A New Approach for Incremental Modular Neural Networks in Time Series Forecasting

V. Landassuri-Moreno
School of Computer Science
University of Birmingham
Birmingham, B15 2TT
UK
V.Landassuri-Moreno@cs.bham.ac.uk

John A. Bullinaria
School of Computer Science
University of Birmingham
Birmingham, B15 2TT
UK
J.A.Bullinaria@cs.bham.ac.uk

Abstract

Modular Neural Networks have been used to solve complex problems in a reduced amount of time, to obtain better performance on a range of tasks, and to provide a better understanding of the human brain. This paper presents the first stage of research into Modular Neural Networks where the evolution of them allows an incremental architecture for solving more than one problem. The basis for developing that approach is described in this work. Experimental tests and related issues are presented using an Evolutionary Algorithm called EPNet, obtaining appropriate modules for 20 Time Series Forecasting tasks.

1 Introduction

Artificial Neural Networks (ANNs) have been widely used to solve a range of problems in different fields, with the characteristic that each ANN is focused to solve one problem (Fig. 1). This generally works very well because the network (or module) is specialized to solve only one problem. But there is no fundamental constraint that requires this approach, and it may prove useful to investigate different approaches.

In particular, it is possible to find other approaches where the ANN is set up to solve more than one problem. In this case there exist two main categories: in the first, the network M_1 has n outputs, where each one is focused to solve a particular problem P_i (Fig. 2a); in the second, the network only has one output, but it is focused to solve P_i tasks, i.e. the network is trained with all training sets from each problem (Fig. 2b). If the problems/tasks are similar or closely related in their behaviour, the cross-task interference produced is likely to be minimal, then it is possible to apply successfully the approaches of Fig. 2 [7]. On the other hand, for problems that are not related, the cross-task interference could be considerable. Consequently, it is sometimes better to have independent modules to solve each problem [3]. In fact, even within a single task or problem there may be significantly different behaviours, for example that correspond to different input or time ranges. In such cases, independent modules may be appropriate there too. Then, there could be diverse procedures to solve one problem with Modular Neural Networks (MNNs), e.g. with the Ensemble method [19, 5], or the divide and conquer approach [9]. Fig. 3 illustrates such architectures, with different modules M_i (each an ANN) to solve one problem P_1.

Figure 1: Classical approach: one module or ANN M_1 to solve one problem P_1.

Thus, it is possible to see that the next step with MNNs is to have a bigger representation (a model that grows as the tasks arrive) that allows the reuse of existing modules to solve new problems, e.g. predict the Time Series (TS): Lorenz (problem1), Mackey-Glass (problem 2), and so on. Therefore, this work presents an initial investigation into implementing that idea, describing the basis of the algorithm and the way in which the modules will be obtained to fit in an incremental MNN for the TS Forecasting task. Following previous work on MNNs [3, 12], an approach based on evolution by natural selection

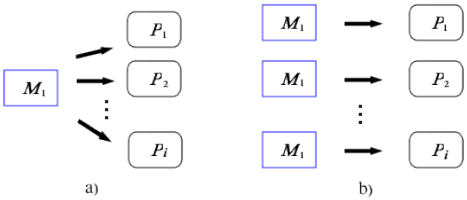

Figure 2: One module or ANN M_1 to solve diverse problems P_i.

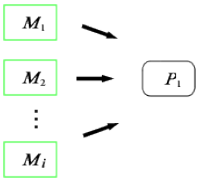

Figure 3: Modular Neural Network (MNN) with many modules M_i to solve one problem P_1.

is used to generate the most efficient systems for the given problems. In this way, this paper presents experimental results from the first stage, where modules are evolved for predicting 20 TS from different fields, with the Multiple-step Forecasting method and a prediction lapse of 30 steps ahead.

The remainder of this paper is organised as follows: In the next section (Sec. 2) is presented the basis for developing an MNN that is able to solve more than one problem, having a compact and incremental architecture. Then (in Sec. 3) the algorithm is described that will be used to evolve the ANN modules and MNNs, with a discussion of some related issues in the evolution of them, taking into account the incremental nature of the MNN. This is followed (in Sec. 4) with the results from the initial series of experiments, and some conclusions (in Sec. 5).

2 Incremental Modular Neural Networks

As noted above, the aim here is to investigate MNNs with bigger representations that allow existing modules to be reused for solving new problems. The algorithm that we propose will involve the following types of modules: 1) Modules able to solve new problems of the same class as existing problems, as presented in [7]; 2) Modules able to collaborate among themselves to solve new problems, in a manner to be discovered by an evolutionary procedure; 3) Modules created to solve a unique new problem, i.e. in cases when the existing modules cannot collaborate among themselves to solve the problem at hand. The evolutionary process aims to ensure that the resultant system will have a compact and incremental architecture.

Fig. 4 illustrates the first (i.e. simplest) approach to develop a MNN with these characteristics, where the super-index j in the modules M_i^j represents the task for which it was created, and the sub-index i represents the number of the module for its original task. In this example, we see that when the kth problem P_k arrives, it uses

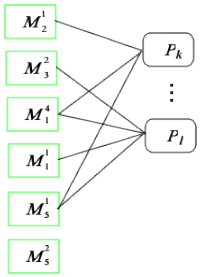

Figure 4: The first MNN proposal.

the existing modules two and five from the task one (M_2^1 and M_5^1), and the existing module one from the task four (M_1^4). For the problem P_l it uses module three from task two (M_3^2), module one from task four (M_1^4) and modules one and five from task one (M_1^1 and M_5^1). If the next problem P_m cannot be solved with the existing modules, a new set of modules $\{M_i^m\}$ will be created for it. Thus, within this representation, it is possible to obtain modules from all the approaches presented in Figures 1 to 3 and then attempt to combine them to find solutions for new tasks/problems as they arise.

Note that we can also go one level deeper, and not only use existing modules (as a whole) to try to solve each new task, but also re-use subsets of neurons within the modules, as presented in Fig. 5. In this example, the modules M_2^1 and M_3^2 already share a neuron – the last one of the first hidden layer from M_2^1. When the problem P_k arrives, it uses one neuron from the output layer of M_2^1 and one neuron from M_3^2. Problem P_l follows a similar procedure. From this configuration it is expected that, at the beginning of the procedure, the existent modules can only solve one task (the classical approach), but as the architecture starts to grow, more modules or neurons could be shared and usefully contribute to solving the new tasks. This configuration is further motivated by its relation to processes found in nature, whereby individuals use existing brain structures to learn to perform new tasks as they appear.

Note that our algorithm discovers new modules for the new tasks as they arrive, which is

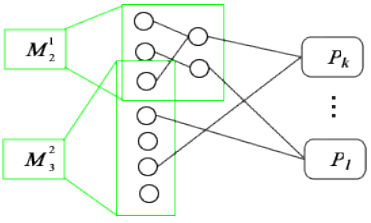

Figure 5: The second MNN proposal.

rather different to finding all the modules in the same evolutionary stage as done by Lui and Yao in their work focussed on solving a single problem [10].

The information presented in this section gives an overview of the general idea for developing this kind of incremental MNN. Existing general knowledge about the operation of neural networks suggests that it should perform well, but to be sure, we need to build an explicit system and test it on real problems. To create a MNN with the characteristics stated above, first we need an Evolutionary Algorithm (EA) to find the compact representation of ANNs (or modules), and then we need to test the performance of what emerges. The next two sections are dedicated to these aspects.

3 The EPNet Algorithm

In this section we present a number of issues and modifications relating to the EA, called EPNet [17, 18], that can be used to evolve ANNs, with the individuals of the last generation employed as modules (via the ensemble method) for the system proposed in the last section.

Since the EPNet algorithm is based upon the standard Evolutionary Programming approach, it presents several advantages over a traditional EA. In particular: it does not use crossover, because useful information in the parent would usually be destroyed by that operator; it uses Lamarkian inheritance, something that is useful for passing information learned by a parent using Modified Back Propagation to the offspring; and during the mutation process, EPNet first measures the importance of a connection before adding or deleting it, as explained in [4].

The original EPNet algorithm proposed by Yao and Lui [18] used a fixed configuration of inputs throughout the whole evolutionary process, which means that we must have existing domain knowledge of the whole problem to determine the required inputs. But since our proposed algorithm will not know which tasks will arrive, it is clearly not possible to use the original EPNet algorithm. Therefore some minimal modifications had to be devised to adjust the algorithm to our requirements. The first change was to leave the inputs specification as another parameter in the algorithm. Then, the existing procedure to add or delete nodes or connections could also be used for the inputs. Another option considered in this case, was to use an automatic procedure to determine the inputs, as explained in the following paragraphs.

Suppose we have a TS defined as a vector $X = [x_1, x_2, \ldots, x_t]$, with its values sampled at regular intervals. Then to predict the value (or point) x_{t+1} we can use a small subset of recent information from our TS. This method is called lagged variables. If we use this, we say that we have an Autoregressive Model and the input space is called the *Embedding Space*. Therefore our TS is transformed into a reconstructed state space using a delay space embedding [14, 2]. Then, it is possible to say that an accurate prediction could be reached using only a finite segment of previous values up to the point to be predicted. Summarizing that, we have:

$$x_{t+1} = F[x_t, x_{t-k}, x_{t-2k}, \ldots, x_{t-(d-1)k}] \quad (1)$$

where d is the number of inputs and k is the time delay. The only condition that needs to be satisfied is that if the attractor is of dimension D, then we must have $d \geq 2D + 1$. But since we know neither D nor the delay, we must use other techniques to find them, e.g. Average Mutual Information for the time delay, and False Nearest Neighbour for the embedded dimension.

For some well known TS, such as the Mackey-Glass, these parameters are already available in the literature, but for most other TS we need to calculate them. Unfortunately, some TS have a rather difficult dynamic and when we attempt to calculate the delay and embedded dimension we can obtain unreliable values. For example, if we apply the Average Mutual Information and the False Nearest Neighbour using the standard package Visual Recurrent Analysis (VRA) [2] for the Daily Morning Gold Prices TS (see Sec. 4) we obtain a delay of 27 and an embedded dimension of 10, and for the Dow Jones TS a delay of 46 and an embedded dimension of 10. In both cases the number of dimensions was limited to ten, so these values could be bigger.

Continuing with describing the modifications required for the EPNet algorithm, we had to use the modified Early Stopping procedure as described in [11]. This is because, for some TS, the validation error does not behave in the smooth manner expected, and consequently the training stage would often stop even if the validation error could be reduced further. For this reason, a procedure to avoid premature stopping was needed, still avoiding the over fitting and maximizing the generalization, but better than the classical Early Stopping approach [11].

The parameters used with EPNet to perform the experiments of the next section were set to: population size 20, generations of evolution 300, initial connection density 80%, initial learning

Table 1: Multiple-step or closed-loop forecasting

Forecast	*Inputs*
y_{t+1}	x_t, x_{t-1}, x_{t-2}
y_{t+2}	y_{t+1}, x_t, x_{t-1}
y_{t+3}	y_{t+2}, y_{t+1}, x_t
y_{t+4}	$y_{t+3}, y_{t+2}, y_{t+1}$

rate 0.25, minimum learning rate 0.1, epochs for learning rate adaptation 5, number of mutated hidden nodes 1, number of mutated connections 1-3, temperatures in SA 5, iterations per temperature in SA 100, stopping after 10 generations of no improvement in the average fitness, 1000 epochs inside the EPNet, and 2000 epochs of further training at the end of the algorithm. All these parameters are convenient traditional values (similar to those used in [17]) and are not intended to be optimal.

Finally, we specify the method used to perform the forecasting, called multiple-step ahead. The input TS X is $[x_1, x_2, \ldots, x_t]$, the number of points ahead to predict is n, the test set is $[x_{t+1}, x_{t+2}, \ldots, x_{t+n}]$, and the forecast in the same interval is $[y_{t+1}, y_{t+2}, \ldots, y_{t+n}]$. Table 1 shows an example in which the network has three inputs and the lapse n to predict is four.

4 Experimental Results

The experiments presented in this section can be seen as the first stage in implementing the incremental MNN as stated in Sec. 2. The Time Series (TS) used in this study belong to different fields, which gives a wide range of different dynamics on which to test our algorithm. The specific TS used were: *Henon, QP2, QP3*, and *Rossler* from [1]; *Ikeda* and *Dow Jones* from [2]; *Logistic* from [6]; *Lorenz* from [16]; *Mackey-Glass* from [15]; Number of daily *Births in Quebec*, Daily closing price of *IBM Stock*, *SP500*, Monthly Flows *Colorado River*, Monthly *Lake Erie* Levels, Daily morning *Gold Prices*, Seismograph (vertical acceleration, nm/sq.sec) of the *Kobe* earthquake, Daily brightness of a variable *Star* and Monthly means of daily relative *Sunspot* numbers from [8]; Santa Fe Competition: *D1* and *Laser* from [13].

Each TS is split into several different data sets for use in the experiments. The first is the training data set; next is the validation set which is used to stop the training early to minimize the over fitting; then there is a test set for use inside EPNet to obtain the fitness of each individual in the population, and finally there is another independent test set on which the best individual found in the evolutionary process is tested to obtain the final performance. For each TS, Table 2 shows the average over ten runs of the Normalized Root Mean Squared Error (NRMS) obtained for the TS with the independent test set. The column "*Min*" shows the NRMS of the best individual found over ten runs. The TS are arranged according to their dynamics: Chaotic from Henon - Rossler; Demographic - Births in Quebec; Economical/Financial: from Dow Jones - SP500; Hydrology Colorado River and Lake Eriel; Physics: Kobe - Sunspots; and the last two form the Santa Fe competition.

Table 2: NRMS Error on independent test set

Time Series	*Mean*	*Std Dev*	*Min*	*Max*
Henon	1.2644	0.3603	0.6131	1.8859
Ikeda	1.0497	0.1608	0.8677	1.3498
Logistic	0.0001	0.0001	6.17E-06	0.0005
Lorenz	0.1081	0.1237	0.0108	0.3050
Mackey-Glass	0.0009	0.0008	8.15E-05	0.0025
QP2	0.1569	0.1323	0.0146	0.4524
QP3	0.9912	0.2189	0.5157	1.3168
Rossler	1.2108	2.5582	0.0016	7.7933
Births in Quebec	0.3464	0.0653	0.2769	0.4949
Dow Jones	3.6316	3.4756	1.2748	9.7451
Gold Prices	1.9218	1.1785	0.8216	4.6065
IBM Stock	1.0446	0.2636	0.7040	1.3930
SP500	1.3690	0.3370	0.6385	1.9614
Colorado River	1.6606	1.7132	0.5667	5.8620
Lake Erie	1.9232	1.0370	0.6229	3.9093
Kobe	0.6558	0.1130	0.5050	0.8941
Star	0.0256	0.0226	0.0018	0.0665
Sunspot	0.7666	0.1546	0.6242	1.0678
D1	2.4011	0.5728	1.5082	3.3198
Laser	0.1965	0.2300	0.0104	0.7309

From all the experiments we can identify four categories to classify the results. Accurate prediction: Logistic, Mackey-Glass, Star and Laser. Accurate prediction in almost all or some points: Lorenz, QP2, Rossler, Births in Quebec, Henon and Kobe. Prediction that follows the trend of the original data: QP3, D1, Colorado River, and Lake Eriel. And the cases where it was not possible to obtain a good prediction at all: Ikeda, Dow Jones, IBM Stock, SP500, Gold Prices and Sunspots. To clarify the results, some tables (average over the ten runs) and graphs are now presented which show the behaviour of the algorithm for particular TS. The graphs correspond to the best individual found over ten runs, therefore their error (NRMS) could be obtained from Table 2 column "*Min*".

The Mackey-Glass TS is a commonly used

Table 3: Average results for Mackey-Glass TS

Parameter	*Mean*	*Std Dev*	*Min*	*Max*
Number of Connections	161.5	36.1363	110	235
Number of Inputs	10.1	1.4491	8	13
Number of Hidden Nodes	12	1.4907	10	14
Error Training Set	0.0733	0.0564	0.0227	0.1973
Error Test Set EPNet	0.0001	5.16E-05	8.72E-05	0.0002
Error Final Test Set	0.0008	0.0007	8.15E-05	0.0023

Table 4: Average results for the Henon TS

Parameter	*Mean*	*Std Dev*	*Min*	*Max*
Number of Connections	128.8	28.14	85	174
Number of Inputs	9.9	0.9944	8	11
Number of Hidden Nodes	10.5	1.8408	8	13
Error on Training Set	3.0130	6.9494	0.0039	22.3053
Error on Test Set EPNet	0.0657	0.1425	0.0005	0.4675
Error on Final Test Set	1.2644	0.3603	0.6131	1.8859

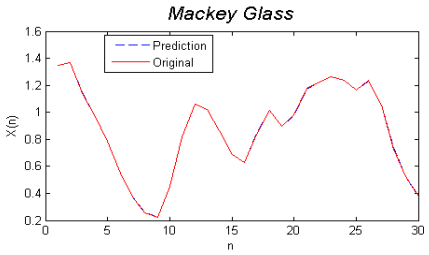

Figure 6: Mackey-Glass TS. 30 step prediction

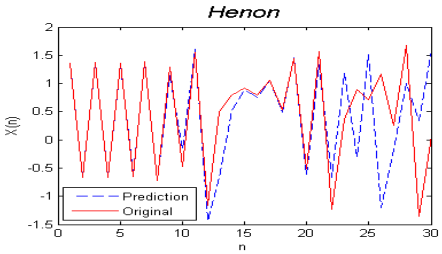

Figure 7: Henon TS. 30 step prediction

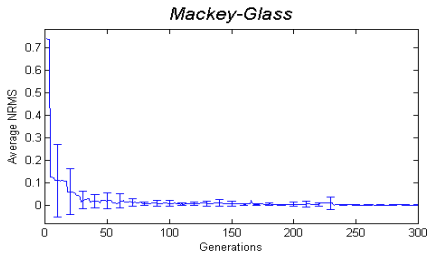

Figure 8: Mackey-Glass. Average NRMS

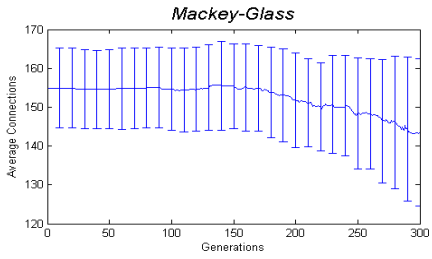

Figure 9: Mackey-Glass. Average Connections

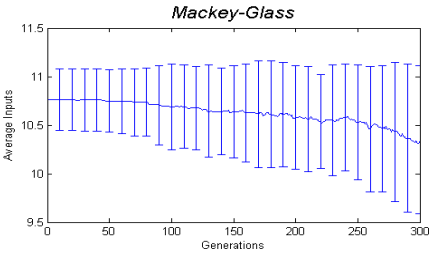

Figure 10: Mackey-Glass. Average Inputs

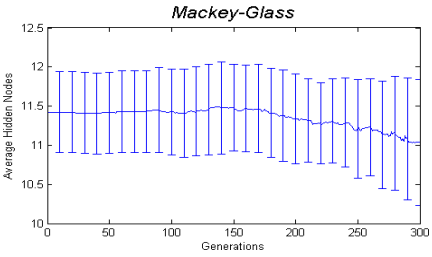

Figure 11: Mackey-Glass. Average Hidden Units

task for testing TS prediction algorithms. The accurate results shown in Table 3 are similar to previous studies with EPNet [17]. Fig. 6 compares the actual and predicted TS, showing that it is not possible to distinguish between the test set and the prediction (i.e. accurate forecasting). In Table 4 is given the results for the Henon TS, and in Fig. 7 is shown the actual TS prediction where acceptable predictions are only made for small numbers of time steps. Fig. 8 presents the average NRMS for the Mackey-Glass TS. There it is possible to see how the error is decresed considerably during the first generations of evolution. Fig. 9 shows the corresponding evolution of the connections, where they start to decrease around the middle of the evolution, as do the inputs in Fig. 10 and the number of hidden units in Fig. 11. In this case, the EPNet Algorithm was able to find a small architecture for the ANN without any decrease of performance.

5 Conclusions

The work presented in this paper is the first stage in the development of an Incremental Modular Neural Network (MNN) for Time Series (TS) Forecasting. The basis for its development has been described, and a series of experimental

test results have been presented using an improved Evolutionary Algorithm to find suitable networks, which in the future will be used as modules for the proposed algorithm. Since the algorithm must solve the TS problems as they arrive, it was not possible to specify the optimal number of inputs for each problem. Therefore modifications of the the standard EPNet evolutionary algorithm were implemented, leaving the inputs as another parameter to be adjusted by the algorithm. Also, modified early stopping procedures for the learning were implemented, and a suitable framework established for representing and testing the networks. In this way, the key issues in applying MNNs to TS forecasting have been addressed. We expect to be able to present further experimental results in a near future using the full incremental MNN.

References

[1] Center for nonlinear dynamics. http://chaos.ph.utexas.edu/~weeks/research/tseries1.html, Accessed on 20 April, 2008.

[2] J. Belaire-Franch and D. Contreras. Recurrence plots in nonlinear time series analysis: Free software. *Journal of Statistical Software*, 7(9), 2002.

[3] J. A. Bullinaria. Understanding the emergence of modularity in neural systems. *Cognitive Science*, 31(4):673–695, 2007.

[4] W. Finnoff, F. Hergert, and H. G. Zimmermann. Improving model selection by nonconvergent methods. *Neural Networks*, 6(6):771–783, 1993.

[5] L. K. Hansen and P. Salamon. Neural network ensembles. *Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

[6] HP-UX. The porting and archive centre for hp-ux. http://hpux.cs.utah.edu/hppd/cgi-bin/wwwtar?/hpux/Physics/fd3-0.3/fd3-0.3-hppa-10.10.depot.gz+fd3/fd3-RUN/opt/fd3/lib/logistic.dat+text, Accessed on 19 April, 2008.

[7] M. Husken, J. E. Gayko, and B. Sendhoff. Optimization for problem classes - neural networks that learn to learn. In *IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*. IEEE Press, 2000. 98-109.

[8] Hyndman, R.J. (n.d.). Time series data library. http://www-personal.buseco.monash.edu.au/~hyndman/TSDL/, Accessed on 20 March, 2008.

[9] V. R. Khare, X. Yao, B. Sendhoff, Y. Jin, and H. Wersing. Co-evolutionary modular neural networks for automatic problem decomposition. *The 2005 IEEE Congress on Evolutionary Computation*, 3:2691–2698 Vol. 3, 2-5 Sept. 2005.

[10] Y. Liu and X. Yao. Evolving modular neural networks which generalise well. *IEEE International Conference on Evolutionary Computation*, pages 605–610, 13-16 April 1997.

[11] L. Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, pages 55–69, London, UK, 1998. Springer-Verlag.

[12] J. Reisinger, K. O.Stanley, and R. Miikkulainen. Evolving reusable neural modules. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*, pages 68–81, New York, UK, 2004. Springer-Verlag.

[13] Santa Fe Competition. The Santa Fe time series competition data. Stanford Psychology, Stanford University. http://www-psych.stanford.edu/~andreas/Time-Series/SantaFe.html, Accessed on 20 March, 2008.

[14] F. Takens. Detecting strange attractors in turbulence. In *Dynamical Systems and Turbulence, Warwick 1980*, volume 898, pages 366–381, Berlin, 1981. Springer.

[15] E. A. Wan. Time series data. Department of Computer Science and Electrical Engineering. Oregon Health & Science University. http://www.cse.ogi.edu/~ericwan/data.html, Accessed on 19 March, 2008.

[16] E. Weeks. Chaotic time series analysis. Physics Department, Emory University. http://www.physics.emory.edu/~weeks/research/tseries1.html, Accessed on 19 March, 2008.

[17] X. Yao and Y. Liu. Epnet for chaotic time-series prediction. In *SEAL'96: Selected papers from the First Asia-Pacific Conference on Simulated Evolution and Learning*, pages 146–156, London, UK, 1997. Springer-Verlag.

[18] X. Yao and Y. Liu. A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 8(3):694–713, 1997.

[19] X. Yao and Y. Liu. Ensemble structure of evolutionary artificial neural networks. *Proceedings of IEEE International Conference on Evolutionary Computation, 1996*, pages 659–664, 20-22 May 1996.