A Tool for Evolving Artificial Neural Networks

Efstratios F. Georgopoulos[1, 3], Adam V. Adamopoulos[2, 3] and Spiridon D. Likothanassis[3]

Abstract. A hybrid evolutionary algorithm that combines genetic programming philosophy, with localized Extended Kalman Filter (EKF) training method is presented here. This algorithm is used for the topological evolution and training of Multi-Layered Neural Networks. It is implemented as a visual software tool in C++ programming language. The proposed hybrid evolutionary algorithm is applied on two bio-signal modeling tasks: the Magneto Encephalogram (MEG) of epileptic patients and the Magneto Cardiogram (MCG) of normal subjects, exhibiting very satisfactory results.

1 INTRODUCTION

One of the main problems that are faced when Artificial Neural Networks (ANN) and especially Multilayer Perceptrons, are applied on some tasks, is finding the network architecture or topology that is best suited for the task at hand. A small network for the problem might causes poor learning ability, while a large one might cause poor generalization. Until now the common method to determine the architecture of a neural network is by trial and error. However, in the last years there have been many attempts, in the direction of designing the architecture of a neural network automatically, that have led to a variety of methods.

Two subcategories of such methods are a) the constructive and b) pruning (destructive) algorithms, [28], [29]. Roughly speaking, a constructive algorithm starts with a minimal network, that is an ANN with a minimal number of hidden layers, hidden neurons and connections, and adds new layers, neurons or connections, if it is necessary, during the training phase. On the opposite, a pruning (destructive) algorithm does the opposite, starts with a maximal network and deletes the unnecessary layers, nodes and connections during training.

Another approach to this problem is by using Genetic Algorithms. Genetic Algorithms are a class of optimization algorithms, which are good in exploring a large and complex space in an intelligent way in order to find values close to the global optimum (see [12], [15], [20] and [22] for details). The design of a near optimal topology can be formulated as a search problem in the architecture space, where each point in the space represents network architecture. The training can be formulated as a search problem in the weight space. Since the end of the last decade, there have been several attempts to combine the technology of neural networks with that of genetic algorithms. Given some performance criteria, for example error, generalization ability, learning time, architectural complexity etc, for the architecture, the performance level of all architectures forms a surface in the space. The optimal architecture design equals to finding the optimum point on this surface.

The first attempts, described in [10], [23], [25] and [27], focused mainly on the problem of training the networks and not in the topology design. They used neural networks with fixed architecture and genetic algorithms in order to search the weight space for some near optimum weight vector that solves the problem of network training. That is, they used genetic algorithms instead of some classical training algorithm. Soon the main research interest moved from the training, to the search for the optimal architectural (or topological) design of a neural network. Some first works used genetic algorithms in order to imitate the pruning algorithms. They start with a network larger than necessary for the task and then use a specially designed genetic algorithm to define which combination of connections is sufficient to, quickly and accurately, learn to perform the target task, using back propagation. Miller et al. [21] did that for some small nets. The same problem, but for larger networks, was faced by Whitley and Bogard in [26]. Bornholdt and Graudenz in [9], used a modified GA in order to evolve a simplified model of a biological neural network and then applied the algorithm to some toy Boolean functions. A different approach to the design and evolution of modular neural network architectures is presented in [13]. Billings and Zheng in [8] used a GA for the architectural evolution of radial basis function (RBF) networks. The most recent approach and maybe the most successful one, to the problem of finding the near optimum architecture is presented in [28]. There, Yao and Liu propose a new evolutionary system, the EPNet, for evolving artificial neural networks' behavior.

The last couple of years, there is an increasing interest in the use of multi-objective optimization methods and especially evolutionary multi-objective techniques for neural network training and structure optimization. Two very interesting approaches are presented in [31] and [32].

The present work is the sequence of a series of efforts concerning the application of evolutionary algorithms for the optimization of neural networks. In [17] a neural network model with binary neurons was evolved by a modified genetic algorithm in order to learn some Boolean functions. In [1], [2], [3], [4], [5], [6], [7], [11], [18] and [19] genetically evolved artificial neural networks were successfully used for a variety of problems.

In this paper we present a hybrid evolutionary method that looks like more to a genetic programming technique for the evolution of a population of feed-forward Multi Layered Perceptrons [14]. This hybrid algorithm combines a genetic programming technique (for details see [16]) for the evolution of the architecture of a neural network, with a training method based on the localized Extended

[1] Technological Educational Institute of Kalamata, Greece, e-mail: sfg@teikal.gr
[2] Dept. of Medicine, Democritus University of Thrace, Greece, e-mail: adam@med.duth.gr
[3] Dept. of Computer Engineering & Informatics, University of Patras, Greece, e-mail: likothan@cti.gr

Kalman Filter (EKF), known as Multiple Extended Kalman Algorithm (MEKA). The MEKA is described in detail in [24]. The novelty of this effort depends on, apart from the combination of evolution techniques with MEKA, the capability of the proposed method to search, not only for the optimal number of hidden units, but also, for the number of inputs needed for the problem at hand; of course this stands only for time series prediction problems where the number of needed past values, which represent the network's inputs, is unknown. This hybrid algorithm is an evolved and heavily enriched version of an older algorithm that was developed by the authors and presented in [4], [7] and [19]. Furthermore this evolutionary neural network system has been implemented as a visual tool in C++ with a graphical user interface. In order to test the ability of this algorithm to produce networks that perform well, we apply the system on two biosignals, namely the Magneto Encephalogram (MEG) recordings of epileptic patients and Magneto Cardiogram (MCG) of normal subjects. The algorithm produces networks with small sizes that perform well.

The rest of the paper is organized as follows. Section 2 describes the hybrid evolutionary algorithm, while the numerical experiments are presented in section 3. Finally, section 4 discusses the concluding remarks.

2 THE HYBRID EVOLUTIONARY ALGORITHM

2.1 THE MULTIPLE EXTENDED KALMAN ALGORITHM - MEKA

Consider a network characterized by a weight vector w. The average cost function that should be minimized during the training phase is defined in terms of *N* input-output patterns as follows:

$$E_{av}(w) = \frac{1}{2N}\sum_{n=1}^{N}\sum_{j\in C}\left[d_j(n) - y_j(n)\right]^2 \tag{1}$$

Where $d_j(n)$ is the desired response and $y_j(n)$ the actual response of output neuron *j* when input pattern *n* is presented, while the set *C* includes all the output neurons of the network. The cost function $E_{av}(w)$ depends on the weight vector *w* due to the fact that $y_j(n)$ itself depends on *w*.

Concentrating on an arbitrary neuron *i*, which might be located anywhere in the network, its behavior during the training phase may be viewed as a non-linear dynamic system, which in the context of Kalman filter theory may be described by the following state-measurement equations [14], [24]:

$$w_i(n+1) = w_i(n) \tag{2}$$

$$d_i(n) = y_i(n) + e_i(n) \tag{3}$$

$$y_i(n) = \varphi\left(x_i^T(n), w_i(n)\right) \tag{4}$$

Where the iteration n corresponds to the presentation of the nth input pattern, $x_i(n)$ and $y_i(n)$ are the input and output vector of neuron i respectively and $e_i(n)$ is the measurement error at the output of neuron i, the instantaneous estimate of which is given by:

$$e_i(n) = -\frac{\partial E(n)}{\partial y_i(n)} \tag{5}$$

$$E(n) = \frac{1}{2}\sum_{j\in C}\left[d_j(n) - y_j(n)\right]^2 \tag{6}$$

The differentiation in equation (5) corresponds to the back-propagation of the global error to the output of neuron *i*. The activation function $\varphi(\bullet)$ is responsible for the non-linearity in the neuron. The weight vector w_i of the optimum model for neuron *i* is to be "estimated" through training with examples. The activation function is assumed to be differentiable. Accordingly, we can use Taylor series to expand equation (3) about the current estimate of the weight vector and thereby linearize the equation as follows [14]:

$$\phi\left(x_i^T(n)w_i(n)\right) \cong q_i^T(n)w_i(n) + \left[\phi\left(x_i^T(n)\hat{w}_i(n)\right) - q_i^T(n)\hat{w}_i(n)\right] \tag{7}$$

where

$$q_i(n) = \left[\frac{\partial\phi\left(x_i^T(n)w_i(n)\right)}{\partial w_{i(n)}}\right]_{w_i(n)=\hat{w}_i(n)} = \hat{y}_i(n)\left[1-\hat{y}_i(n)\right]x_i(n) \tag{8}$$

$\hat{y}_i(n)$ is the output of neuron *i* that results from the use of the weight estimate. In equation (8) we have assumed the use of the logistic function; other sigmoid functions, like the hyperbolic tangent, can be used as well. The first term of the right hand side of equation (7) is the desired linear term while the remaining term represents a modeling error. Thus substituting equation (7) and (4) in (3) and ignoring the modeling error we obtain:

$$d_i(n) = q_i^T(n)w_i(n) + e_i(n) \tag{9}$$

Where $e_i(n)$ and $q_i(n)$ are defined in equations (5) and (8) respectively.

Equations (2) and (9) describe the linearized behavior of neuron *i*. Given the pair of equations (2) and (9), we can make use of the standard Recursive Least Squares (RLS) algorithm equations [14], which is a special case of the Kalman filter, to make an estimate of the weight vector $w_i(n)$ of neuron *i*. The resulting solution is defined by the following system of recursive equations [14] that describe the Multiple Extended Kalman Algorithm (MEKA) [24]:

$$r_i(n) = (\lambda - 1)\cdot P_i(n-1)\cdot q_i(n) \tag{10}$$

$$k_i(n) = r_i(n)\cdot\left(1 + r_i^T(n)\cdot q_i(n)\right) - 1 \tag{11}$$

$$w_i(n+1) = w_i(n) + e_i(n)\cdot k_i(n) \tag{12}$$

$$P_i(n+1) = (\lambda - 1)\cdot P_i(n) - k_i(n)\cdot r_i^T(n) \tag{13}$$

Where, *n=1,…,N* is the iteration number and *N* is the total number of examples.

The vector $q_i(n)$ represents the linearized neuron activation function given in equation (6), $P_i(n)$ is the current estimate of the inverse of the covariance matrix of $q_i(n)$ and $k_i(n)$ is the Kalman gain. The parameter λ is a forgetting factor which takes values in the range (0,1], and $e_i(n)$ is the localized measure of the global error. Equation (13) is called the Riccatti difference equation.

Each neuron in the network perceives its own effective input $q_i(n)$, hence it has to maintain its own copy of $P_i(n)$ even in the case in which it may share some of its inputs with other neurons in the network.

2.2 THE EVOLUTIONARY ALGORITHM

The proposed evolutionary algorithm is an improved version of a modified genetic algorithm that was used aforetime by the authors. It maintains the basic working philosophy of evolutionary algorithms and resembles genetic programming (see [16] for details) since it evolves complicated structures like linked lists and not simple bit strings as genetic algorithms do.

The algorithm evolves, using a number of genetic operators, a population of artificial neural networks (multilayered perceptrons) that are represented as linked lists of network layers and neurons; thus it is used the direct encoding scheme. The basic steps of the algorithm are as follow:

1. <u>Initialization</u>: An initial population of neural networks (called individuals) is created. Every individual has a random number of neurons (or nodes) and connections (synapses). The connection weights are initialized to some random values within a specific range.

2. <u>Training</u>: Every individual (neural network) in the population is trained using MEKA for a small number of training epochs. For populations other than initial, training occurs only for those networks that have been changed by the application of genetic operators.

3. <u>Fitness Evaluation:</u> As fitness function it is used a function that combines the performance of the network in the training and/or validation set with the size of the network. The performance is evaluated using the Mean Squared Error (MSE) or the Mean Relative Error (MRE). While, the size is the number of neurons and/or the number of active synapses in the network. So the fitness function for the case of MRE has a formula of the type:

$$Fitness(i) = \frac{1}{1 + MRE(i) + sp \cdot MRE(i) \cdot SIZE(i)} \tag{14}$$

   Where *sp* is a parameter that controls the weight of the network size in the evaluation of fitness, *MRE(i)* is the value of MRE of individual *i*, *SIZE(i)* is the size of individual i which can be calculated as the number of active connections or the number of neurons and *i* is an index taking values in the range 1 to population size.

4. <u>Selection</u>: Selection operator is been used in order to create a new, intermediate, population from the old one, by selecting individuals based on their fitness. This can be done using any of the following three different selection schemes that have been implemented, namely:
   - The Elitism Roulette Wheel Selection Operator, with variable elitist pressure (for more details see [12], [16], [20] and [22]).
   - The Rank Based Selection (for more details see [12], [16], [20] and [22]).
   - The Tournament Selection with variable tournament size (for more details see [12], [16], [20] and [22]).

5. <u>*Mutation*</u>*:* It works on the members of Three different mutation operators are implemented:
   - *Input Mutation:* it selects randomly a neural network from the population and changes its number of inputs. This operator works only on time series modeling and prediction problems, where the number of past values (network inputs) needed to predict future values is not usually known a priory.
   - *Hidden mutation:* it selects randomly a neural network from the population and changes the structure of its hidden region by adding or deleting a random number (selected uniformly from a given interval) of hidden neurons.
   - *Non Uniform Weight mutation:* it is responsible for the fine tuning capabilities of the system. It selects randomly a number of connection weights and changes their values to new ones as follows: Let suppose that w is the old weight value then the new one is given by the formula:

   $$w(n+1) = w(n) \pm \Delta w(t, ub - w(n)) \tag{15}$$

   Where *lb* and *ub* are the lower and upper bounds of the weight values, *t* is the generation number, and *Δ(t,y)* is a function that returns a value in the range [0,y], such that the probability of *Δ(t,y)* being close to 0 increases as t increases. This property causes this operator to search the solution space initially uniformly (while t is small) and very locally at the later stages. In our experiments the following function, [20] was used:

   $$\Delta(t, y) = y \cdot \left(1 - r^{\left(1 - t/T\right)^{b}}\right) \tag{16}$$

   Where *r* is a random number on [0,1], *T* is the maximal generation number (a parameter of the algorithm), and *b* is a system parameter determining the degree of non-uniformity.

   - *Gaussian weight mutation:* it works like the *Non Uniform Weight mutation* operator with the difference that the new weight value is calculated by the formula:

   $$w(n+1) = w(n) + \Delta w(n) \tag{17}$$

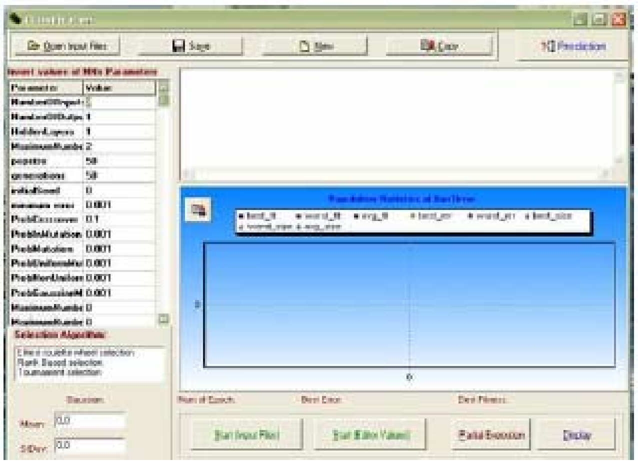
   Where, *Δw* is a small random number following Gaussian distribution.
   - *Uniform weight mutation: it works like the Gaussian mutation operator with the difference that, Δw is a small random number following Uniform distribu*tion.

6. <u>*Crossover*</u>*:* It selects two parents (neural networks) and generates one or two offspring by recombining parts of them. The offspring take the place of their parents in the new population. In the presented algorithm crossover recombines whole neurons with their incoming connections. But since we have to deal with networks with different structures, the new connections that might have to be produced are initialized with random weight values as in the initialization phase. Herein crossover works more like a mutation operator, like in most genetic programming systems, than as the recombination operator of genetic algorithms

Therefore the presented hybrid evolutionary algorithm works in brief as follows: it starts with a population of randomly constructed Neural Networks (step 1). Networks undergo some training for a couple of epochs with MEKA, using the training set (step 2). Performance is measured with the fitness function (step 3) using the validation set, in order to improve generalization. Then a new, intermediate, population is created, by selecting the more fit individuals according to their fitness (step 4) using any of the three selection schemes. Some members of this intermediate population undergo transformations by means of genetic operators to form the members (individuals) of the new population: mutation (step 5) and crossover (step 6) operators. The new population that is created is trained again (step 2); new members are trained for a couple of epochs, while the members that have survived and passed from the old population may be trained with MEKA for some more epochs, or may not be trained at all. This is the new generation. This whole process continues until a predefined termination condition is fulfilled; the termination condition might be a maximum number of generation or a minimum error (MSE or MRE) value. Once terminated the algorithm is expected to have reached a near-optimum solution, i.e. a trained network with near optimum architecture.

2.3 THE TOOL

This hybrid evolutionary algorithm has been implemented as a visual tool in C++ programming language, having a graphical user interface (GUI). Specifically, it was used the Borland C++ version 6.0 IDE for Windows. Figure 1 and 2 depict two of the basic forms of the program, for the two main categories of problems that it can be used for, classification and time series prediction. Figure 3 is the "statistics" form that illustrates the evolutionary process and prints useful information about it.



Insert values of NNs Parameters

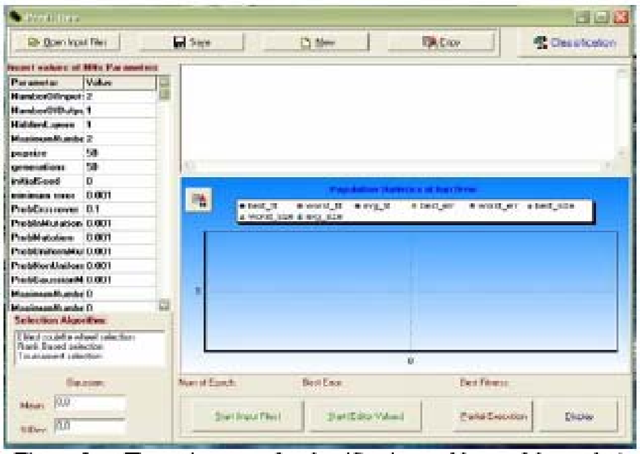
Parameter	Value
NumberOfInputs	2
NumberOfOutputs	1
HiddenLayers	1
MaximumNumbe	2
popsize	50
generations	50
initialSeed	0
minimum error	0.001
ProbCrossover	0.1
ProbInMutation	0.001
ProbMutation	0.001
ProbUniformMu	0.001
ProbNonUnifor	0.001
ProbGaussianM	0.001
MaximumNumbe	0
MaximumNumbe	0

Selection Algorithm:
Eliteist roulette wheel selection
Rank Based selection
Tournament selection

Gaussian:
Mean: 0,0
StDev: 0,0

Num of Epoch:
Best Error:
Best Fitness:

Figure 1. The main form for classification problems of the evolutionary neural network system



Insert values of NNs Parameters

Parameter	Value
NumberOfInputs	2
NumberOfOutputs	1
HiddenLayers	1
MaximumNumbe	2
popsize	50
generations	50
initialSeed	0
minimum error	0.001
ProbCrossover	0.1
ProbInMutation	0.001
ProbMutation	0.001
ProbUniformMu	0.001
ProbNonUnifor	0.001
ProbGaussianM	0.001
MaximumNumbe	0
MaximumNumbe	0

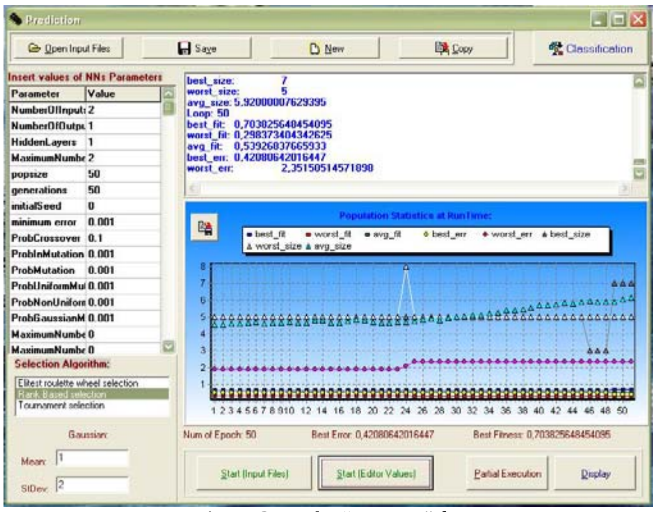
Selection Algorithm:
Eliteist roulette wheel selection
Rank Based selection
Tournament selection

Gaussian:
Mean: 0,0
StDev: 0,0

Num of Epoch:
Best Error:
Best Fitness:

Figure 2. The main form for prediction problems of the evolutionary neural network system

The user can select between this two problem categories. Then he/she can insert the values of the various genetic parameters, the training, validation and test files, as well as the output log files. The user can observe the evolutionary process using some real time graphical display of the error, the performance of the best ever network and other parameters.



Insert values of NNs Parameters

Parameter	Value
NumberOfInputs	2
NumberOfOutputs	1
HiddenLayers	1
MaximumNumbe	2
popsize	50
generations	50
initialSeed	0
minimum error	0.001
ProbCrossover	0.1
ProbInMutation	0.001
ProbMutation	0.001
ProbUniformMu	0.001
ProbNonUnifor	0.001
ProbGaussianM	0.001
MaximumNumbe	0
MaximumNumbe	0

```
best_size:        7
worst_size:       5
avg_size: 5.92000007629395
Loop: 50
best_fit:  0,703025640454095
worst_fit: 0,298373404342625
avg_fit:   0,53926037665933
best_err:  0,42080642016447
worst_err:           2,35150514571898
```

Selection Algorithm:
Eliteist roulette wheel selection
Rank Based selection
Tournament selection

Gaussian:
Mean: 1
StDev: 2

Num of Epoch: 50
Best Error: 0,42080642016447
Best Fitness: 0,703825648454095

Figure 3. The "statistics" form

3 NUMERICAL EXPERIMENTS

In order to examine the ability of the algorithm to produce networks that learn and generalize well we have tested it on two real world problems: the modeling of the MEG recordings of epileptic patients and the modeling MCG recordings of normal subjects.

Brain dynamics can be evaluated by recording the changes of the neuronal electric voltage, either by the electroencephalogram (EEG), or by the MEG. The EEG recordings represent the time series that match up to neurological activity as a function of time. On the other hand the MEG is generated due to the time varying nature of the neuronal electric activity, since time-varying electric currents generate magnetic fields. EEG and MEG are considered to be complementary, each one carrying a part but not the whole of the information related to the underlying neural activity. Thus, it has been suggested that the EEG is mostly related to the inter-neural electric activity, whereas the MEG is mostly related to the intra-neural activity. The MEG recordings of epileptic patients were obtained using a Super-conductive QUantum Interference Device (SQUID) and were digitized with a sampling frequency of 256Hz using a 12-bit A/D Converter. SQUID is a very sensitive magnetometer, capable to detect and record the bio-magnetic fields produced in the human brain due to the generation of electrical micro-currents at neural cellular level [30].

The same stands for the MCG recordings which are magnetic recordings of the heart operation of normal subjects. MEG and MCG data were provided by the Laboratory of Medical Physics of the Democritus University of Thrace, Greece, where a one-channel DC SQUID is operable. Both biosignal data were normalized in the interval [0,1] in order to be processed by the neural networks.

In all the experiments we used, for comparison reasons, the same parameter values, which are depicted in table 1. For the case of the MEG modeling, as training set where used 1024 data samples (corresponding to a four seconds epoch of the MEG) while for the testing was used 512 data samples (corresponding to a two seconds epoch of the MEG). For the case of the MCG modeling, as training set where used 1024 data samples and for the test set was used 1024 data samples The algorithm was left to run over 1000 generations.

In order to evaluate the forecasting capability of the produced networks we used three well-known error measures, the Normalized Root Mean Squared Error (NRMSE), the Correlation Coefficient (CC) and the Mean Relative Error (MRE). The performance of the hybrid algorithm for the case of MEG modeling is depicted in tables 2 and 3 and in figure 4, while for the case of MCG in tables 4 and 5 and in figure 5.

Table 1. Parameters used for the cases of MEG and MCG modeling

Parameter	*Value*
Population	50
Max number of Generations	1000
Crossover Probability	0,1
Input Mutation Probability	0,1
Weight Mutation Probability	0,1
Uniform Mutation Probability	0,1
nonUniform Mutation Probability	0,1
Gaussian Mutation Probability	0,1
MeanGaussian	0,1
StDev Gauss	0,25
Predicting Horizon	1

Table 2. MEG forecasting - Errors on the Training Set

Architecture	NRMSE	C.C.	MRE
4-9-1	0.2540	0.9674	0.0350
3-3-1	0.2913	0.9581	0.0386
4-5-1	0.2564	0.9672	0.0357
3-2-1	0.2967	0.9557	0.0411
3-3-1	0.2705	0.9656	0.0369
3-4-1	0.2655	0.9645	0.0361

Table 3. MEG forecasting - Errors on the Test Set

Architecture	NRMSE	C.C.	MRE
4-9-1	0.1971	0.9805	0.0403
3-3-1	0.2189	0.9757	0.0438
4-5-1	0.2063	0.9786	0.0434
3-2-1	0.2309	0.9733	0.0466
3-3-1	0.2177	0.9765	0.0446
3-4-1	0.2111	0.9775	0.0429

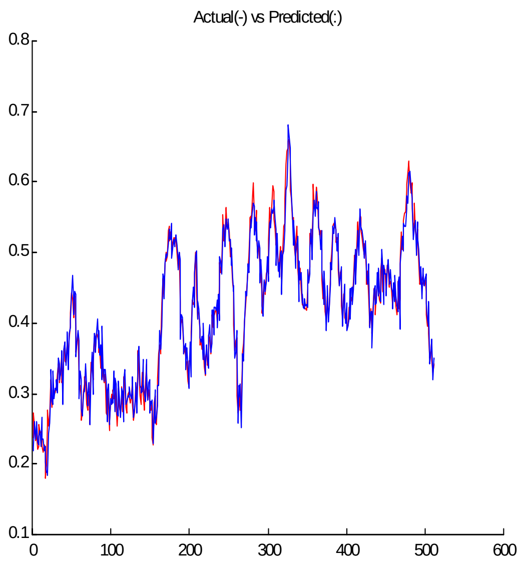


Figure 4. MEG forecasting, performance on the test set.

Table 4. MCG forecasting - Errors on the Training Set

Architecture	NRMSE	C.C.	MRE
12-10-1	0.1918	0.9815	0.8421
9-20-1	0.1840	0.9829	0.8073
10-11-1	0.1790	0.9839	0.6790
13-10-1	0.1869	0.9824	0.7875
13-12-1	0.2213	0.9761	0.8983
10-9-1	0.2274	0.9740	0.9702
4-4-1	0.3593	0.9441	0.9822
3-1-1	0.6481	0.8262	0.9061

Table 5. MCG forecasting - Errors on the Test Set

Architecture	NRMSE	C.C.	MRE
12-10-1	0.2419	0.9704	1.4222
9-20-1	0.2081	0.9781	1.1874
10-11-1	0.2214	0.9752	1.3642
13-10-1	0.1858	0.9827	0.9617
13-12-1	0.2163	0.9774	0.8677
10-9-1	0.2193	0.9760	0.9882
4-4-1	0.3586	0.9445	0.9817
3-1-1	0.6497	0.8241	0.9915

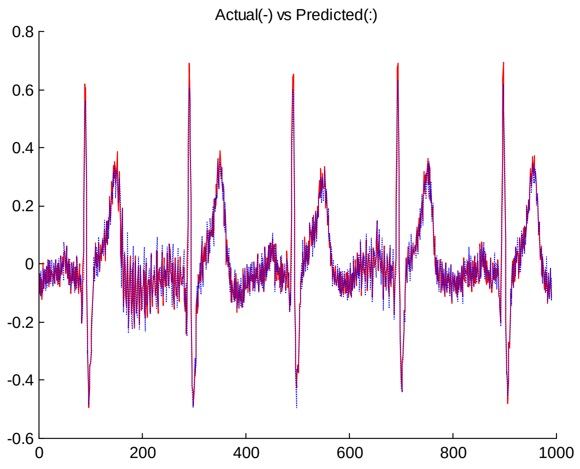


Figure 5. MCG forecasting, performance on the test set.

4 CONCLUSIONS

In the current paper it was presented a hybrid biological inspired evolutionary algorithm that combines a genetic programming technique with a training method based on the Multiple Extended Kalman Algorithm. This hybrid algorithm is implemented in C++ as a software system with a graphical user interface.

The main novelties of the proposed hybrid algorithm are the combination of genetic programming technique with MEKA, the use of a fitness function that combines performance with network size, the ability to evolve not only the structure of the hidden layers but the number of inputs as well, and the large number of different genetic operators and especially mutation operators that have been implemented. Another novelty is the representation used for neural networks. As said before, every network in the population is represented as a link list of layers and neurons, using the direct encoding scheme. The use of link lists has some certain advantages that have to do mainly with the memory management; you use only the memory that is needed every time and you don’t have to allocate a maximum memory size, for maximum network size like other representation schemes. Moreover link lists are dynamic data structures, which it means that the neural network architecture can change dynamically during run time in contrast with other data structures like matrices that in C++ can not change during run time.

This hybrid algorithm was used for the modeling of two biological time series, namely the Magneto Encephalogram (MEG) recordings of epileptic patients and Magneto Cardiogram (MCG) of normal subjects. All the reported cases refer to predictions on recordings of the dynamics of nonlinear systems. In all the performed experiments the algorithm was able to find a near optimum network architecture that gave small prediction errors. Therefore we can conclude that the algorithm is able to produce small and compact networks that learn and generalize well.

The algorithm has only tested on time series prediction problems and it is in our intention to test it on some difficult classification problems as well.

One of the main drawbacks of this kind of algorithms, namely the evolutionary algorithms, hybrid or not, is that they are computational expensive in terms of computer memory and CPU time. Even though the proposed algorithm belongs to this category, the use of MEKA for just a couple of epochs for the training phase of the neural networks and the representation where each member of the population is a network represented as a link list so that there is no need to use encoding and decoding functions for the calculation of network's performance, makes the algorithm less computational expensive than other approaches to the same problem of neural networks evolution.

The algorithm could be further improved by adding some more genetic operators for better and faster local search both to the architecture and weight space and this is going to be one of our future research targets. Furthermore, in the integrated software system there are already implemented a large number of genetic operators whose influence to the performance of the hybrid algorithm needs to be appraised; we need to see which of the three selection schemes, or the many mutation operators give better results. Another future research direction will be the combination of MEKA with other evolutionary techniques like Particle Swarm Optimization and Differential Evolution for neural network evolution.

REFERENCES

[1] Adamopoulos, A., Andreou, A., Georgopoulos, E., Ioannou, N. and Likothanassis, S., "Currency Forecasting Using Recurrently RBF Networks Optimized by Genetic Algorithms", Computational Finance 1997 (CF'97), London Business School, 1997.

[2] Adamopoulos, A., Anninos, P., Likothanassis, S., and Georgopoulos, E. "On the Predictability of MEG of Epileptic Patients Using RBF Networks Evolved with Genetic Algorithms", BIOSIGNAL'98, Brno, Czech Republic, June 23-25, 1998a.

[3] Adamopoulos, A., Georgopoulos E., Manioudakis, G. and Likothanassis, S. "An Evolutionary Method for System Structure Identification Using Neural Networks" *Neural Computation '98,* 1998b.

[4] Adamopoulos, A., G. Georgopoulos, S. Likothanassis and P. Anninos, "Forecasting the MagnetoEngephaloGram (MEG) of Epileptic Patient Using Genetically Optimized Neural Networks", *Genetic and Evolutionary Computation Conference (GECCO'99)*, Orlando, Florida USA, July 14-17, 1999

[5] Andreou, A., Georgopoulos, E., and Likothanassis, S., and Polidoropoulos, P., "Is the Greek foreign exchange-rate market predictable? A comparative study using chaotic dynamics and neural networks", Proceedings of the *Fourth International Conference on Forecasting Financial Markets*, Banque Nationale de Paris and Imperial College, London, 1997.

[6] Andreou, A., Georgopoulos, E., Zombanakis, G. and Likothanassis, S., "Testing Currency Predictability Using An Evolutionary Neural Network Model", Proceedings of the *fifth International Conference on Forecasting Financial Markets*, Banque Nationale de Paris and Imperial College, London, 1998.

[7] Andreou A., Georgopoulos E. and Likothanassis, S. "Exchange Rates Forecasting: A Hybrid Algorithm Based On Genetically Optimized Adaptive Neural Networks", *Computational Economics Journal,* Kluwer Academic Publishers, vol. 20, issue 3, pp. 191 – 210, December 2002

[8] Billings, S. A., and Zheng, G. L. Radial basis function network configuration using genetic algorithms. *Neural Networks*, Vol. 8, pp. 877-890, 1995.

[9] Bornholdt S. and Graudenz, D. General asymmetric neural networks and structure design by genetic algorithms. *Neural Networks*, Vol. 5, pp327 – 334, 1992.

[10] Davis, L. Mapping classifier systems into neural networks. *Proceedings of the 1988 Conference on Neural Information Processing Systems*, Morgan Kaufmann, 1988.

[11] Georgopoulos E., Likothanassis S. and Adamopoulos A., "Evolving Artificial Neural Networks Using Genetic Algorithms", *Neural Network World*, 4/00, pp. 565 – 574, 2000.

[12] Goldberg, D. *Genetic Algorithms in Search Optimization & Machine Learning*, Addison-Wesley 1989.

[13] Happel, B., et al. Design and evolution of modular neural network architectures. *Neural Networks*, Vol. 7, pp. 985 – 1004, 1994.

[14] Haykin, S., "Neural Networks - A Comprehensive Foundation", McMillan College Publishing Company, ch. 6, p.213, New York, 1994.

[15] Holland, J. *Adaptation in Natural and Artificial Systems,* MIT press 1992.

[16] Koza J.R., Genetic programming: on the programming of computers by means of natural selection, MIT Press, Cambridge, MA, 1992.

[17] Likothanassis S. Georgopoulos E. and Fotakis D. (1997). Optimizing the Structure of Neural Networks Using Evolution Techniques*. 5th International Conference on Applications of High - Performance Computers in Engineering*, pp. 157-168, Santiago de Compostela, Spain, July.

[18] Likothanassis, S. D., Georgopoulos, E. F., Manioudakis, G. D. and Adamopoulos, A.V., "Currency Forecasting Using Genetically Optimized Neural Networks", *HERCMA* Athens September 1998.

[19] Likothanassis, S. D., Georgopoulos, E. F. "A Novel Method for the Evolution of Neural Networks", *3rd IMACS/IEEE International Conference on Circuits Systems and Computers (CSC'99)*, July 1999.

[20] Michalewicz, Z., "*Genetic Algorithms + Data Structures = Evolution Programs*", Springer-Verlag, 1996.

[21] Miller, G., et al. Designing neural networks using genetic algorithms. *Proceedings of the 3rd International Conference on Genetic Algorithms*, Morgan Kaufmann 1989.

[22] Mitchell M. *An Introduction to Genetic Algorithms*, MIT Press 1996.

[23] Montana, D. and Davis, L. Training feedforward neural networks using genetic algorithms. *BBN Systems and Technologies*, Cambridge, MA 1989.

[24] Shah, S., Palmieri, F. and Datum, M., "Optimal Filtering Algorithms for Fast Learning in Feed-Forward Neural Networks", *Neural Networks*, Vol. 5, pp. 779-787, 1992.

[25] Whitley, D. Applying genetic algorithms to neural network problems, *International Neural Networks Society*, p.230 1988.

[26] Whitley, D., and Bogart, C. The evolution of connectivity: Pruning neural networks using genetic algorithms. *International Joint Conference on Neural Networks, Washington D.C.,* 1. Hillsdale, NJ: Lawpence Erlbaum, pp. 134-137, 1990.

[27] Whitley, D., and Hanson, T. Optimizing neural networks using faster, more accurate genetic search. *3rd Intern. Conference on Genetic Algorithms, Washington D.C.,* Morgan Kaufmann, pp. 391-396, 1989.

[28] Yao, X. & Liu, Y. A New Evolutionary System for Evolving Artificial Neural Networks, *IEEE Transactions on Neural Networks*, vol. 8, no. 3, 1997.

[29] Yao, X. "Evolving Artificial Neural Networks", *Proceedings of the IEEE*, 87(9):1423:1447, September 1999.

[30] Anninos, P. Jacobson, J. Tsagas, N. Adamopoulos, A. "Spatiotemporal Stationarity of Epileptic Focal Activity Evaluated by Analyzing Magneto Encephalographic (MEG) data and the Theoretical Implications". *Panminerva Med*. 39, 189-201, 1997.

[31] Graning, L.; Yaochu Jin; Sendhoff, B.: Generalization Improvement in Multi-Objective Learning, Neural Networks, IJCNN apos;06. International Joint Conference on Volume , Issue , 0-0 0 Page(s):4839 – 4846, 2006.

[32] Vieira, D. A. G. and J. A. Vasconcelos and W. M. Caminhas: Controlling the parallel layer perceptron complexity using a multiobjective learning algorithm, Neural Computing and Applications, vol. 16, n. 4, p.p. 317—325, 2007.