons is defined to be 20.

3.3 Mackey–Glass

The *Mackey–Glass* chaotic time series is a function prediction problem, again, from the CMU neural benchmark repository [12]. The Mackey–Glass time series [13] is represented by the ANN input vector of $\{x(t-18), x(t-12), x(t-6), x(t)\}$ used to predict a future value $x(t+P)$

(the single output neuron has linear activation function). Since the function is chaotic, this task is more difficult for values of P greater than its characteristic period of approximately 50.

For the experiments a value of $P = 84$ is used for the 3,001 training and 501 test patterns. The maximal number of hidden neurons is 20. The fitness function is given by $\mathscr{F}_m = \frac{1}{1+SSE}$, where *SSE* is the *Sum Square Error* on the training set.

4 Experimental Results

From the 30 evolutionary runs the performance of the overall best ANN and mean classification errors (with standard deviation) of the best ANNs of each run are presented.

4.1 Glass Results

Table 2 gives statistical details on the results for the Glass problem. Training and test set errors are the percentage of incorrectly classified patterns. The *Neurons* value is the number of hidden neurons.

Best Net	*single*	*multi–bit*	*multi–real*
Train error	**18.63**	20.5	24.22
Test error	**28.3**	32.08	32.08
Neurons	**7**	11	9
Links	**66**	122	114
Means			
Train error	**24.82**	25.73	29.75
StdDev.	5.86	6.07	5.38
Test error	**41.64**	41.76	43.9
StdDev.	7.01	5.59	6.44
Neurons	**7.33**	8.53	9.13
StdDev.	2.05	3.66	2.51
Links	**82.9**	125.23	116.13
StdDev.	26.16	47.1	33.74

Table 2: Glass Evolution Statistics (averaged on 30 runs).

	Avg	StdDev	Best	Worst
Training	19.63	1.92	16.82	23.36
Test	36.6	3.79	32.08	43.4

Table 3: Glass Training Statistics (averaged on 10 runs).

In this example, the conventional single–chromosomal encoding produces slightly better results than the multi-chromosomal variants. The most notable difference is in the mean number of links with *single* generating the smallest networks.

To put these results into perspective, we have trained a $9-8-6$ GMLP (sigmoidal activation functions) having all possible forward connections (174) for 500 epochs with the back–propagation variant *Rprop* [14]. The results (averaged on 10 runs) are presented in table 3.

Compared to the evolution results, the mean test set error after training is lower, but the best test set errors achieved by evolution are equal (*multi*) or even better (*single*). These results demonstrate the potential of combined structure and weight evolution, evidently exhibiting a larger variance of results than the directed training method.

Notably, evolution and training results are considerably superior to those reported in [11] (p28, table 5) with a test set error of 52.83%. Better results are reported in [15] using a neural network ensemble approach with test set errors from 20% to 30%. However, in the latter work no indication is given which of the three PROBEN1 glass data sets have been selected (the one used here, *glass2*, is the most difficult).

4.2 Two Spirals Results

The results for the experiments with the Two Spirals problem are given in table 4. Again, training and test set errors are the percentage of incorrectly classified patterns, and the *Neuron* values are the hidden neuron counts.

Best Net	*single*	*multi–bit*	*multi–real*
Train error	25.26	**23.45**	31.44
Test error	27.84	**24.23**	33.51
Neurons	9	**8**	11
Links	**44**	46	62
Means			
Train error	32.33	**30.43**	38.74
StdDev.	3.88	2.86	2.39
Test error	35.12	**33.21**	40.5
StdDev.	3.2	3.25	2.42
Neurons	**7.17**	8.6	9.43
StdDev.	2.24	1.85	2.78
Links	**30.23**	47.2	46.73
StdDev.	12.43	20.45	20.28

Table 4: Two Spirals Evolution Statistics (30 runs).

The multi–chromosomal representation with binary encoding wins marginally over the *single* representation, while *multi–real* generates the largest mean test error. Again, on average *single* clearly produces the smallest networks. An exemplary evolved network is depicted in figure 2. Table 5 presents the results for a fully connected $2-18-2$ GMLP (76 connections, sigmoidal activation) trained by Rprop for 5000 epochs.

Considering the mean test set error, two of the evolutionary approaches (*single*, *multi–bit*) generate better results than conventional training. When looking at the test set errors of the best networks, all evolution products are superior to the trained network. It is known that good ANN performance for the Two Spirals problem is only achievable with specific architectures or training regimes. In [16]

we could achieve a test set error of 6.73% when evolving network structure, activation functions, and Rprop learning parameters. Though, these networks were already considered to be small, they had around 12 hidden neurons and 80 connections, which is roughly twice the complexity of the evolved networks above.

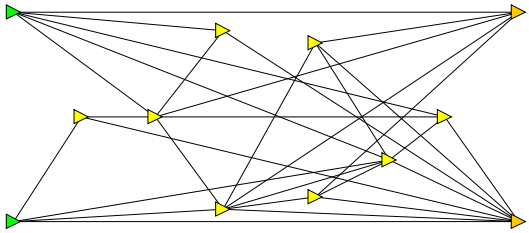

Figure 2: An evolved Two Spirals network.

	Avg	StdDev	Best	Worst
Training	37.58	2.93	32.99	43.81
Test	38.3	3.05	35.05	44.85

Table 5: Two Spirals Training Statistics (10 runs).

4.3 Mackey–Glass Results

Finally, table 6 presents the results for the Mackey–Glass time series prediction. Here, the *Normalized Root Mean Square Error* (NRMSE) is used to assess the quality of the networks. The *Neuron* values are, again, the hidden neuron counts.

Best Net	*single*	*multi–bit*	*multi–real*
Train error	0.49004	0.50072	**0.45316**
Test error	0.48969	0.48709	**0.43635**
Neurons	12	10	**6**
Links	52	43	**24**
Means			
Train error	2.06491	4.24671	**0.66954**
StdDev.	2.74516	5.23194	0.51552
Test error	2.08859	4.29186	**0.65979**
StdDev.	2.80059	5.31418	0.51504
Neurons	9.33	9.33	**8.43**
StdDev.	1.85	2.64	2.03
Links	43.2	44.3	**37.57**
StdDev.	10.66	16	12.41

Table 6: Mackey–Glass Evolution Statistics (30 runs).

In this example the *multi–real* representation clearly outperforms the others. Actually, the mean test errors for *single* and *multi–bit* indicate difficulties of the evolutionary search to find reasonable network solutions in every run. Sometimes evolution is quite successful, as indicated by the

	Avg	StdDev	Best	Worst
Training	0.3901	0.03668	0.3249	0.441
Test	0.3979	0.03452	0.3392	0.4464

Table 7: Mackey–Glass Training Statistics (10 runs).

quality of the best networks, but in some runs it fails to find meaningful solutions (NRMSE < 1.0). This behavior can most likely be attributed to the specific importance of a single network parameter: the bias value of the linear output neuron giving the prediction value. While with *multi–real* this bias is constantly mutated in small steps, the chance of mutation is small with the other representations. Hence, the bit representations rely more on the quality of networks in the random start population than *multi–real*. The differences in network complexity are small with *multi–real* generating a bit smaller networks. Table 7 presents the results training a $4-9-9-1$ MLP (126 connections, sigmoidal activation functions) for 500 cycles using Rprop.

The trained networks clearly exceed the quality of the evolved networks. We believe that the existence of a single very important parameter in the network is the reason for these differences, because analytical training methods can intrinsically discriminate between more or less influential parameters to be adapted. In [17] the best result achieved with coevolutionary techniques is 0.1868, and in [18] a neural system based on a *Long Short Term Memory* (LSTM) produced a minimal NRMSE of 0.26. However, in both of the latter approaches conventional training is part of the model generation.

5 Conclusion

We have presented experiments investigating the impact of multi–chromosomal representations on the performance of evolved artificial neural networks (ANNs). A technical benefit of the multi–chromosomal representation is the ability to use different encodings, genetic operators, and parameters for specific chromosomes. On three benchmark problems (two classification, one prediction) we observed that even though ANN structure *and* weights were evolved in a rather small number of 1,000 generations, network performance in terms of test set error can be superior to non–evolutionary approaches including conventional ANN training in specific problem instances (Glass). However, our hypothesis that shuffling of chromosomes in multi–chromosomal representations induces improved generalization of the evolved networks could not be confirmed. In fact, on the three benchmark problems each of the three investigated representations turned out to be the winner once.

A key factor influencing the performance of the evolved networks could be the partitioning of genetic code onto specific chromosomes. Here, we determined the number and organization of chromosomes in a straight–forward manner (e.g., all links are encoded on a single chromo-

some), but in nature it is common that cooperating genes are located on different chromosomes [6]. One could argue that our choice of gene distribution helps in evolving good networks for the specific Glass problem, but is less productive for the other problems investigated. Hence, a focus of future work will be on the difficult problem of self–organization of the chromosome structure. This could be achieved by specific encodings, where each network parameter has an additional tag (code) identifying its category (e.g., weight or neuron), so that the parameters may be arbitrarily dispersed on chromosomes. Similar ideas could be used to not only evolve the distribution of genes on a (fixed) number of chromosomes, but also the number and size of chromosomes.

References

[1] Markus Spitzlinger and Helmut A. Mayer. Evolution of Fuzzy Controllers with Multi–Chromosomal Representation of Membership Functions. In *Proceedings of the 10th IFSA World Congress*, pages 417–420, June 2003.

[2] Kenneth O. Stanley and Risto Miikkulainen. Efficient Reinforcement Learning through Evolving Neural Network Topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference 2002*, San Francisco, CA, 2002. Morgan Kaufmann.

[3] X. Yao and Y. Liu. A New Evolutionary System for Evolving Artificial Neural Networks. *IEEE Transactions on Neural Networks*, 8(3):694–713, May 1997.

[4] Rachel Cavill, Steve Smith, and Andy Tyrrell. Multi–chromosomal genetic programming. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pages 1753–1759. ACM Press, June 2005.

[5] Helmut A. Mayer and Markus Spitzlinger. Multi–Chromosomal Representations and Chromosome Shuffling in Evolutionary Algorithms. In *2003 Congress on Evolutionary Computation*, pages 1145–1149. IEEE, December 2003.

[6] Harvey Lodish, David Baltimore, Arnold Berk, S. Lawrence Zipursky, Paul Matsudaira, and James Darnell. *Molecular Cell Biology*. Scientific American Books, 3rd edition, 1995. ISBN 07167-2380-8.

[7] R. Hinterding and K. Juliff. A Genetic Algorithm for Stock Cutting: An Exploration of Mapping Schemes. Technical Report 24COMP3, Department of Computer and Mathematical Sciences, Victoria University of Technology, Melbourne, Australia, 1993.

[8] Simon Ronald, Steve Kirkby, and Peter Eklund. Multi–chromosome Mixed Encodings for Heterogeneous Problems. In *Proceedings of the 4th IEEE International Conference on Evolutionary Computation*, pages 37–42. IEEE Press, 1997.

[9] A. K. Bhatia and Sandip K. Basu. Packing Bins Using Multi–chromosomal Genetic Representation and Better-Fit Heuristic. In *Proceedings of the International Conference on Neural Information Processing*, pages 181–186, 2004.

[10] H.-P. Schwefel. *Evolution and Optimum Seeking*. Sixth–Generation Computer Technology Series. Wiley, New York, 1995.

[11] 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.

[12] Matt White et al. CMU neural networks benchmark collection. Available from *http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/neural/bench/cmu/*, 1996.

[13] M. Mackey and L. Glass. Oscillation and Chaos in Physiological Control Systems. *Science*, 197(287), 1977.

[14] 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.

[15] M. Islam, X. Yao, and K. Murase. A constructive algorithm for training cooperative neural network ensembles. *IEEE Transactions on Neural Networks*, 14(4):820–834, July 2003.

[16] Helmut A. Mayer and Roland Schwaiger. Differentiation of Neuron Types by Evolving Activation Function Templates for Artificial Neural Networks. In *Proceedings of the 2002 World Congress on Computational Intelligence, International Joint Conference on Neural Networks*, pages 1773–1778. IEEE, May 2002.

[17] Helmut A. Mayer and Roland Schwaiger. Evolutionary and Coevolutionary Approaches to Time Series Prediction Using Generalized Multi-Layer Perceptrons. In *Congress on Evolutionary Computation*, pages 275–280. IEEE, July 1999.

[18] Felix A. Gers, Douglas Eck, and Jürgen Schmidhuber. Applying LSTM to Time Series Predictable through Time–Window Approaches. In Georg Dorffner, Horst Bischof, and Kurt Hornik, editors, *Proceedings International Conference on Artificial Neural Networks (Lecture Notes in Computer Science)*, volume 2130, pages 669–676. Springer, August 2001.