2006 International Joint Conference on Neural Networks
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

Nominal-scale Evolving Connectionist Systems

Michael J. Watts, *Member, IEEE*

***Abstract*— A method is presented for extending the Evolving Connectionist System (ECoS) algorithm that allows it to explicitly represent and learn nominal-scale data without the need for an orthogonal or binary encoding scheme. Rigorous evaluation of the algorithm over benchmark data sets shows that it is able to learn, generalise and adapt well to classification problems. The algorithm is potentially useful for data mining tasks.**

I. Introduction

Evolving Connectionist Systems (ECoS) [9], [10] are a class of constructive artificial neural network (ANN) algorithms that are capable of fast one-pass learning and are resistant to catastrophic forgetting. They are thus well-suited to applications where new data is becoming continuously available, such as speech-recognition systems [7].

The original ECoS network was the Evolving Fuzzy Neural Network (EFuNN) [11] but other versions, including the minimalist Simple Evolving Connectionist System (SECoS) [15] have also been developed.

Many real-world problems involve nominal scale data. Here, "nominal scale" refers to the nominal scale of Stevens' Measurement Theory [13], where the magnitude of the numbers have no meaning beyond their use as class labels. Such numbers cannot, therefore, be used as input values to ANN.

The usual approach in connectionist systems is to represent nominal-scale variables orthogonally, where each possible class of each variable is assigned its own input neuron. When the variable is that class, the corresponding input is set to unity, and all other inputs associated with that variable are set to zero. Other approaches [4] use the probabilities of a variable being a specific class as inputs, although it is unclear how this scheme would deal with an open-ended data stream, such the ECoS model was designed to deal with.

While the orthogonal representation scheme is successful, there are several objections to this representation of multiple classes:

1) Expansion of the dimensionality of the input space: each class of each variable adds one dimension to the input space. For even a small number of variables, with a small number of classes each, this can quickly increase the dimensionality of the data such that the infamous "curse of dimensionality" becomes apparent. Also, interpreting the importance of the contribution of each of the input neurons becomes more difficult as the number of inputs increases.
2) Independence of inputs: each input in an orthogonal representation is independent of all other inputs. However, each class of a particular variable is not independent. If a variable is one class, it cannot be another class. Orthogonal representation ignores this restriction.
3) Changing the number of classes: as each class is represented by an independent input neuron, it is not possible to add additional classes to the input data at a later time. It is desirable in some applications (such as those where new data is becoming continuously available, which was the motivation for the ECoS design) that additional input classes be accommodated. Orthogonal representation requires that additional input neurons be added but an algorithm for doing so does not yet exist for ECoS.

National Centre for Advanced Bio-Protection Technologies, PO Box 84, Lincoln University, Canterbury, New Zealand (phone: 64-3-325-3696; fax: 64-3-325-3864; email: wattsm2@lincoln.ac.nz).

While some algorithms in computational intelligence, such as crisp rule-based systems or decision trees, are able to represent nominal-scale data explicitly, ANN cannot. It is desirable for ANN to be able to learn this data, however, especially if the ANN is being used for knowledge discovery in a data mining application. The goal of this paper, then, is to investigate the following two questions:

1) Is it possible to modify the ECoS algorithm so that nominal scale data can be explicitly represented within the structure of the networks?
2) If it is possible to do this, how well does the algorithm perform?

The remainder of this paper describes the algorithm and preliminary experimental investigations of a development of the ECoS model that allows it to learn nominal-scale data. This model is named the Nominal-scale Evolving Connectionist System, or NECoS.

II. NECoS Architecture

NECoS is a three neuron-layer constructive artificial neural network that grows parts of its structure in response to training data. The first layer is the input layer, where each neuron corresponds to a single input variable. Each input neuron has a set of "signal counters" associated with it, with there being one counter for each of the classes associated with that particular variable. Whenever an input vector is propagated through the input layer, the signal counters for each neuron are updated, with the signal counter for the class observed at each neuron being incremented.

The second neuron layer is the evolving layer. This is the layer that grows (that is, has neurons added to it) during training. The input and evolving layer are fully connected. The weights of the connections from the input layer to the evolving layer represent class labels. That is, rather than having continuous weights attached to them, as is the case with conventional ANN and other ECoS networks, the numbers attached to these connections in NECoS are solely class labels. The activation of the evolving layer neurons is based on the similarity of the current input vector I to the incoming connection weight vector. Any nominal-scale similarity measure can be used, although the output is expected to be in the range $[0, 1]$, where unity means a complete match and zero a complete mismatch. The similarity measure used in the experiments presented in this paper is the similarity coefficient, which is calculated according to Equation 1.

$$S = \frac{n_m}{n_i} \tag{1}$$

where:
n_m is the number of elements of the input and weight vector that match, and
n_i is the size of the input vector (number of input neurons).

Of the neurons in the evolving layer, only the winning (most highly activated) neuron is allowed to propagate its activation to the following neuron layer.

The third and final layer of neurons is the output layer. Again, the evolving layer and output layer are fully connected. The connections from the evolving layer to the output layer are weighted with floating point numbers. The activation of the output layer neurons is calculated by a simple multiply and sum operation over the evolving layer neuron activations and the evolving to output layer connection weights.

This architecture is shown in Figure 1. This particular NECoS network deals with three input variables, one output variable, and has two neurons in the evolving layer. The values of the signal counters are shown next to each input neuron: this shows that the first input variable has three classes associated with it, the second has two and the third four.

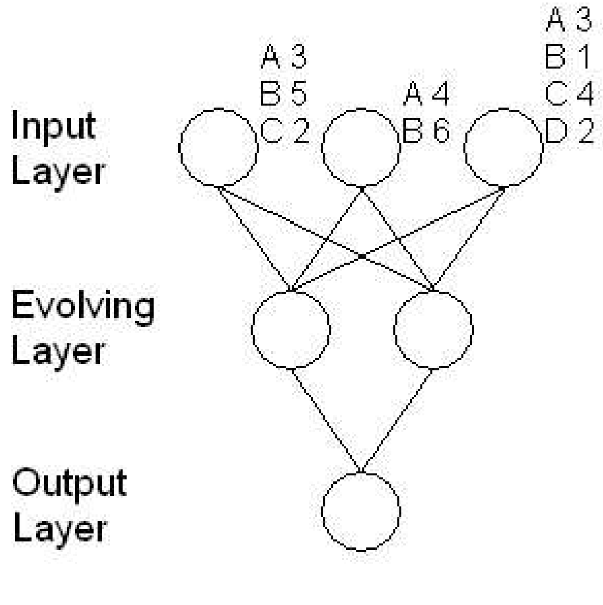

Fig. 1. Architecture of NECoS

In structure and function, the NECoS network is quite similar to the Grow and Learn (GAL) network [1]. However, GAL was restricted to learning binary input classes, and there was no learning in the connections. NECoS, on the other hand, learns both by adding neurons and by modifying connection weights, as described in the following section.

III. NECoS Learning

The learning algorithm for NECoS networks is based heavily upon the standard ECoS algorithm. The only difference between the two is the manner in which learning is affected in the input to evolving layer of connections. The overall learning algorithm is described below.

- Propagate the input vector **I** through the network.
- Find the most highly activated (winning) neuron j and its activation A_j.
- IF A_j is less than the sensitivity threshold S_{thr}
  - Add a neuron.
- ELSE
  - Evaluate the errors between the calculated output vector $\mathbf{O}_c$ and the desired output vector $\mathbf{O}_d$.
    - IF the absolute error over the output is greater than the error threshold E_{thr}
      - Add a neuron.
  - ELSE
    - Update the connections to the winning evolving layer neuron.
- Repeat for each training vector.

When a neuron is added, its incoming connection weight vector is set to the input vector **I**, and its outgoing weight vector is set to the desired output vector $\mathbf{O}_d$.

In the standard ECoS algorithm, the amount the incoming weight vector moves closer to **I** is determined by the learning rate one parameter (η_1), where a higher η_1 corresponds to a greater change. It is desirable to maintain this semantic in the NECoS training algorithm. Whereas the standard ECoS learning algorithm alters the weights in the input to evolving layer so that the neuron is spatially closer to **I**, in NECoS this would not make sense, because nominal scale weights do not describe a position is space. Instead, the learning algorithm is based on the concept of making the incoming weight vector more similar to **I** by changing some of the connection labels to match the labels in **I**, where the degree to which the incoming weight vector is changed is determined by η_1. This is achieved by the following algorithm:

- Find each connection $\mathbf{W}_{i,j}$ that does not match the corresponding element c of $\mathbf{I}_i$, that is, find the mismatching incoming connections of j. Let n_m be the number of mismatched connections, m be the set of mismatched connections, and $\mathbf{I}_c$ be the set of mismatched input vector class labels.
- For each mismatched connection m_c find, from the input neuron signal counters, the corresponding frequency f_c of the target category for that input.

- Rank the mismatching connections m_r in descending order of f_c.
- Calculate the number of connections to change $n_c = \lceil \eta_1 n_m$.
- For each of the top n_c connections in m_r, set $\mathbf{W}_{i,j}$ to $\mathbf{I}_i$

Thus, the mismatching connections are changed to equal the most frequently occurring class labels, thereby making the incoming weight vector more similar to the current input vector.

The weights from neuron j to output o are modified according to Equation 2.

$$\mathbf{W}_{j,o}(t+1) = \mathbf{W}_{j,o}(t) + \eta_2(A_j \times E_o) \tag{2}$$

where:
$\mathbf{W}_{j,o}(t)$ is the connection weight from j to output o at time t
η_2 is the learning rate two parameter for the evolving to output layer connections
A_j is the activation of j
E_o is the signed error at o, as measured according to Equation 3.

$$E_o = \mathbf{O}_d - A_o \tag{3}$$

where:
$\mathbf{O}_d$ is the desired activation value of o
A_o is the actual activation of o.

IV. Experiments

The purpose of the experiments described in this section was to evaluate the performance of the NECoS algorithm, and to compare it to that of its two immediate progenitors, SECoS and EFuNN.

Each of the data sets used in these experiments were sourced from the UCI Machine Learning Repository [8].

A. Lenses Database

The purpose of the first experiment was simply to see if the NECoS algorithm works. The contact lenses database was selected for this task, which deals with the problem of deciding which type of contact lenses to fit to a patient. The database consists of twenty-four examples, where each example represents four input variables, corresponding to the age, spectacle prescription, presence of astigmatism and tear production rate of the patient. There are three, two, two and two classes respectively. The output is three classes, specifying hard, soft, or no contact lenses.

Due to the small size of the database, bootstrapping, or resampling with replacement [5] was performed over the data. A NECoS network was trained over each sample of the data and its performance, in the form of overall percentage correctly classified, Cohen's Kappa statistic, and the number of neurons added during training, were evaluated over the sample. One thousand resamplings were performed and the mean and standard deviation of the performance metrics calculated. The training parameters used are presented in Table I. For comparison, SECoS and EFuNN networks were also trained and tested in the same manner. The results over all three types of networks are presented in Table II.

TABLE I
Training parameters for lenses data.

Parameter	Value
Sensitivity Threshold	0.5
Error Threshold	0.25
Learning Rate One (η_1)	0.5
Learning Rate Two (η_2)	0.5

TABLE II
Accuracies over lenses data.

	Percent	Kappa	Neurons
NECoS	88.78/7.65	0.80/0.14	12.62/1.70
SECoS	62.4/10.40	0.25/0.15	15.94/2.48
EFuNN	56.67/11.15	0.27/0.12	10.49/2.81

A two-tailed t-test ($p = 0.001$) showed that the performance of NECoS is significantly better than that of both SECoS and EFuNN. Also, NECoS had significantly fewer evolving layer neurons than SECoS but significantly more than EFuNN. These results show that the NECoS algorithm is able to classify examples based on categorical input values, in this case more accurately than either SECoS or EFuNN.

B. Servo Database

The servo data set is a simulation of a servo motor. The data set has a total of 167 examples. There are four input variables, with five, five, six and five classes, respectively. The output is the time required for the servo to respond to a commanded change in position. This is a function approximation problem, and it was expected that NECoS would not perform well on this type of problem. This is because the activation of the evolving layer neurons, and hence the activation of the output neurons, is based on the degree of similarity between the current input vector and the exemplar vectors as stored within the input to evolving layer connection weights. Therefore, the number of discrete values an evolving layer neuron can take is equal to the number of input neurons. In contrast, a SECoS network using orthogonal encoding of inputs can assume a larger number of states due to the larger number of input variables used and the use of continuous-valued connection weights in the input-to-evolving layer of connections.

For this experiment, the data set was randomly divided into two sets: a training and testing data set, consisting of 80% of the examples, and a validation set, consisting of the remaining 20%. For each run, two thirds of the examples in the training and test set were randomly selected as training data and the remaining third retained as testing data. A NECoS network was trained, and its accuracy evaluated over both the training and testing data sets. This was repeated one thousand times for each set of parameters, and ten different sets of parameters were investigated. At the end of the ten thousand runs, the network with the best generalisation performance (that is, the network that performed the best over the testing data) was identified and its performance over the validation set determined. This methodology was based on the procedures recommended by Flexer [6] and by Prechelt [12]. For comparison purposes, the same methodology was repeated with SECoS and EFuNN networks.

The parameter sets that yielded the best generalisation accuracies are presented in Table III. The accuracies are presented in Table IV, where both the Mean Squared Error (MSE) and R^2 metric are presented. The mean and standard deviation of the performance measures are presented.

TABLE III

TRAINING PARAMETERS FOR SERVO DATA.

Parameter	NECoS	SECoS	EFuNN
Sensitivity Threshold	0.7	0.7	0.75
Error Threshold	0.025	0.025	0.05
Learning Rate One (η_1)	0.7	0.7	0.75
Learning Rate Two (η_2)	0.7	0.7	0.75

TABLE IV

ACCURACIES OVER SERVO DATA.

		Recall Set			
		Train	Test	Validation	Neurons
NECoS	R^2	0.974/0.026	0.246/0.462	0.341	63.79/3.48
	MSE	0.119/0.121	1.788/0.629	1.159	
SECoS	R^2	0.997/0.003	0.378/0.326	0.498	65.18/3.40
	MSE	0.012/0.011	1.770/0.590	0.053	
EFuNN	R^2	0.987/0.017	0.520/0.223	0.182	74.52/4.03
	MSE	0.054/0.072	1.985/0.741	3.092	

As expected, the performance of NECoS was significantly worse (two tailed t-test, $p = 0.001$) over the testing sets that that of both SECoS and EFuNN. SECoS was significantly more accurate than NECoS over both the training and testing data sets and more accurate over the validation set than EFuNN. NECoS was, however, significantly smaller than both SECoS and EFuNN. None of the networks performed particularly well over this data set.

C. Mushrooms Database

The mushrooms database has been previously used in the testing of neural network rule extraction algorithms [2], [3] and is also sometimes used as a benchmark for ANN learning algorithms [16]. The problem is to classify mushrooms as either edible or poisonous based on their attributes. There are a total of 8124 examples, with there being twenty-two input variables, describing such things as odor of the mushroom, the mushrooms stalk and shape and the colour of the cap. The number of classes per variable ranges from two to twelve.

The same procedure as described above for the servo problem was used again, with there being ten sets of parameters investigated and one thousand resamplings done for each parameter set, for NECoS, SECoS and EFuNN.

The parameter sets that yielded the best generalisation accuracy for the three types of networks are listed in Table V. The accuracies, as percentages correctly classified and the Kappa statistic, are presented in Table VI. Again, the mean and standard deviation of the performance measures are presented.

TABLE V

TRAINING PARAMETERS FOR MUSHROOM DATA.

Parameter	NECoS	SECoS	EFuNN
Sensitivity Threshold	0.1	0.8	0.5
Error Threshold	0.6	0.1	0.025
Learning Rate One (η_1)	0.5	0.25	0.5
Learning Rate Two (η_2)	0.5	0.25	0.5

TABLE VI

ACCURACIES OVER MUSHROOM DATA.

		Recall Set			
		Train	Test	Validation	Neurons
NECoS	%	99.76/0.15	99.74/0.18	99.63	44.3/3.34
	κ	0.995/0.003	0.995/0.004	0.993	
SECoS	%	99.48/0.28	99.45/0.32	100.0	22.13/1.36
	κ	0.990/0.006	0.989/0.006	1.0	
EFuNN	%	100.0/0.0	100.0/0.0	100	795.09/16.91
	κ	1.0/0.0	1.0/0.0	1.0	

While the accuracies of EFuNN were significantly greater (two tailed t-test, $p = 0.001$) than SECoS and NECoS, and the accuracies of SECoS significantly greater than NECoS, the performance of both SECoS and NECoS was very close to that of EFuNN. The difference in accuracy between EFuNN and NECoS over the testing sets, for example, was less than 0.3%. Significantly, EFuNN achieved this performance at the cost of complexity, with there being more than eighteen times as many neurons added to EFuNN during training as there were to NECoS. When the added complexity of the EFuNN architecture is considered, the EFuNN networks had, on average, thirty-seven times as many connection as NECoS. While the accuracy of NECoS was not quite as good as the performance reported in some of the literature (both [3] and [16] report accuracies of 100%, as was achieved by EFuNN), these results do show that NECoS was able to both learn the training data and generalise to the testing data. One matter of concern was the size of the NECoS networks compared to the size of the SECoS networks - on average, NECoS had twice as many neurons in its evolving layer as SECoS, although far fewer than EFuNN.

D. Adaptation of NECoS

A central principle of the ECoS paradigm is the ability to adapt to new data [9] without catastrophic forgetting. This final experiment was intended to investigate the adaptive ability of NECoS.

The experiments were carried out over the mushroom data set (Subsection IV-C), with the same parameters as listed there. Whereas the training data in the previous experiment was presented to NECoS in one single training session, in this experiment the training data was split into two equal-sized sets, labelled Train A and Train B. A NECoS network was firstly trained over Train A, and its performance over Train A and B, and the testing set evaluated. The network was then further trained on Train B and its performance again evaluated. Of particular interest was how well it adapted to the new training data, and how much (if at all) it forgot Train A. As the purpose of this experiment is to test adaptation, the accuracies over the validation data set were not assessed. Neither was a comparison with SECoS and EFuNN carried out: previous work [15] has already demonstrated that SECoS and EFuNN can readily adapt to new data without catastrophic forgetting.

The results are presented in Table VII, where it can be seen that not only was NECoS able to adapt to the new data, but it did so without forgetting of the old data. In fact, the accuracies over data set Train A were significantly higher after further training on the Train B set, as were the accuracies over the testing data set. These results clearly show that NECoS is capable of additional learning without catastrophic forgetting.

TABLE VII
NECoS ADAPTATION ACCURACIES OVER MUSHROOM DATA.

		Recall Set			
Train Set		Train A	Train B	Test	Neurons
A	%	99.63/0.24	99.56/0.27	99.56/0.27	37.25/2.95
	κ	0.993/0.004	0.991/0.005	0.991/0.005	
B	%	99.76/0.18	99.79/0.15	99.75/0.17	44.41/3.33
	κ	0.995/0.004	0.996/0.003	0.995/0.003	

V. Conclusions and Future Work

Two questions were posed in the Introduction to this paper. Firstly, can nominal scale data be explicitly represented in an ECoS network? The answer to this question is that it is possible to construct an ECoS network that deals explicitly with and learns nominal scale data. The second question was how well does the algorithm work? The preliminary experimental results show that the algorithm is able to perform well over classification problems but does not perform well over the function approximation problem that it was tested on.

Several avenues of future research may be pursued. Firstly, investigation of similarity measures more sophisticated than the simple similarity coefficient used here is a relatively simple task with the potential to yield improved performance of the NECoS algorithm. Investigating the removal of the "winner takes all" constraint on the activation of the evolving layer neurons is also desirable, as this may allow NECoS networks to achieve better performance over function approximation problems: by allowing multiple neurons to contribute, a smoother approximation of continuous outputs can be achieved. Development of an algorithm for automatically adding input classes during training is also a possibility. Finally, an algorithm for extracting the knowledge captured by the NECoS, in the form of either crisp rules or decision trees, is also being developed. This will allow NECoS to be used for knowledge discovery and data mining.

References

[1] Alpaydin, E. GAL: Networks that grow when they learn and shrink when they forget. International journal of pattern recognition and Artificial Intelligence, 8(1):391-414. 1994.
[2] Duch, W., Adamczak, R. and Krzysztof, G. Extraction of Logical Rules from Neural Networks. Neural Processing Letters. 7:211-219. 1998.
[3] Duch, W., Adamczak, R., Grabczewski, K. and Jankowski, N. Neural methods of knowledge extraction. Control and Cybernetics 29(40. 2000
[4] Duch, W., Grudzinski, K. and Stawski, G. Symbolic features in neural networks. 5th Conference on Neural Networks and Soft Computing, Zakopane, June 2000, pp 180-185.
[5] Efron, B., and Tibshirani, R.J. An introduction to the bootstrap. Monographs on Statistics and Applied Probability, No. 57. Chapman and Hall, London. 1993.
[6] Flexer, A. Statistical Evaluation of Neural Network Experiments: Minimum Requirements and Current Practice. In: Trappl, R., Cybernetics and Systems '96, Proceedings of the 13th European Meeting on Cybernetics and Systems Research. Austrian Society for Cybernetic Studies, 1005-1008. 1996.
[7] Ghobakhlou, A. and Watts, M. and Kasabov, N. Adaptive Speech Recognition with Evolving Connectionist Systems. Information Sciences, 156:71-83. 2003.
[8] Hettich, S., Blake, C.L. and Merz, C.J. UCI Repository of machine learning databases. http://www.ics.uci.edu/∼mlearn/MLRepository.html. University of California, Irvine, Dept. of Information and Computer Sciences. 1998.
[9] Kasabov, N. The ECOS framework and the ECO learning method for evolving connectionist systems. Journal of Advanced Computational Intelligence, 2(6) 195-202. 1998.
[10] Kasabov, N. Evolving Connectionist Systems : Methods and Applications in Bioinformatics, Brain Study and Intelligent Machines. Springer Verlag, 2002.
[11] Kasabov, N., Evolving Fuzzy Neural Networks - Algorithms, Applications and Biological Motivation, in Methodologies for the Conception, Design and Application of Soft Computing, Takeshi Yamakawa and Gen Matsumoto, editors, World Scientific 271-274. 1998.
[12] Prechelt, L. A Quantitative Study of Experimental Evaluations of Neural Network Learning Algorithms: Current Research Practice. Neural Networks 9(3) 457-462. 1996.
[13] Stevens, S. S. On the theory of scales of measurement. Science, 103, 677-680. 1946.
[14] Watts, M.J. Fuzzy rule extraction from simple evolving connectionist systems. International Journal of Computational Intelligence and Applications. 4(3) 1-10. 2004.
[15] Watts, M.J. and Kasabov, N. Simple evolving connectionist systems and experiments on isolated phoneme recognition. In Proceedings of the the First IEEE Conference on Combinations of Evolutionary Computation and Neural Networks, San Antonio, May 2000, 232-239. 2000.
[16] Wilson, D.R. and Martinez, T.R. The general inefficiency of batch training for gradient descent learning. Neural Networks, 2003.