NEURAL NETWORKS AND GENETIC ALGORITHMS

NEURAL NETWORKS AND GENETIC ALGORITHMS

NEURAL NETWORKS

Knowledge Extraction from Local Function Networks

Kenneth McGarry, Stefan Wermter and John MacIntyre
School of Computing, Engineering and Technology,
University of Sunderland, St Peters Campus,
St Peters Way, Sunderland, SR6 ODD, UK
ken.mcgarry@sunderland.ac.uk

Abstract

Extracting rules from RBFs is not a trivial task because of nonlinear functions or high input dimensionality. In such cases, some of the hidden units of the RBF network have a tendency to be "shared" across several output classes or even may not contribute to any output class. To address this we have developed an algorithm called LREX (for Local Rule EXtraction) which tackles these issues by extracting rules at two levels: *h*REX extracts rules by examining the *hidden* unit to class assignments while *m*REX extracts rules based on the input space to output space *mappings*. The rules extracted by our algorithm are compared and contrasted against a competing local rule extraction system. The central claim of this paper is that local function networks such as radial basis function (RBF) networks have a suitable architecture based on Gaussian functions that is amenable to rule extraction.

1 Introduction

Neural networks have been applied to many real-world, large-scale problems of considerable complexity. They are useful for pattern recognition and they are robust classifiers, with the ability to generalize in making decisions about imprecise input data [Bishop, 1995]. They offer robust solutions to a variety of classification problems such as speech, character and signal recognition, as well as functional prediction and system modeling where the physical processes are not understood or are highly nonlinear.

Although neural networks have gained acceptance in many industrial and scientific fields they have not been widely used by practitioners of mission critical applications such as those engaged in aerospace, military and medical systems. This is understandable since neural networks do not lend themselves to the normal software engineering development process. Knowledge extraction by forming symbolic rules from the internal parameters of neural networks is now becoming an accepted technique for overcoming some of their limitations [Shavlik, 1994; Sun, 2000].

In this paper we describe our method of extracting knowledge from an RBF network which is classed as a local type of neural network. That is, its internal parameters are limited to responding to a limited subset of the input space. We also compare and contrast our technique with a specialized local type neural architecture. The extracted rules are examined for comprehensibility, accuracy, number of rules generated and the number of antecedents contained in a rule.

The paper is structured as follows: section two describes the motivations for performing knowledge extraction. Section three describes why the architecture of the radial basis function network is particulary suitable for knowledge extraction. Section four outlines how our knowledge extraction algorithm produces rules from RBF networks and section five explains the results of the experimental work. Section six discusses the conclusions of the experimental work.

2 Knowledge Extraction

In this section we discuss motivations, techniques and methodology for knowledge extraction from RBF networks. RBF networks provide a localized solution [Moody and Darken, 1989] that is amenable to extraction, which section three discusses in more detail. It is possible to extract a series of IF..THEN rules that are able to state simply and accurately the knowledge contained in the neural network. In recent years there has been a great deal of interest in researching techniques for extracting symbolic rules from neural networks. Rule extraction has been carried out upon a variety of neural network types such as multi-layer perceptrons [Thrun, 1995], Kohonen networks and recurrent networks [Omlin and Giles, 1994]. The advantages of extracting rules from neural networks can be summarized as follows:

- The knowledge learned by a neural network is generally difficult to understand by humans. The provision of a mechanism that can interpret the networks input/output mappings in the form of rules would be very useful.

- Deficiencies in the original training set may be identified, thus the generalization of the network may be improved by the addition/enhancement of new classes. The identification of superfluous network parameters for removal would also enhance network performance.
- Analysis of previously unknown relationships in the data. This feature has a huge potential for knowledge discovery/data mining and possibilities may exist for scientific induction [Craven and Shavlik, 1997].

In addition to providing an explanation facility, rule extraction is recognised as a powerful technique for neuro-symbolic integration within hybrid systems [McGarry *et al.*, 1999].

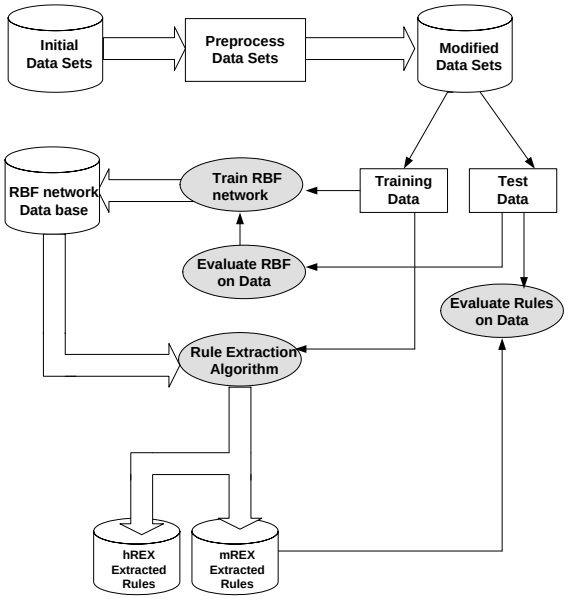


Figure 1: Knowledge extraction system data flow and data transformation

3 Radial Basis Function Networks

Radial basis function (RBF) neural networks are a model that has functional similarities found in many biological neurons. In biological nervous systems certain cells are responsive to a narrow range of input stimuli, for example in the ear there are cochlear stereocilla cells which are locally tuned to particular frequencies of sound [Moody and Darken, 1989]. Figure 2 shows a network trained on a noisy Xor data set for illustration. This network has two input features, two output classes and four hidden units.

The RBF network consists of a feedforward architecture with an input layer, a hidden layer of RBF "pattern" units and an output layer of linear units. The input layer simply

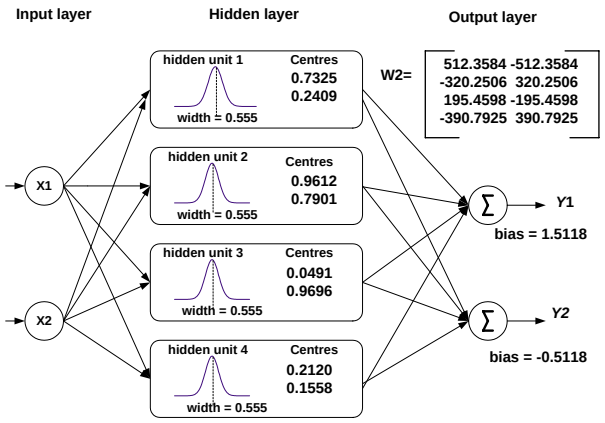


Figure 2: Parameters for RBF network trained on noisy Xor

transfers the input vector to the hidden units, which form a localized response to the input pattern. Learning is normally undertaken as a two-stage process. The first stage consists of an unsupervised process in which the RBF centres (hidden units) are positioned and the optimum field widths are determined in relation to the training samples.

The second stage of learning involves the calculating the hidden unit to output unit weights and is achieved quite easily through a simple matrix transformation.

The radial basis functions in the hidden layer are implemented by kernel functions, which operate over a localized area of input space. The effective range of the kernels is determined by the values allocated to the centre and width of the radial basis function. The Gaussian function has a response characteristic determined by equation 1.

$$Z_j(x) = exp\left(-\frac{||x-\mu||^2}{\sigma_j^2}\right) \quad (1)$$

The response of the output units is calculated quite simply using equation 2.

$$\sum_{j=l}^{J} W_{lj} Z_j(x) \quad (2)$$

where:
W = weight matrix, Z = hidden unit activations,
x = input vector, μ = n-dimensional parameter vector,
σ = width of receptive field.

3.1 RBF training

The first stage was to train RBF networks to an acceptable level of accuracy on all data sets. The specific level of accuracy varied with each data set, the literature was examined to

provide guidance on what accuracy levels could be achieved. The accuracy levels stated in the tables are the best out of up to 10 test runs. Training of the RBF networks required the setting of three parameters, the global error, the spread or width of the basis function and the maximum number of hidden units. The value assigned to the global error setting may result in fewer hidden units being used than the maximum value. If the error value is not reached, training will terminate when the maximum number of hidden units has been assigned. The training and test data for the construction of the RBF networks were generally split 75/25.

3.2 Data Sets

In order to allow good benchmarking and comparison we used a mixture of well known benchmark data as well as two new vibration data sets for our tests. The data sets were selected from various sources but mainly obtained from the collection maintained by the University of California at Irvine (UCI). The vibration data sets were produced as part of two large projects which were concerned with the monitoring the health of industrial machinery. The data sets represent a variety of synthetic and real world problems of varying complexity (i.e. number of examples, input features and classes).

Table 1: Composition of data sets used in experimental work

Data Set	Ex	O/P	I/P	C	D	M
Xor(binary)	4	2	2	No	Yes	No
Xor(continuous)	100	2	2	Yes	No	No
Iris	150	3	4	Yes	No	No
Vowell(Peterson)	1520	10	5	Yes	Yes	No
Vowell(Deterding)	990	11	11	Yes	Yes	No
Protein(yeast)	1484	10	8	Yes	No	No
Protein(ecoli)	336	8	8	Yes	No	No
Credit(Japanese)	125	2	9	Yes	Yes	Yes
Credit(Australian)	690	2	15	Yes	Yes	Yes
Diabetes(Pima)	768	2	8	Yes	No	No
Monks1	556	2	6	No	Yes	No
Sonar	208	2	60	Yes	No	No
Vibration 1	1028	3	9	Yes	No	No
Vibration 2	1862	8	20	Yes	No	No

Table 1 gives details of the data sets. The columns indicate the number of examples, the number of ouput features or classes, the number of input features, whether the data set contains continuous data or discrete data and the last column indicates if any data is missing.

4 LREX: Rule Extraction Algorithm

The development of the LREX algorithm was motivated by the local architecture of RBF networks which suggested that rules with unique characteristics could be extracted. In addition, there was little published work on extracting rules from ordinary RBF networks [Lowe, 1991]. Therefore our work fills a substantial gap in rule extraction research.

The LREX algorithm is composed of two modules: the *m*REX module extracts IF..THEN type rules based on the premise that a hidden unit can be uniquely assigned to a specific output class. Therefore, by using the centre locations of the hidden units an input vector could be directly mapped to an output class. Experimental work performed on the simpler data sets tended to reinforce this belief. However, hidden unit sharing occurs within networks trained on non-linear or complex data. This phenomena reduces rule accuracy as several hidden units may be shared amongst several classes. The second module, *h*REX was developed to identify which hidden units are shared between classes. Analysis of how each hidden unit contributes provides information to determine a class. The extracted rules are IF..THEN type rules where any given hidden may appear across several classes. The next two sections describe how the *m*REX and *h*REX modules provide the user with complimentary types of extracted rules that explain the internal operation of the original RBF network.

4.1 *m*REX: Input-to-output mapping

The functionality of *m*REX algorithm is shown in figure 3.

Input:
- Hidden weights μ (centre positions)
- Gaussian radius spread σ
- Output weights $W2$
- Statistical measure S
- Training patterns

Output:
- One rule per hidden unit

Procedure:
- Train RBF network on data set
- Collate training pattern *"hits"* for each hidden unit
- For each hidden unit
  - Use $W2$ correlation to determine Class label
  - Use *"hits"* to determine S
  - Select S format $\{min, max, std, mean, med\}$
  - For each μ_i
    - $X_lower = \mu_i - \sigma_i * S$
    - $X_upper = \mu_i + \sigma_i * S$
  - Build rule by:
    - antecedent $= [X_lower; X_upper]$
    - Join antecedents with AND
    - Add Class label
  - Write rule to file

Figure 3: *m*REX rule-extraction algorithm

The first stage of the *m*REX algorithm is to use the W2 weight matrix (see figure 2) to identify the class allocation of each hidden unit. The next stage is to calculate the lower and upper bounds of each antecedent by adjusting the cen-

tre weights μ using the Gaussian spread σ . The lower and upper limits are further adjusted using a statistical measure S gained from the training patterns classified by each hidden unit. S is used empirically to either contract or expand each antecedents range in relation to the particular characteristics of these training patterns.

The entire rule set for the Iris domain is presented in figure 4. Note that there are four extracted rules, one for each RBF hidden unit.

Rule 1 :
IF (SepalLength $\geq$ 4.1674 AND $\leq$ 5.8326) AND
IF (SepalWidth $\geq$ 2.6674 AND $\leq$ 4.3326) AND
IF (PetalLength $\geq$ 0.46745 AND $\leq$ 2.1326) AND
IF (PetalWidth $\geq$ 0.53255 AND $\leq$ 1.1326)
THEN..Setosa

Rule 2 :
IF (SepalLength $\geq$ 5.2674 AND $\leq$ 6.9326) AND
IF (SepalWidth $\geq$ 1.9674 AND $\leq$ 3.6326) AND
IF (PetalLength $\geq$ 3.1674 AND $\leq$ 4.8326) AND
IF (PetalWidth $\geq$ 0.46745 AND $\leq$ 2.1326)
THEN..Versicolor

Rule 3 :
IF (SepalLength $\geq$ 5.9674 AND $\leq$ 7.6326) AND
IF (SepalWidth $\geq$ 2.3674 AND $\leq$ 4.0326) AND
IF (PetalLength $\geq$ 5.0674 AND $\leq$ 6.7326) AND
IF (PetalWidth $\geq$ 1.4674 AND $\leq$ 3.1326)
THEN..Virginica

Rule 4 :
IF (SepalLength $\geq$ 4.8674 AND $\leq$ 6.5326) AND
IF (SepalWidth $\geq$ 1.6674 AND $\leq$ 3.3326) AND
IF (PetalLength $\geq$ 4.1674 AND $\leq$ 5.8326) AND
IF (PetalWidth $\geq$ 1.1674 AND $\leq$ 2.8326)
THEN..Virginica

Figure 4: *m*REX extracted rules from Iris domain

4.2 *h*REX: Hidden unit analysis

A different approach to rule extraction is taken by the *h*REX algorithm which uses quantization and clustering on the network parameters (weights and activation levels) to form an abstraction of its operation. The number of extracted rules is determined by the user who can place an upper limit on the rules extracted for each class. This is a useful feature since it enables a tradeoff to be made between rule size and rule comprehensibility. This is achieved by three important parameters:

- α which determines the minimum weight value (postive) to be quantized as a "one", weights below this cutoff point are quantized to -1 and do not participate in rule extraction.
- β which determines the minimum hidden unit activation level. Hidden units with activation levels below this cutoff point will not be quantized and will play no further part in rule extraction.
- N determines the maximum number of clusters that the training set (for each class) is divided into. This process abstracts the input space into a number of distinct regions which will require a separate rule to identify.

These parameters are determined empirically for a satisfactory arrangement. Figure 5 shows the algorithm in detail. Note that valid rules consist of both a positive quantized weight (QW2) and a positive quantized activation (AQZ) level. A rule consists of one or more hidden units which must all be active for the class lable to be satisified.

Input:
Output weights $W2$
Hidden unit activations Z (training data)
Output weights quantization modifier α
Hidden unit activation quantization modifier β
Maximum Cluster number *N*
Training patterns by sorted by class *T*
Intermediate information:
Quantized W2 weights *QW2*
Quantized hidden unit activations *QZ*
Average Quantized hidden unit activations *AQZ*
Output:
One rule per cluster
Procedure:
Quantize W2 weights with α
Quantize hidden unit activations Z with β
Separate training patterns by class *T*
For each class
  Partition *QZ* up to *NC* Clusters
  For each *N* Cluster
    Identify Positive *QZ* activations
    Calculate Average *AQZ* value for cluster
    Identify Positive *QW2* weights attached to QZ
  Build rule by:
    IF AQZ==Positive AND QW2 ==Positive
      Hidden unit *H* belongs to rule
      Join Hidden Units with AND
      Add Class label
  Write rule to file

Figure 5: *h*REX rule-extraction algorithm

Some *h*Rules rules extracted from the ecoli domain are presented in figure 6. For instance, for Rule 4 to "fire", each antecedent must be satisfied so hidden units 5, 19, 20, 24, 25, 26, 28 and 31 must all be active. It can be seen that hidden unit 20 participates in both class 3 and class 4.

*h*REX rules are useful for identifying the internal structural relationships formed by the hidden units. This is demon-

```
Rule ♯9 Class: 3
IF((H5 == TRUE) AND
   (H19 == TRUE) AND
   (H20 == TRUE) AND
   (H24 == TRUE) AND
   (H25 == TRUE) AND
   (H26 == TRUE) AND
   (H28 == TRUE) AND
   (H31 == TRUE))
THEN
   Class: 3

Rule ♯10 Class: 4
IF((H12 == TRUE) AND
   (H20 == TRUE) AND
   (H22 == TRUE) AND
   (H23 == TRUE) AND
   (H32 == TRUE))
THEN
   Class: 4
```

Figure 6: *h*REX extracted rules from ecoli domain

strated on those RBF networks that have a poor performance on certain classes. These RBF networks produce *h*REX rules which exhibit a large degree of hidden unit sharing or in the worse cases fail to generate any *h*REX rules for these classes.

Figure 7 shows the accuracy of the *h*REX rules against the rule size (comprehensibility) for RBF networks trained on the Vibration 1, Monks and Sonar data sets. The Vibration shows a steady increase in accuracy with each additional rule until it levels off at a cluster size of 12. The rules extracted from Sonar actually lose accuracy beyond a certain point before the accuracy reaches a steady value. Generating additional rules for the Monks after the optimum cluster size is reached produces an oscillating effect where the accuracy does not level off.

5 Analysis of Results

The performance of the RBF rule extraction algorithm was compared with a related system called MCRBP/RULEX which was developed by Andrews and Geva [Andrews and Geva, 1999]. MCRBP builds RBF-like networks with specialized activation functions. Once the networks are trained, the RULEX algorithm can then be used to extract IF..THEN rules with boundaries. The rules extracted by RULEX are in a very similar format to those produced by the author's system. Table 2 shows the results of the experimental work. The first column identifies the data set. The second column presents the *mREX* accuracy alongside the original RBF accuracy. The third column details the *h*REX accuracy next to the original RBF accuracy and the fourth column shows the Rulex accuracy

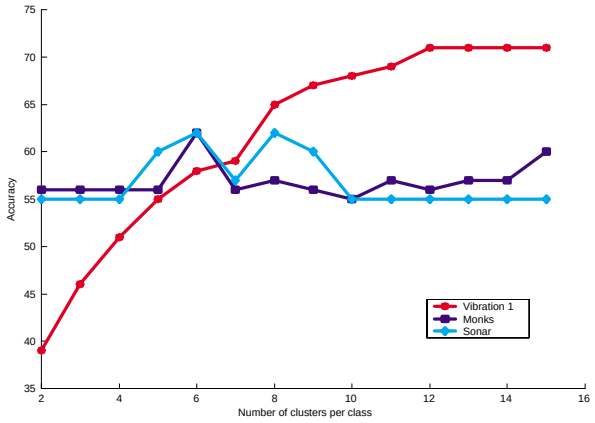


Figure 7: hREX rule size and complexity

Table 2: Comparison between RBF net, mREX, hREX and Rulex accuracy

Data set	***m*REX**	***h*REX**	**Rulex**
Xor(binary)	100/100	100/100	100
Xor(continuous)	96/100	100/100	100
Iris	93/96	93/96	100
Vowell(Peterson)	43/86	22/86	–
Vowell(Deterding)	9/62	20/62	38
Protein(yeast)	26/57	66/57	28
Protein(ecoli)	49/87	72/87	88
Credit(Japanese)	73/93	66/93	93
Credit(Australian)	66/71	64/71	88
Diabetes(Pima)	65/76	70/76	69
Monks1	79/83	60/83	72
Sonar	57/95	58/95	–
Vibration 1	56/73	69/73	61
Vibration 2	73/94	72/94	–

Table 3 shows the number of rules generated by the three systems. The rule set size quoted for LREX is based on the unmodified basic version.

RULEX extracts highly compact rule sets compared with LREX. The majority of the domains can be represented with as few as 3-5 rules. Unfortunately, RULEX completely failed to generate rules for three of the domains. This problem was tracked down to the initial MCRBP network, as it was unable to form a viable classifier on the training data. Therefore, any rules extracted would be invalid. RULEX also failed to provide rules to cover a specific class in the vibration 1 domain. Training the MCRBP networks took fewer attempts to reach acceptable accuracies than the equivalent RBF networks (typically 2-3 runs).

MCRBP/RULEX could not form a viable network on the vowel, sonar and vibration 2 domains. It is likely that the specialized architecture cannot cope with the large number of

Table 3: Comparison between rule set size of mREX, hREX and Rulex

Data set	mREX	hREX	Rulex
Xor(binary)	4	4	4
Xor(continuous)	4	4	4
Iris	4	4	5
Vowell(Peterson)	30	80	–
Vowell(Deterding)	200	110	11
Protein(yeast)	120	24	9
Protein(ecoli)	35	24	9
Credit(Japanese)	50	6	2
Credit(Australian)	50	20	5
Diabetes(Pima)	300	11	3
Monks1	20	24	3
Sonar	20	10	–
Vibration 1	30	25	2
Vibration 2	100	32	–

input features present in these data sets. However, by using non-overlapping local functions the MCRBP/RULEX algorithm can form a rule from each function that is specific to a class. This requires fewer rules to form a classifier.

The *h*REX algorithm produces fewer rules than the *m*REX algorithm and are generally more accurate. A smaller rule set enables a better understanding of the internal operation of the RBF network. further analysis of the *h*REX rules proved to be interesting as several of the RBF networks have up to 35-40% of their hidden units shared between the various output classes. Such results tend to occur with those RBF networks that have lower accuracies and may implie that the original settings of the internal parameters during training were not optimal e.g. a badly chosen value for the width of the basis function can be a source of error.

6 Conclusions

The work described in this paper has tackled the difficult issue of knowledge extraction from RBF networks which has been avoided in the literature because of the problems with overlapping neurons. The rules extracted by the LREX algorithm provide information about the original RBF network in two forms; an input to output mapping and information regarding those hidden units that participate in classification. The knowledge extracted by the *m*REX algorithm transforms the original RBF network into a rule based classifier. This makes the input to output mapping of the RBF network transparent and open to scrutiny. However, the number of rules produced is dependent on the number of hidden units and therefore a large number of rules may obscure the comprehensibility. This problem is partially solved by the *h*REX algorithm which can generate a maximum number of rules determined in advance by the user. The tradeoff is rule size (and generally accuracy) versus comprehensibility. Some RBF networks may naturally be described by small rule sets that are accurate but still allow a good understanding of their internal structure. Other RBF networks may have modeled complex functions and their hidden units are used by several classes, in which case the *h*REX algorithm will provide useful information regarding the extent of this activity.

References

[Andrews and Geva, 1999] R. Andrews and S. Geva. On the effects of initialising a neural network with prior knowledge. In *Proceedings of the International Conference on Neural Information Processing (ICONIP'99)*, pages 251–256, Perth, Western Australia, 1999.

[Bishop, 1995] C. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[Craven and Shavlik, 1997] M. Craven and J. Shavlik. Using neural networks for data mining. *Future Generation Computer Systems*, 1997.

[Lowe, 1991] D. Lowe. On the iterative inversion of RBF networks: a statistical interpretation. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 29–33, Bournemouth, UK, 1991.

[McGarry *et al.*, 1999] K. McGarry, S. Wermter, and J. MacIntyre. Hybrid neural systems: from simple coupling to fully integrated neural networks. *Neural Computing Surveys*, 2(1):62–93, 1999.

[Moody and Darken, 1989] J. Moody and C. J. Darken. Fast learning in networks of locally tuned processing units. *Neural Computation*, pages 281–294, 1989.

[Omlin and Giles, 1994] C. W. Omlin and C. L. Giles. Extraction and insertion of symbolic information in recurrent neural networks. In V.Honavar and L.Uhr, editors, *Artificial Intelligence and Neural Networks:Steps Towards principled Integration*, pages 271–299. Academic Press, San Diego, 1994.

[Shavlik, 1994] J. Shavlik. A framework for combining symbolic and neural learning. *Machine Learning*, 14:321–331, 1994.

[Sun, 2000] R. Sun. Beyond simple rule extraction: the extraction of planning knowledge from reinforcement learners. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, Lake Como, Italy, 2000.

[Thrun, 1995] S. Thrun. Extracting rules from artificial neural networks with distributed representations. In G.Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7*. MIT Press, San Mateo, CA, 1995.

Violation-Guided Learning for Constrained Formulations in Neural-Network Time-Series Predictions*

Benjamin W. Wah and Minglun Qian
Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois, Urbana-Champaign
Urbana, IL 61801, USA
E-mail: {wah, m-qian}@manip.crhc.uiuc.edu

Abstract

Time-series predictions by artificial neural networks (ANNs) are traditionally formulated as unconstrained optimization problems. As an unconstrained formulation provides little guidance on search directions when a search gets stuck in a poor local minimum, we have proposed recently to use a constrained formulation in order to use constraint violations to provide additional guidance. In this paper, we formulate ANN learning with cross-validations for time-series predictions as a non-differentiable nonlinear constrained optimization problem. Based on our theory of Lagrange multipliers for discrete constrained optimization, we propose an efficient learning algorithm, called *violation guided back-propagation* (VGBP), that computes an approximate gradient using back-propagation (BP), that introduces annealing to avoid blind acceptance of trial points, and that applies a relax-and-tighten (R&T) strategy to achieve faster convergence. Extensive experimental results on well-known benchmarks, when compared to previous work, show one to two orders-of-magnitude improvement in prediction quality, while using less weights.

1 Introduction

We study in this paper new formulations and learning algorithms for predicting stationary time-series using artificial neural networks (ANNs).

ANNs for modeling time-series generally have special structures that store temporal information either explicitly using time-delayed structures or implicitly using feedback structures. Examples of the first class include time-delayed neural networks (TDNN) and FIR neural networks (FIR-NN), whereas examples of the latter include recurrent neural networks (RNN) [Haykin, 1994]. In the ANNs studied in this paper, we use a hybrid architecture with a recurrent structure, whose neurons are connected by FIR filters, instead of links with constant weights. We believe that such an architecture is more powerful for modeling unknown temporal information.

Time-series predictions using ANNs have traditionally been formulated as an unconstrained optimization problem of minimizing the mean squared errors (MSE):

$$\min_{w} \ \mathcal{E}_{av}(t_0, t_1) = \sum_{t=t_0}^{t_1} \sum_{i=1}^{N_o} (o_i(t) - d_i(t))^2, \qquad (1)$$

where N_o is the number of output nodes in the ANN, o_t and d_t are, respectively, the actual and desired outputs of the ANN at time t, w is a vector of all the weights, and the training data consist of patterns observed at $t = t_0, \cdots, t_1$. Extensive research has been conducted in the past on designing ANNs with a small number of weights that can generalize well. However, such learning algorithms have limited success because little guidance is provided in an unconstrained formulation when a search is stuck in a local minimum in its weight space. In this case, the sum of squared errors in (1) does not indicate which patterns are violated and the best direction for the trajectory to move.

To address the issue on lack of guidance, we have proposed recently a constrained formulation [Wah and Qian, 2000] on ANN learning that accounts for the error on each training pattern in a constraint:

$$\min_{w} \ \mathcal{E}_{av}(t_0, t_1) = \sum_{t=t_0}^{t_1} \sum_{i=1}^{N_o} \phi((o_i(t) - d_i(t))^2 - \tau) \qquad (2)$$

$$\text{such that } \forall\ i, t,\ h_i(t) = (o_i(t) - d_i(t))^2 \leq \tau,$$

where $h_i(t) \leq \tau$ prescribes that the error of the i^{th} output unit on the t^{th} training pattern be less than τ, and $\phi(x) = \max\{0, x\}$. A constrained formulation is beneficial in difficult training scenarios because violated constraints provide additional guidance during a search, leading a trajectory towards a direction that reduces overall constraint violations.

In time-series prediction, *cross validations* are often used to prevent data from over-fitting. There are two types of cross-validation errors: *single-step validation errors* that measure output errors when external inputs to an ANN are true observed data, and *iterative validation errors* that measure output errors when external inputs to an ANN are predicted outputs from previous iterations.

*Research supported by National Aeronautics and Space Administration Contract NAS 2-37143.

Proc. Int'l Joint Conf. on Artificial Intelligence, AAAI, 2001.

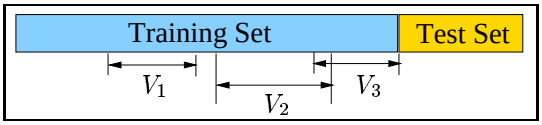


Figure 1: Multiple validation sets in a training set. V_1,V_2 and V_3 are three validation sets. The test test is used for testing the ANN after learning is completed.

In traditional learning with cross validations, a part of training patterns is reserved *a priori* in a validation set and not used in learning, and the single objective in learning is to minimize validation errors. Hence, errors measured in validation will not be included in learning. This approach is problematic because it allows only one validation set in learning and excludes patterns in the validation set to be used in learning. As a result, learning may not converge when the time-series contains multiple regimes or when training patterns are scarce.

Based on a constrained formulation, we have proposed a new cross-validation method [Wah and Qian, 2000] that defines multiple validation sets in learning and that includes the error from each validation set as a new constraint (see Figure 1). The use of multiple validation sets is especially suitable for time-series with inadequate training data and for time-series with multiple stationary regimes. The validation error for the i^{th} output unit from the k^{th}, $k = 1, \cdots, v$, validation set is defined by the *normalized mean squared error* ($nMSE$):

$$nMSE = e_{k,i} = \frac{1}{\sigma_i^2 N_k} \sum_{t=t_{k,0}}^{t_{k,1}} (o_i(t) - d_i(t))^2, \qquad (3)$$

where σ_k^2 is the variance of the true time series in $[t_{k,0}, t_{k,1}]$, and N_k is the number of patterns in the k^{th} validation set. (Note that errors on the test set in Figure 1 are defined in a similar way.) The constrained formulation then becomes:

$$\begin{aligned} \min_w \mathcal{E}_{av}(t_0, t_1) &= \sum_{t=t_0}^{t_1} \sum_{i=1}^{N_o} \phi((o_i(t) - d_i(t))^2 - \tau) \\ \text{s.t.} \quad & h_i(t) = (o_i(t) - d_i(t))^2 \le \tau, \\ & h^I_{k,i}(w) = e^I_{k,i} \le \tau^I_{k,i}, \\ & h^S_{k,i}(w) = e^S_{k,i} \le \tau^S_{k,i}, \end{aligned} \qquad (4)$$

where e^I (resp. e^S) is the $nMSE$ of the iterative (resp. single-step) validation error, and τ, $\tau^I_{k,i}$ and $\tau^S_{k,i}$ are predefined small positive constants.

Eq. (4) is a constrained nonlinear programming problem (NLP) with *non-differentiable functions*. The formulation, when applied to large time-series predictions, cannot be handled by existing Lagrangian methods that require the differentiability of functions. Methods based on penalty formulations have difficulties in convergence when penalties are not chosen properly. Sampling algorithms [Wah and Wang, 1999] based on our recently developed theory of Lagrange multipliers for discrete constrained optimization [Wah and Wu, 1999], when continuous variables are discretized to floating-point numbers, are too inefficient for solving large learning problems. To address this issue, we present in this paper an efficient learning algorithm called *violation-guided back-propagation* (VGBP).

2 Theory of Lagrange Multipliers for Discrete Constrained Optimization

To use a Lagrangian method to solve (4), we first transform it into an augmented Lagrangian function:

$$\begin{aligned} L(w, \lambda) &= \mathcal{E}_{av}(t_0, t_1) \qquad (5) \\ &+ \sum_{t=t_0}^{t_1} \sum_{i=1}^{N_o} \left(\lambda_i(t) \phi(h_i(t) - \tau) + \frac{1}{2} \phi^2(h_i(t) - \tau) \right) \\ &+ \sum_{j=I,S} \sum_{k=1}^{v} \sum_{i=1}^{N_o} \left(\lambda^j_{k,i} \phi(h^j_{k,i} - \tau^j_{k,i}) + \frac{1}{2} \phi^2(h^j_{k,i} - \tau^j_{k,i}) \right). \end{aligned}$$

Since (5) is not differentiable, we discretize variables finely and solve the problem in discrete space using the theory of Lagrange multipliers for discrete constrained optimization [Wah and Wu, 1999]. Algorithm designed to solve constrained NLPs in discrete space can be extended to solve constrained NLPs in continuous space because numerical evaluations of continuous variables using digital computers can be considered as discrete approximations of the original variables up to a computer's precision. The theory in discrete space is summarized in three definitions and one theorem.

Definition 1. $\mathcal{N}_{dn}(w)$ is a *finite* user-defined set of points w' so that w' is reachable from w in one step and that w can reach any point in the discrete weight space through $\mathcal{N}_{dn}(w)$.

Definition 2. Point w is a CLM_{dn} iff (4) is feasible at w and the objective function is the smallest in $\{w\} \cup \mathcal{N}_{dn}(w)$. Note that a CLM_{dn} of (4) is the same as a feasible point.

Definition 3. $SP_{dn}(w^\star, \lambda^\star)$, a *discrete-neighborhood saddle point* at $(w^\star, \lambda^\star)$, satisfies:

$$L_d(w^*, \lambda) \le L_d(w^*, \lambda^*) \le L_d(w, \lambda^*) \qquad (6)$$

for all $w \in \mathcal{N}_{dn}(w^*)$ and all real vector λ.

Theorem 1. *First-order necessary and sufficient condition on CLM_{dn}* [Wah and Wu, 1999]. A point in the discrete space of (4) is a CLM_{dn} iff it satisfies (6) for any $\lambda \ge \lambda^*$, where $\lambda \ge \lambda^*$ means that each element of λ is not less than the corresponding element of λ^*.

The theorem shows that solving (4) in discrete space is equivalent to (the much easier problem of) finding SP_{dn} of (5). Note that the theorem does not hold in continuous space.

3 Violation-Guided Back-Propagation

In this section we describe an efficient algorithm to look for SP_{dn} in the Lagrangian space defined in (5). The shaded box in Figure 2 [Wah and Chen, 2000] shows the framework with two parts: one performing descents in the w subspace and another performing ascents in the λ subspace. As indicated earlier, random sampling in a Lagrangian space with discrete w is too inefficient. To this end, we propose in Section 3.1 to use BP to compute an approximate gradient direction in order to generate a probe. Since gradient descents may lead to infeasible local minima, we present a new annealing strategy in Section 3.2 to help escape from infeasible points. Last, we exploit special properties in the constrained formulation for

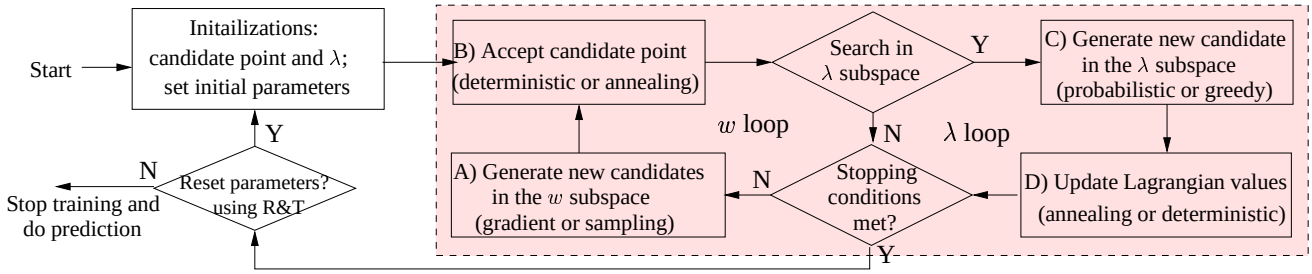


Figure 2: An iterative learning procedure using a discrete constrained formulation for ANN time-series prediction. The shaded box represents the routine to look for SP_{dn}. R&T stands for our proposed *relax-and-tighten* strategy.

ANN learning and present in Section 3.3 a new relax-and-tighten (R&T) strategy to successively tighten constraints as more relaxed constraints are satisfied. The R&T strategy is depicted in the two boxes on the left of Figure 2.

3.1 Framework to look for SP_{dn}

The w loop in Figure 2 performs descents in the w subspace by generating candidates in Box (A) and by accepting the candidates generated using deterministic or annealing rules in Box (B). Occasionally, the λ loop carries out ascents in the λ subspace by generating candidates in the λ subspace in Box (C) and by accepting them using deterministic or annealing rules in Box (D). In this subsection we present the functions of Boxes (A), (C) and (D) and leave the discussion of Box (B) to the next subsection.

For a learning problem with a large number of weights and/or training patterns, it is essential that the points generated be likely candidates to be accepted. Since (5) is not differentiable, we choose an approximate gradient direction by setting output error $g_i'(t) \leftarrow \lambda_i(t) g_i(t)$, applying BP to compute the gradient of the mean squared errors of $g_i'(t)$, generating a trial point using the approximate gradient and step size η, and mapping the trial point to (discretized) floating-point space. In this way, a training pattern with a large error (and its corresponding Lagrange multiplier) will contribute more in the overall gradient direction, leading to an effective suppression of constraint violations,

Step size η used in deriving a candidate point must be dynamic because the same candidate point will be generated repeatedly using a fixed η and a deterministic gradient algorithm. In our algorithm, we generate η uniformly in $(0, \eta_0)$ and adapt η_0 dynamically based on the acceptance ratio a of candidate points generated. The reason for the latter strategy is that a high a indicates that the current direction is promising, leading to increases in η_0 and larger step sizes. On the other hand, a low a indicates that the step size is too large for the current search terrain, leading to decreases in η_0 and smaller step sizes. After extensive experiments, we adjust η_0 as follows:

$$\eta_0 \longleftarrow \begin{cases} \eta_0 * \left(1 + \frac{2(a-0.7)}{1-0.7}\right) & \text{if } a > 0.7 \\ \eta_0 \div \left(1 + \frac{2(0.5-a)}{0.5}\right) & \text{if } a < 0.5 \end{cases} \tag{7}$$

Box (C) in Figure 2 increases λ as follows:

$$\lambda \longleftarrow \lambda + 1 \text{ if true violation } > 1.1\tau \tag{8}$$

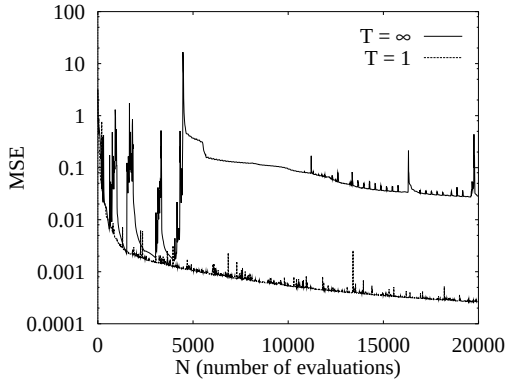


Figure 3: Progress of MSE defined in (1) for $T = 1$ and $T = \infty$ during learning of an ANN to predict the MG17 time-series.

where τ is the tolerance defined in (4) and (5). This rule penalizes a violated constraint relative to τ. We do not generate λ probabilistically because we like their effects on guidance to take place as soon as possible. The deterministic update of λ leads to the deterministic acceptance of λ in Box (D).

3.2 Probabilistic acceptances in the w subspace

Since the gradient direction computed by BP does not consider constraints due to cross validation and the step size is chosen heuristically, it is possible that a search may get stuck in infeasible local minima. In previous studies, restarts are often used to help escape from such points. However, our experimental results have shown that uncontrolled restarts may lead to loss of valuable local information collected during a search. To solve this problem, we propose an annealing strategy in Box (B) that decides whether to go from current point (w, λ) to (w', λ) according to the Metropolis probability:

$$A_T(\mathbf{w}', \mathbf{w})|_\lambda = exp\left\{\frac{(L(\mathbf{w}) - L(\mathbf{w}'))^+}{T}\right\} \tag{9}$$

where $x^+ = \min\{0, x\}$, and T is a parameters introduced to control the acceptance probability.

Figure 3 plots the progress of the mean squared errors (MSE) defined in (1) of training an ANN to predict the *Mackey-Glass*-17 time-series (in short MG17) by using two fixed temperatures: $T = 1$ and $T = \infty$, respectively. (MG17 is used as a running example throughout this section unless specified otherwise.)

When $T = \infty$ is combined with restarts, the algorithm accepts every trial point generated in the same way as traditional BP. Figure 3 illustrates this behavior by showing a search that

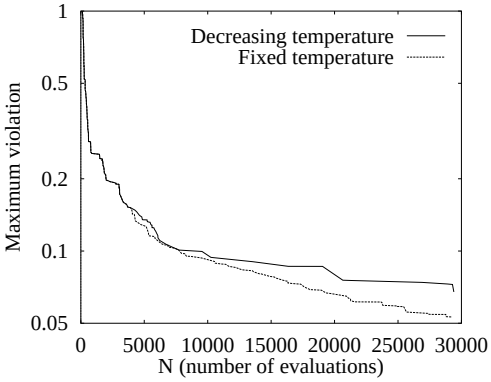


Figure 4: Decreases of maximum violation over all constraints between using a fixed temperature ($T = 5$) and using a schedule of decreasing temperatures ($T_0 = 5$, $T_{i+1} = 0.25T_i$ every 4000 evaluations, $T_\infty = 0.0003$, and η_0 was adjusted every 50 evaluations).

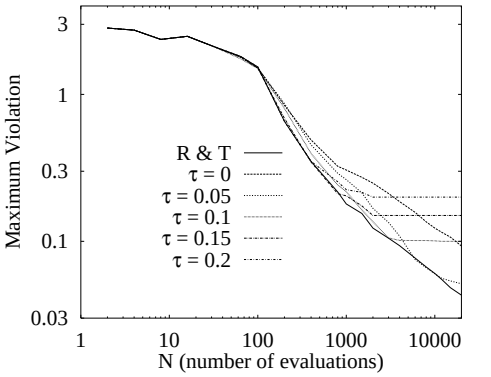


Figure 5: Decreases of maximum violation over all training patterns between using different initial violation tolerance τ (broken lines) and using our relax-and-tighten (R&T) strategy (solid line).

explores a local region in the first 4000 evaluations, got stuck in an infeasible local minimum, and restarted to a new point without keeping any history information.

On the other hand, using $T = 1$ allows the search to accept trial points according to (9) and rejects poor points with high probability. Consequently, the algorithm keeps implicitly the history information of points searched in the past and progresses smoothly without escaping into poor regions blindly.

In contrast to conventional annealing schedules that start a search at high temperatures and decrease the temperature to zero as time runs out, we use a fixed temperature throughout the search. A fixed temperature is chosen so that local descents will only be carried out by BP and not by annealing at low temperatures, and that a search will always have an opportunity to explore better regions. Figure 4 shows the improvements in maximum constraint violations when a fixed temperature is used as compared to those at low temperatures using a dynamic temperature schedule.

3.3 Relax-and-tighten (R&T) strategy

It is undesirable to set violation tolerance $\tau = 0$ initially in a search because we do not know whether such a violation tolerance can be achieved by the search. Moreover, setting $\tau = 0$ will result in considerably large violations in each pattern, leading to large λ's, a rugged search space, and a more difficult search. On the other hand, if we set a loose $\tau > 0$ initially, then most constraints can be satisfied easily, and the algorithm can focus on the few patterns with large constraint violations and increase their corresponding λ's.

Another observation is that the progress of a search differs considerably for different fixed τ's. These differences are illustrated in Figure 5 that shows the average maximum violations for different N (number of evaluations) over five independent runs. When N is small, there is little difference in maximum violations. As N is increased, runs with larger τ's have faster decreases in maximum violation than those with smaller τ's. Eventually, all the curves level off when either all constraints are almost satisfied using the specified τ or further improvement is impossible using the given ANN topology. The figure also shows a steeper rate of decrease of maximum violations with larger τ's.

Our proposed R&T strategy exploits the different convergence behavior due to different τ's by dynamically adjusting τ during a search in order to achieve the fastest convergence rate through the search. This is done by choosing a loose τ initially and by tightening $\tau \leftarrow \beta\tau$ when the maximum violation of all constraints satisfies $\max_i\{h_i(t)\} \leq (1+\gamma)\tau$, where $0 < \gamma < \beta < 1$. In this way, the search will try to use the largest possible τ at any time and will switch to a smaller τ as the convergence behavior using the original τ levels off.

Figure 5 illustrates the behavior of our proposed R&T algorithm. Initially, we set $\tau = 0.2$, leading to the steepest convergence behavior. When the convergence behavior levels off, we switch to $\tau = 0.15$ by tightening the constraints, again leading to the steepest convergence behavior for the range of N used. By repeatedly tightening constraints, the convergence behavior of R&T leads to the envelope of the best convergence behavior at all times.

The choice of the initial τ is not critical to convergence as long as it is large enough because the larger the τ is, the steeper the curve will be and the shorter the amount of time before it will level off and tighten τ. In our implementation, we set initial $\tau = 0.8 \max_i\{h_i(t)\}$ over all constraints, $\beta = 0.95$, and $\gamma = 0.1$. Around those values, convergence is not sensitive to different β's and γ's.

The R&T strategy works well on a constrained formulation of ANN learning because all constraints are defined in the same range (limited by the activation function) and all constraints have similar magnitudes. In a general constrained NLP in which constraint violations may vary in large ranges, it will be necessary but difficult to define different amount of relaxations for different constraints. As a result, R&T does not work well in solving general constrained NLPs.

3.4 Parameters in VGBP

In this section, we summarize the values of parameters used in VGBP. First, we set

$$T = \alpha N_p R, \tag{10}$$

where N_p is the number of training patterns and is known when training begins, R is the range in which ANN outputs are normalized and is set to a default value of one, and α is a constant. T should be proportional to N_p because (5) is proportional to N_p when all patterns have approximately the

Table 1: Single-step and iterative test performance in $nMSE$ on *laser*. (The test set consists of patterns from 1001 to 1100. As a comparison, we also show the performance on patterns from 1001 to 1050. Boxed numbers indicate the best results; N/A stands for data not available.)

Method	Number of weights	Training	Single-step predictions		Iterative predictions	
		100-1000	1001-1050	1001-1100	1001-1050	1001-1100
FIR network [Wan, 1993]	1105	0.00044	0.00061	0.023	0.0032	0.0434
ScaleNet: Multi-scale ANN [Geva, 1998]	N/A	0.00074	0.00437	0.0035	N/A	N/A
VGBP (Run 1)	461	0.00036	0.00043	0.0034	0.0054	0.0194
VGBP (Run 2)	461	0.00107	0.00030	0.00276	0.0030	0.0294

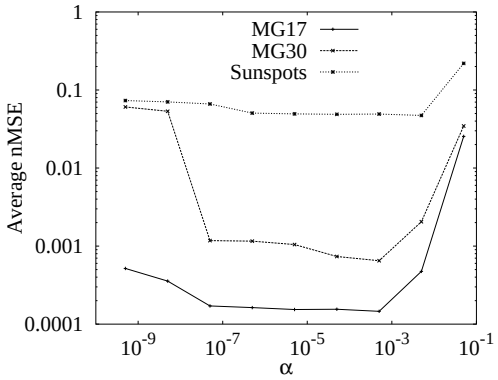


Figure 6: Robustness of VGBP with respect to α in predicting the MG-17, MG-30, and sunspots time-series.

same level of violation. Likewise, T should be proportional to R because R affects (5) in a similar manner.

Figure 6 shows the average $nMSE$'s over 10 runs of VGBP under different α's for MG17, MG30, and sunspots. Since VGBP is robust over a wide range of $\alpha \in [10^{-6}, 10^{-2}]$, we set the default α to be 10^{-3} in our implementation.

We further set η_0 in (7) to be 1.0. Since η_0 is adjusted dynamically, its initialization has no significant impact on performance. The setting of τ, β and γ in R&T has been discussed in Section 3.3.

In short, all the parameters in VGBP are set either by default or automatically, with no tuning required by users.

4 Experimental Results

We have evaluated VGBP with respect to $nMSE$ and the number of weights on several benchmarks.

Laser is a set of chaotic intensity pulsation of an NH_3 laser in the Santa Fe competition. In that competition, FIR-NN [Wan, 1993] took the first place. Table 1 shows that VGBP improves over previous algorithms in terms of prediction quality as well as number of weights used.

Sunspots contains yearly average sunspot numbers from 1700 to 1994. Using data from 1700 to 1920 for training and single-step predictions on four durations, Table 2 shows that VGBP achieves much better performance on all prediction periods, while using less weights than previous designs.

Table 3 compares single-step prediction results using VGBP with previous work on 5 chaotic time series. The two sets of *Mackey-Glass* and *Henon map* have one input and one output, whereas *Lorenz attractor* and *Ikeda attractor* have one input and two outputs as specified in [Wan, 1997] and [Aussem, 1999].

Table 2: Single-step test performance in $nMSE$ on *sunspots* for different algorithms. Results on AR(12), WNet, COMM are from [Wan, 1997], ScaleNet is from [Geva, 1998], and DRNN is from [Aussem, 1999]. Boxed numbers indicate the best results; N/A stands for data not available; n represents the number of weights/free variables used in each method.)

Method	n	Training	Single-Step Testing			
		1700-1920	1921-55	1956-79	1980-94	1921-94
AR(12)	12	0.128	0.126	0.36	0.306	0.238
WNet	113	0.082	0.086	0.35	0.313	0.219
SSNet	N/A	N/A	0.077	N/A	N/A	N/A
DRNN	30	0.105	0.091	0.273	N/A	N/A
COMM	N/A	0.079	0.065	0.24	0.188	0.148
ScaleNet	N/A	0.086	0.057	0.13	N/A	N/A
VGBP	11	0.0559	0.0337	0.0524	0.0332	0.0397

In terms of single-step predictions, Table 3 shows that VGBP uses less weights and achieves impressive $nMSE$'s one to two orders of magnitude smaller than those of other methods.

Similarly, VGBP achieves much more accurate iterative predictions as compared to those of [Wan, 1997] and [Aussem, 1999]. For example, VGBP was able to achieve iterative-prediction $nMSE$'s of 0.018 for MG17 and 0.0064 for MG30, respectively, for the duration 501-600. In contrast, applying Wan's training algorithm on FIR-NN [Wan, 1997] leads to iterative-prediction $nMSE$'s of 0.3832 for MG17 and 0.1487 for MG30. Figure 7 plots the iterative predictions of the ANN found by VGBP and that by Wan's algorithm for MG17 and MG30. The figure shows that our iterative predictions are accurate for as many as 100 steps.

In general, it may be hard to predict chaotic time series multiple steps into the future since they are unpredictable by their nature. For this reason, DRNN [Aussem, 1999] did not emphasize prediction performance, especially iterative-prediction performance [Aussem, 2001]. However, our work has shown that it is possible to predict at least the first 100 steps for the *Mackey-Glass* time series, and better for the other three sets of chaotic time series, although iterative predictions are much harder for the latter. For example, we can achieve an $nMSE$ of only 0.1369 for the first 20 steps of *Henon map*, whereas Wan's algorithm achieves an $nMSE$ of 0.6252.

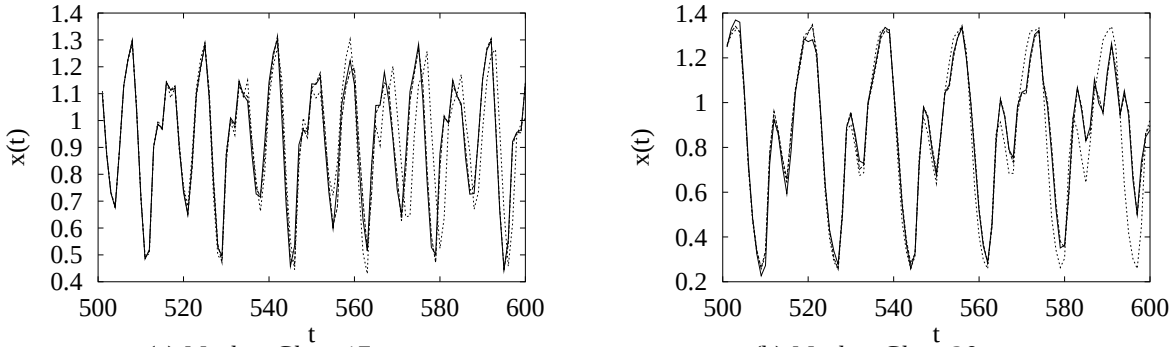
In short, constrained formulations solved by VGBP lead to superior prediction performance for the benchmarks tested.

References

[Aussem, 1999] A. Aussem. Dynamical recurrent neural networks towards prediction and modeling of dynamical sys-

Table 3: Comparison of single-step-prediction performance in $nMSE$ on five methods: Carbon copy (C.C), linear and FIR [Wan, 1993], DRNN [Aussem, 1999], and VGBP. Carbon copy simply predicts the next time-series data to be the same as the proceeding data ($x(t+1) = x(t)$). The training (resp. testing) set indicates patterns used for learning (resp. testing). *Lorenz attractor* has two data streams labeled by x and z, respectively, whereas *Ikeda attractor* has two streams – real ($Re(x)$) and imaginary ($Im(x)$) parts of a plane wave.

Bench-Mark	Training Set	Testing Set	Performance Metrics		Design Methods C.C.	Linear	FIR	DRNN	VGBP
MG17	1-500	501-2000	$nMSE$		0.6686	0.320	0.00985	0.00947	0.000057
			# of weights		0	N/A	196	197	121
MG30	1-500	501-2000	$nMSE$		0.3702	0.375	0.0279	0.0144	0.000374
			# of weights		0	N/A	196	197	121
Henon	1-5000	5001-10000	$nMSE$		1.633	0.874	0.0017	0.0012	0.000034
			# of weights		0	N/A	385	261	209
Lorenz	1-4000	4001-5500	$nMSE$	x	0.0768	0.036	0.0070	0.0055	0.000034
				z	0.2086	0.090	0.0095	0.0078	0.000039
			# of weights		0	N/A	1070	542	527
Ikeda	1-10000	10001-11500	$nMSE$	$Re(x)$	2.175	0.640	0.0080	0.0063	0.00023
				$Im(x)$	1.747	0.715	0.0150	0.0134	0.00022
			# of weights		0	N/A	2227	587	574



(a) *Mackey-Glass-17* (b) *Mackey-Glass-30*

Figure 7: Comparisons of 100-step iterative predictions on two sets of *Mackey-Glass* time-series. Solid lines represent actual data; long dashed lines indicate predicted data using VGBP; and short dashed lines are prediction results by running Wan's FIR-NN training algorithm in [Wan, 1993].

tems. *Neurocomputing*, 28:207–232, 1999.

[Aussem, 2001] A. Aussem. Personal communications, March 2001.

[Geva, 1998] A. B. Geva. ScaleNet – multiscale neural-network architecture for time series prediction. *IEEE Trans. on Neural Networks*, 9(5):1471–1482, Sept. 1998.

[Haykin, 1994] S. Haykin. *Blind Deconvolution*. Prentice Hall, Englewood Cliffs, NJ, 1994.

[Wah and Chen, 2000] B. W. Wah and Y. X. Chen. Constrained genetic algorithms and their applications in nonlinear constrained optimization. In *Proc. Int'l Conf. on Tools with Artificial Intelligence*, pages 286–293. IEEE, November 2000.

[Wah and Qian, 2000] B. W. Wah and M. L. Qian. Time-series predictions using constrained formulations for neural-network training and cross validation. In *Proc. Int'l Conf. on Intelligent Information Processing, 16th IFIP World Computer Congress*, pages 220–226. Kluwer Academic Press, August 2000.

[Wah and Wang, 1999] B. W. Wah and T. Wang. Simulated annealing with asymptotic convergence for nonlinear constrained global optimization. *Principles and Practice of Constraint Programming*, pages 461–475, October 1999.

[Wah and Wu, 1999] B. W. Wah and Z. Wu. The theory of discrete Lagrange multipliers for nonlinear discrete optimization. *Principles and Practice of Constraint Programming*, pages 28–42, October 1999.

[Wan, 1993] E. A. Wan. *Finite Impulse Response Neural Networks with Applications in Time Series Prediction.* PhD thesis, Standford University, 1993.

[Wan, 1997] E. Wan. Combining fossils and sunspots: Committee predictions. In *IEEE Int'l Conf. on Nerual Networks*, volume 4, pages 2176–2180, Houston, USA, June 1997.

Mobile Robot Learning of Delayed Response Tasks through Event Extraction: A Solution to the Road Sign Problem and Beyond

Fredrik Linåker[†,‡] **Henrik Jacobsson**[†,‡]
[†]Department of Computer Science, University of Skövde, Sweden
[‡]Department of Computer Science, University of Sheffield, United Kingdom
{fredrik.linaker, henrik.jacobsson}@ida.his.se

Abstract

We show how event extraction can be used for handling delayed response tasks with arbitrary delay periods between the stimulus and the cue for response. Our approach is based on a number of information processing levels, where the lowest level works on raw time-stepped based sensory data. This data is classified using an unsupervised clustering mechanism. The second level works on this classified data, but still on the individual time-step basis. An event extraction mechanism detects and signals transitions between classes; this forms the basis for the third level. As this level only is updated when events occur, it is independent of the time-scale of the lower level interaction. We also sketch how an event filtering mechanism could be constructed which discards irrelevant data from the event stream. Such a mechanism would output a fourth level representation which could be used for delayed response tasks where irrelevant, or distracting, events could occur during the delay.

1 Introduction

Consider an automated robot driver that navigates the streets to reach a certain goal location. On its journey, it encounters road signs, describing the upcoming junctions. Having detected a road sign, the system needs to be able to later on make the appropriate decision based on this information. That is, based on a stimulus, the system needs to store information and then make an appropriate response once the junction has been reached. This may not occur for several seconds or even minutes, depending on the vehicle's traveling speed and the distances involved. This problem was described in [Rylatt and Czarnecki, 2000], which in turn was based on a more abstract description by [Ulbricht, 1996].

The problem is in fact of a very general nature, in that it involves a *delayed response task* (Figure 1). Such tasks are quite common in real-life, involving associations between inputs and actions at different points in time. As shown by [Rylatt and Czarnecki, 2000], most existing neural network approaches are however quite inept at handling these sorts of problems. Rylatt and Czarnecki showed that, in fact, even their own, for the task specially constructed, recurrent neural network architecture was unable to learn the appropriate associations if they lay more than just a few time-steps apart.

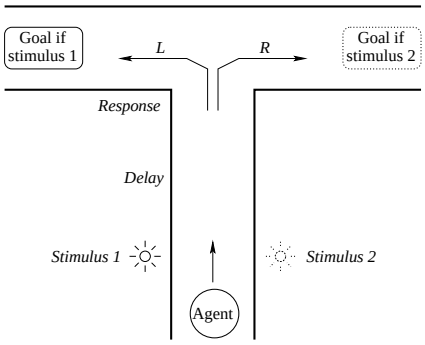


Figure 1: The delayed response task, adapted from [Ulbricht, 1996]. The robot travels past a stimulus, here a light either on the left or the right side, then continues down the corridor for a number of steps until it reaches the junction at which time the robot needs to decide whether it should turn left or right, depending on the location of the light it passed earlier.

In this paper, we present a quite different approach from Rylatt and Czarnecki, in that we do not work directly on the input sequence but instead let an unsupervised system extract a set of *events* from the inputs and then we work on this sequence of events. As we will show, the time intervals between events may in fact be arbitrarily large without affecting the performance of the system; a drastic change from previous approaches. We also describe a more complex situation, the 'Extended Road Sign Problem', which involves having distractions occur during the delay period. Our solution is based on having several information processing levels, as depicted in Figure 2.

Level 1: Contains raw multi-dimensional sensor data, which is time-step based[1]. This is the level where [Rylatt and Czarnecki, 2000] approached the delayed response task. We however argue, like [Nolfi and Tani, 1999], that real-world

[1]Where a time-step is defined as a single update of sensors, neuronal elements, and actuators with a regular time interval, typically in the millisecond range.

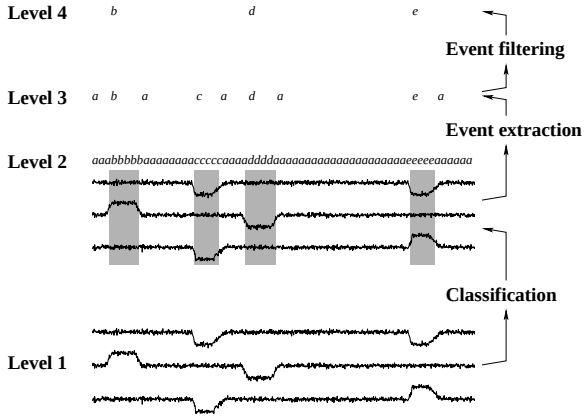


Figure 2: The proposed four-level information processing architecture which can handle delayed response tasks with very long-term dependencies.

task dependencies do most often not manifest themselves at this level of individual time-stepped neuronal updates, but rather on much slower time scales, involving several seconds or even minutes. Rylatt and Czarnecki's somewhat simplified simulations and specially modified neural network architecture was only able to learn dependencies which lay up to 13 time-steps apart. But as most robot systems have a high sampling frequency, Rylatt and Czarnecki's system would not be sufficient. For example, the Khepera robot we use in our experiments, has sensor sampling rates of approximately 20 times per second. Rylatt and Czarnecki's system would therefore, in effect, not be able to learn tasks involving even just single second delays. In the following, we will show that by using event extraction, the delays can instead be arbitrarily large and this enables the system to handle more realistic long-term dependencies.

Classification: The process whereby the raw multi-dimensional sensor data is divided into a set of classes. While [Nehmzow and Smithers, 1991] employed a set of manually pre-defined (fixed) classes for this, [Tani and Nolfi, 1998] instead let the system determine the class structure by itself, thereby reducing the user intervention. However, Tani and Nolfi still had to manually specify the number of classes, and had to divide the training into several different phases. They also had large problems with inputs which were distinct but not very frequent in the training set, and the learning process was very slow. Recently, [Linåker and Niklasson, 2000] have constructed a more flexible classification system, the ARAVQ, which is able to swiftly classify inputs into a dynamic number of automatically extracted classes, overcoming most of the problems in Tani and Nolfi's system. (The ARAVQ system is the one used in the following simulations.)

Level 2: Contains raw time-step based multi-dimensional data which has been tagged with class labels. If each class is alloted a character in the alphabet, the input can be rewritten, with some loss of information, as a (very long) letter sequence. This is the level at which [Ulbricht, 1996] approached the road sign problem, trying to learn associations between the letters in the sequence. She did not, however, provide any account for how the input had become this letter sequence (i.e. no Level 1 nor any classification), thereby working on essentially *ungrounded* symbols. And, further, as her system was still time-step based, she had the same difficulties learning long-term dependencies as Rylatt and Czarnecki had in their Level 1 system.

Event extraction: The process whereby only the transitions between class membership of the lower level data are extracted, thereby filtering out repetitions. This is a fairly straightforward mechanism as long as the class membership is exclusive (each input belonging to one—and only one—class); if two succeeding inputs are classified differently, an event is generated. The detection of an event generates a signal to the next level.

Level 3: Contains general events, interspersed over long periods of time. Updates occur on a considerably slower time-scale than the time-step based levels below. This means that longer time-dependencies can be detected, as noted by [Tani and Nolfi, 1998]. While Tani and Nolfi's robot system worked at this level, it did not involve any coupling back to the real-world as the input was merely classified and filtered, and not acted upon in any manner. That is, their system did not use the extracted events to control the robot; it was only an idle observer of what was going on. We here provide an account for how extracted events can be used to learn delayed response tasks and also how this can be used to identify candidates which should pass through an event filtering on to yet another level.

Event filtering: The process which discards events which are considered as irrelevant for accomplishing the task. This process requires that the events have been rated with some sort of 'usefulness' score, related to how relevant they are for achieving the task. This event rating should ideally be based on a delayed reinforcement learning system, such as Q-learning, as rewards in the real world do often not come immediately as an action is performed. In the following we provide a simpler but less realistic evaluation mechanism, based on a recurrent neural network which has learnt a simple supervised version of the task at Level 3 (see Section 4). The idea behind this is that once a simple version of the task has been learnt on a low level, it can be generalized to more complex situations on higher information processing levels that have access to longer time horizons.

Level 4: Contains only the events which are considered as relevant for achieving the task. It is updated on an even slower time-scale than Level 3 and thus can handle events that have occurred even further apart.

It is worth noting that from Level 2 and upwards, the system can work on an essentially symbolic representation of the input sequence. These symbols provide a representation whose size is virtually independent from the dimensionality of the actual sensory and motor systems. This relaxes the information processing and storage demands on the system, assuming that the symbolic representation is more compact than its corresponding input, which usually is the case.

The next section describes an example of how a simple system can be constructed, based on straightforward building blocks, in order to achieve the discussed information processing.

2 Architecture

In addition to the information processing capabilities described in the previous section, the system needs to be able to control the robot, i.e. making the appropriate response once the cue for responding comes. There are several possibilities of executing actions. Levels 1 and 2 are both time-step based, which means that the inputs can be directly coupled to a single action which lasts a proportionate single time-step. This can accommodate for simple, non-goal-oriented, and/or 'innate' reflex actions.

However, associations between inputs at Levels 3 and upwards are based on events. These events can occur at widely interspersed points in time. A single time-step action is therefore not appropriate as a response; the response should ideally affect the performance the entire time interval up to the next event. A simple solution is to repeatedly execute the *same* action until the next event occurs, e.g. to keep turning in one direction until the next event occurs. This would however be a very rigid and inflexible solution, not allowing the system to modify its responses into smoother real-time interactions.

Instead, we propose, that the event-based levels affect, or modulate, the actual input-to-output (sensation-to-action) *mapping* of the time-step based levels. This modulation would also give the system the ability to focus on particular sensor subsets which are of most importance to the particular response. We show this by manually constructing a set of very simple input-to-output mappings, or *behaviours*, which the event-based levels can choose between. Each behaviour only works on a subset of the available sensory channels. As the higher levels themselves do not act directly on the actuators, there is also no need for between-level action selection, something which has caused problems in other layered control architectures such as Brooks' well-known Subsumption Architecture. Our architecture is summarized in Figure 3.

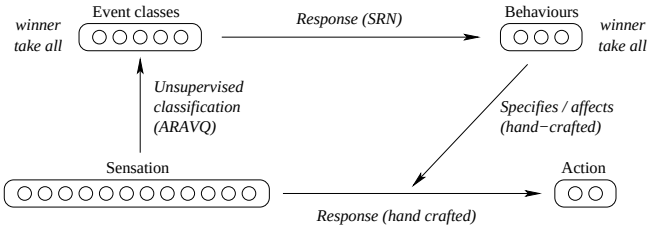


Figure 3: The architecture realizing information processing levels 1 through 3 in Figure 2, including the means for controlling the agent's actions.

Inputs from the robot sensors were fed through an ARAVQ network (Section 2.1), which classified the inputs into a set of classes. When the classification became different for two succeeding time-steps, an event was generated and a localistic representation of the winner was fed into a Simple Recurrent Network (SRN, see Section 2.2), which learnt associations between inputs (events) and outputs (behaviours) at different time points. The SRN specified which of the behaviours (Section 2.3) the robot was to employ until the next event occurred.

2.1 The ARAVQ

The adaptive resource allocating vector quantization (ARAVQ) network [Linåker and Niklasson, 2000] is a vector quantization network which contains a set of model vectors, representing event classes. When a series of novel and stable inputs are encountered, the system dynamically incorporates additional model vectors. The number of allocated model vectors is determined by the characteristics of the input signal, which in turn reflects the characteristics of the environment, or the agent-environment interaction, that underlies the sensory flow which we apply the ARAVQ network to here. It is also biased by the ARAVQ's parameter settings.

The ARAVQ has four user-defined parameters: a novelty criterion δ, a stability criterion ϵ, an input buffer size n and a learning rate α. These are all explained below. In order to cope with noisy inputs, the ARAVQ filters the input signal using the last n input vectors, which are stored in an input buffer $X(t)$. The values in the input buffer are averaged to create a more reliable, *filtered*, input $\overline{x}(t)$ to the rest of the network. That is, a finite moving average $\overline{x}(t)$ is calculated for the last n time-steps.

There is a set $M(t)$ of model vectors (each one representing an event class), which is initially empty. (The ARAVQ does not start working until the input buffer is filled, i.e, until time-step $n-1$.) Additional model vectors are only allocated when novel and stable inputs are encountered, i.e., when the following criteria are fulfilled:

- The input is considered as *novel* if the Euclidean distance between the existing model vectors $M(t)$ and the last n inputs, compared to the distance between the moving average $\overline{x}(t)$ and the last n inputs is larger than the distance δ.
- The input is considered *stable* if the difference between the actual inputs $x(t), x(t-1), \ldots, x(t-n+1)$ and the moving average $\overline{x}(t)$ is below the threshold ϵ.

For convenience, we define the following general distance measure between a set of (model/filtered input) vectors V and a set of actual inputs X:

$$d(V,X) = \frac{1}{|X|}\sum_{i=1}^{|X|} \min_{1\leq j\leq |V|} \{||x_i - v_j||\}; x_i \in X, v_j \in V, \tag{1}$$

where $||.||$ denotes the Euclidean distance measure. The distance between the filtered input and the actual inputs is defined as:

$$d_{\overline{x}(t)} = d(\{\overline{x}(t)\}, X(t)), \tag{2}$$

and the distance between the existing model vectors and the actual inputs is:

$$d_{M(t)} = \begin{cases} d(M(t), X(t)) & |M(t)| > 0 \\ \epsilon + \delta & \text{otherwise.} \end{cases} \tag{3}$$

Event Class Incorporation: If both the stability and novelty criteria are met, the filtered input is incorporated as an additional model vector:

$$M(t+1) = \begin{cases} M(t) \cup \overline{x}(t) & d_{\overline{x}(t)} \leq min(\epsilon, d_{M(t)} - \delta) \\ M(t) & \text{otherwise.} \end{cases} \tag{4}$$

Classification: Each time-step, a winning model vector $win(t)$ is selected, indicating which class the (filtered) input currently matches:

$$win(t) = \arg \min_{1 \leq j \leq |M(t)|} \{||\overline{x}(t) - m_j||\}; m_j \in M(t). \quad (5)$$

Adaptation: If the winning model vector matches the (filtered) input very closely, the (filtered) input is considered to represent a 'typical' instance of the class, and the model vector is modified to match the input even closer:

$$\Delta m_{win(t)} = \begin{cases} \alpha[\overline{x}(t) - m_{win(t)}] & ||\overline{x}(t) - m_{win(t)}|| < \frac{\epsilon}{2} \\ 0 & \text{otherwise,} \end{cases} \quad (6)$$

where α is a user-defined learning rate. The structure of the ARAVQ network is depicted in Figure 4.

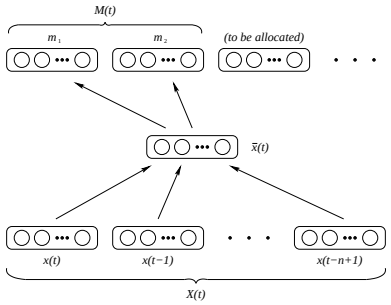


Figure 4: The ARAVQ network. The n last inputs are buffered and used to calculate a filtered input $\overline{x}(t)$. This particular network has allocated two model vectors, m_1 and m_2; additional model vectors will be allocated automatically when novel and stable inputs are encountered. (In the following, model vectors are for convenience labeled a, b, c, etc. instead of m_1, m_2, m_3, etc.)

2.2 The SRN

The simple recurrent network (SRN) [Elman, 1990] is essentially a three-layer feed-forward network which stores a copy of the previous hidden activation pattern and feeds it as additional input at the next time-step. This provides a memory trace of previous inputs which enables the network to learn associations over several time-steps.

The output of the SRN (for an arbitrary number of nodes) is defined as:

$$o_k(t) = f(\sum_j w_{kj} h_j(t) + w_{k\theta}) \quad (7)$$

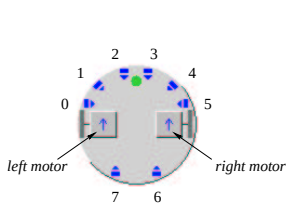
where the hidden activation is defined as

$$h_j(t) = f(\sum_i v_{ji} i_i(t) + \sum_m v_{jm} h_m(t-1) + v_{j\theta}) \quad (8)$$

and $i(t)$ is the input at time t. The index k is used for identifying the output nodes, j and m are used for the hidden nodes (at time t and $t-1$ respectively), and i is used for the input. Biases are introduced as additional elements in the weight matrices, w and v (indexed with θ). The function, f, is the sigmoid activation function $f(x) = 1/(1+e^{-x})$.

2.3 Behaviours

Three different input-to-output mappings, or behaviours, were constructed: a corridor follower, a left wall follower, and a right wall follower. Each behaviour only needed to use a subset of the available sensor readings; see Figure 5. The Khepera robot which was used has eight infrared proximity sensors with integer activation in the range $[0, 1023]$, 0 denoting no object present within sensor range and 1023 denoting an obstacle very close. The robot has two separately controlled wheels which can be set to integer values in the range $[-10, 10]$, -10 denoting maximum backward spinning, 0 no wheel movement, up to 10 which rotates the wheel forward at maximum speed.



(a)

```
/* Corridor follower */
if (sensor[4] > 800) {
  motor[LEFT] = 0;
  motor[RIGHT] = 5;
} else if (sensor[1] > 800) {
  motor[LEFT] = 5;
  motor[RIGHT] = 0;
} else {
  motor[LEFT] = 5;
  motor[RIGHT] = 5;
}
```

(b)

```
/* Left wall follower */
if (sensor[1] > 200) {
  motor[LEFT] = 5;
  motor[RIGHT] = -4;
} else if (sensor[0] < 800) {
  motor[LEFT] = 2;
  motor[RIGHT] = 5;
} else {
  motor[LEFT] = 5;
  motor[RIGHT] = 5;
}
```

(c)

```
/* Right wall follower */
if (sensor[4] > 200) {
  motor[LEFT] = -4;
  motor[RIGHT] = 5;
} else if (sensor[5] < 800) {
  motor[LEFT] = 5;
  motor[RIGHT] = 2;
} else {
  motor[LEFT] = 5;
  motor[RIGHT] = 5;
}
```

(d)

Figure 5: The Khepera robot and the hand crafted behaviours.

In addition to the three hand crafted behaviours, a simple selection mechanism was implemented. At each time-step, it detected the most active output node of the SRN and executed the code associated with the behaviour.

3 Experiments

A simulated version of the Khepera robot was used in the experiments. The activation of the eight distance sensors, the two motors, and two of the robot's light sensors, placed in concert with distance sensors 0 and 5 in Figure 5(a), were normalized to the range $[0.0, 1.0]$ and fed as input to the architecture. That is, the ARAVQ network received a total of 12 inputs. The parameter settings of the ARAVQ were $\delta = 0.7$, $\epsilon = 0.2$, $n = 10$ and $\alpha = 0.05$. This led to the extraction of eight different event classes for the constructed T-maze environments shown in Figures 6 and 7; each event class was allocated a separate shade and at each location, the winner of the classification process was plotted, leaving the trail shown in the figure.

The event extraction discards the repetitions of each class, leaving sequences which are only six characters long. A character was manually assigned to each of the eight extracted classes, for presentational purposes. An interpretation of each

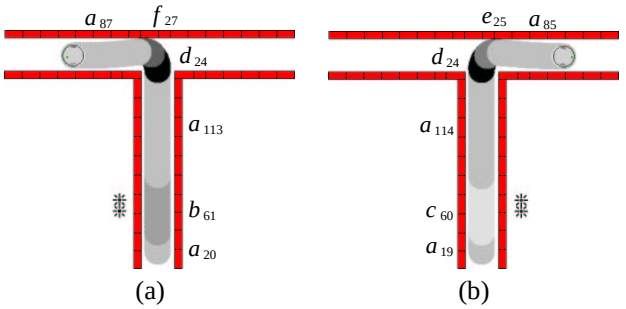


Figure 6: The two different cases for the delayed response task and the resulting input classification at each location along the simulated robot's path, the subscript denoting the number of repetitions of each class.

class is shown in Table 1, based on how the ARAVQ seems to apply the classes.

model vector	*interpretation*
a	corridor
b	corridor + left light
c	corridor + right light
d	junction
e	wall on left side only
f	wall on right side only
g	left-turning corner
h	right-turning corner

Table 1: The eight automatically extracted model vectors and how they can be interpreted.

Each of the three behaviours was also assigned a character, this time creating an 'output alphabet', as shown in Table 2.

behaviour	*description*
C	corridor following
L	left side wall following
R	right side wall following

Table 2: The three hand crafted behaviours.

The SRN worked on this eight-character input and three-character output alphabet. As two hidden nodes were used, a 8-input, 2-hidden, 3-output SRN was created. Network weights were randomly initialized in the interval $[-5.0, 5.0]$, the learning rate was 0.01 and a momentum of 0.8 was used. The training set was the two extracted sequences, for the left and right turn, respectively, as shown in Table 3. The network was trained for $10,000$ epochs using back-propagation through time (BPTT) which *unfolds* the recurrent connections of the network. The BPTT here unfolded the network 5 times.

The SRN had no problems learning the correct associations

case	*sequence*
Left turn	$\begin{matrix} a & b & a & d & f & a \\ C & C & C & L & C & C \end{matrix}$
Right turn	$\begin{matrix} a & c & a & d & e & a \\ C & C & C & R & C & C \end{matrix}$

Table 3: The extracted sequences of model vector winners and the behaviours that should be selected for the paths taken in Figure 6.

between the light stimuli b and c and the behaviours L and R occurring two events later. That is, the system could turn in the right direction, irrespective of the length of the delay, as this information was removed in the event extraction. The Road Sign Problem was thus solved. We now, however, have a similar problem to that of [Rylatt and Czarnecki, 2000], but on the level of intermediate *events* instead of intermediate time-steps. We call this problem 'The Extended Road Sign Problem'.

4 The Extended Road Sign Problem

While the length of the delay between the stimulus and the response has become irrelevant through the use of event extraction, the system would still have problems handling distracting events during the delay. That is, if the input changes drastically during the delay, intermediate events will be generated. This means that the problem of finding relationships between the stimulus and the subsequent response will become harder as there are a number of distracting events which have taken place in between. Examples of this are shown in Figure 7 where there is a right or left turn in the corridor after the stimulus has been passed, generating another three events which however are of no relevance for the task, i.e. they serve only as distractions.

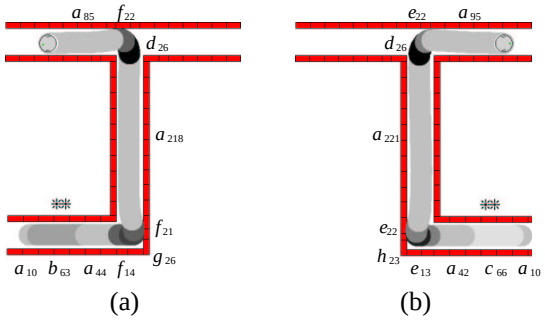


Figure 7: Two examples of distracting events (turns) happening in between the stimulus (light) and the cue (junction).

The SRN still managed to find the correct association, but only if it was first trained on the simpler tasks, and then continued training on the more difficult examples shown above. The hidden node activation plot of an SRN which has learnt

the task is depicted in Figure 8. Note that a number of clusters have formed for each of the functional states the robot can be in.

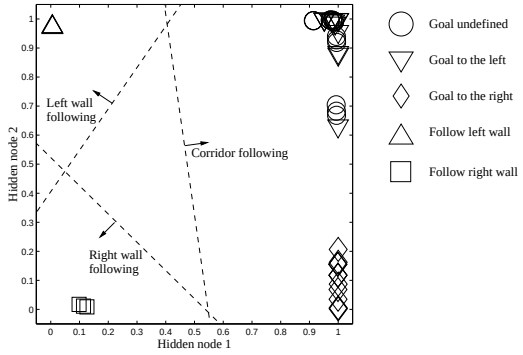


Figure 8: Hidden node activation of the SRN.

When irrelevant inputs, such as corners (input characters g and h), are received by the SRN, the state remains relatively stable, i.e. it stays in the same location as the previous time-step. That is, large movements in the internal (hidden node) activation space occur only when *functionally important* events occur. When, for instance, input character b (left stimulus) is received, the activation jumps to the upper right corner and stays there until the input character d (the cue for responding) arrives. At that time, the activation quickly jumps to the upper left corner, making the robot activate its left wall following behaviour, effectively starting to turn left at the junction. A similar situation occurs if instead the other stimulus is present, where the bottom right and left corners are used by the SRN. (Before either stimuli has been encountered, the SRN state is in the upper right corner, i.e. this particular robot has a bias for turning left at junctions.)

We can now also sketch how a solution to the Extended Road Sign Problem might look. Note that if the hidden activation space of this SRN was clustered, using for example another ARAVQ network, the functionally unimportant events would likely cause repetitions of the same class winner as they would lead to only small perturbations in the state space. Only when functionally important events occur, such as the stimulus or cue, do large jumps in activation space occur, thereby leading to another class perhaps becoming the best match. Then using the same filtering process used for repetitions of input patterns, the irrelevant events would be removed. Note that this solution is based on a Level 3 system (SRN) which has already successfully learnt the associations in a relatively simple scenario. The filtering can then extract information which effectively lets the next higher level (Level 4) handle delays with an *arbitrary* number of distracting events as they will be filtered out in the aforementioned process.

5 Conclusions

We have shown how a layered information processing system can be constructed which handles delayed response tasks like the Road Sign Problem [Rylatt and Czarnecki, 2000]. Our solution is based on attacking the problem at a higher level of abstraction than the raw time-step based sensory data. As shown, the learning system (in this case an SRN) has a considerably easier task when the redundant data has been filtered out, letting the system instead work on a sequence of discrete events. These events are grounded in sensori-motor interactions. As discussed, the event extraction provides the means for handling *arbitrarily* long delays between the stimulus and the subsequent cue for response.

In addition to handling the Road Sign Problem, we suggest an Extended Road Sign Problem. This involves distractions during the delay, thereby putting further demands on the system not to lose track of what it is supposed to do. As discussed, another abstraction level, which works on *filtered* event streams, could be added. This filtering would be task-specific and would be based on that the system has already learnt, on a simple version of the task, which of the inputs are relevant. We believe this approach will steer us in the right direction on the road to acquiring more complex and intelligent behaviours from our robotic friends.

Acknowledgments

This research was funded by a grant from the Foundation for Knowledge and Competence Development (1507/97), Sweden and the University of Skövde, Sweden.

References

[Elman, 1990] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[Linåker and Niklasson, 2000] F. Linåker and L. Niklasson. Time series segmentation using an adaptive resource allocating vector quantization network based on change detection. In *Proc. of the Int. Joint Conf. on Neural Networks*, volume VI, pages 323–328. IEEE Computer Society, 2000.

[Nehmzow and Smithers, 1991] U. Nehmzow and T. Smithers. Mapbuilding using self-organising networks in really useful robots. In *Proc. of the First Int. Conf. on Sim. of Adaptive Behavior*, pages 152–159. MIT Press, 1991.

[Nolfi and Tani, 1999] S. Nolfi and J. Tani. Extracting regularities in space and time through a cascade of prediction networks. *Connection Science*, 11(2):125–148, 1999.

[Rylatt and Czarnecki, 2000] R.M. Rylatt and C.A. Czarnecki. Embedding connectionist autonomous agents in time: The 'road sign problem'. *Neural Processing Letters*, 12:145–158, 2000.

[Tani and Nolfi, 1998] J. Tani and S. Nolfi. Learning to perceive the world as articulated. In *Proc. of the Fifth Int. Conf. on Sim. of Adaptive Behavior*, pages 270–279. MIT Press, 1998.

[Ulbricht, 1996] C. Ulbricht. Handling time-warped sequences with neural networks. In *Proc. of the Fourth Int. Conf. on Sim. of Adaptive Behavior*, pages 180–189. MIT Press, 1996.

NORN Finance Forecaster - A Neural Oscillatory-based Recurrent Network for Finance Prediction

Raymond S. T. Lee and James N. K. Liu

Department of Computing, Hong Kong Polytechnic University

Hung Hom, Hong Kong

csstlee@comp.polyu.edu.hk, csnkliu@comp.polyu.edu

Abstract

Financial prediction is so far the most important applications in contemporary scientific study. In this paper, we present a fully integrated stock prediction system – NORN Finance Forecaster – A Neural Oscillatory-based Recurrent Network for finance prediction system to provide both a) Long-term trend prediction, and b) Short-term stock price prediction. One of the major characteristics of the proposed system is the automation of the conventional financial technical analysis technique such as market pattern analysis via NOEGM (Neural Oscillatory-based Elastic Graph Matching) model and its integration with the Time-difference recurrent neural network model. This will provide a fully integrated and automated tool for analytic and investigation of stock investment. From the implementation point of view, the stock pricing information of 33 major Hong Kong stocks in the period of 1990 to 1999 are being adopted for system training and evaluation. As compared with contemporary neural prediction model, the proposed system has achieved challenging results in terms of efficiency and accuracy.

1. Introduction

Financial prediction such as stock prediction and foreign currency exchange rate forecast so far is one of the most important and hottest topics, both in the scientific and financial fields. In fact, due to the importance of this particular topic, a well-established school of concepts and techniques have been devised in the previous decades, namely the technical [Murphy, 1986] and fundamental analysis [Ritchie, 1996] techniques. However, owing to the fact that these tools are based on totally different approaches of analysis, they always give rise contradictory results. Besides, 'pattern matching' technique is still widely used for most financial analysts and market technicians. Actually, there are numerous market patterns that have found (and believe) to exhibit repeated market moves following their occurrence. So they are especially useful for long-term financial trend prediction. These market patterns can be classified into two major categories: 'reversal patterns' and 'continuation patterns' [Zirilli, 1997]. The correct identification of these patterns is a very useful information to predict the financial trend movement. However, the identification of these patterns so far is highly subjective and prone to errors. The major obstacles for the automation of this pattern matching process is that these patterns are highly variant and 'elastic' in the sense that they can exist under different appearances.

In this paper, we present an integrated neural network-based financial prediction system which fully automates the process of 1) long-term trend prediction and 2) short-term (e.g. one-day) stock forecast. Unlike the contemporary neural network-based financial prediction model, we have integrated and automated critical technical analysis tools into the proposed model, namely the market pattern identification scheme in trend prediction and oscillatory-based RSI (relative strength index) analysis technique. From the implementation point of view, stock information for 33 major Hong Kong stocks in the period from 1990 to the end of 1999 (ten-year data) have been used. Compared with other contemporary time series neural network prediction models including the feedforward backpropagation model from NeuroForecaster and Genetica, NORN has achieved challenging results in terms of efficiency, accuracy and the ability to integrate critical technical analysis techniques.

2. NORN: system framework

2.1 System overview

The neural oscillatory-based recurrent network model (NORN) proposed in this paper is based on the integration of critical technical analysis techniques 1) Market pattern analysis and 2) Oscillator-based system with hybrid RBF (Radial Basis Function) recurrent network for stock prediction. NORN basically consists of the following modules:

- Automatic market analysis module using NOEGM model;
- Hybrid RBF recurrent network with TDSL (Time-Difference Structural Learning) for stock prediction.

The major objective of NORN on one hand is the provision of a fully automatic stock prediction model (based on an integrated recurrent neural network), and on the other hand, to integrate it with critical technical analysis techniques to increase the degree of accuracy.

2.2 Market pattern analysis via NOEGM

NOEGM – system overview

The Neural Oscillatory Elastic Graph Matching (NOEGM) model consists of three main modules: 1) Multi-frequency bands feature extraction for the stock pattern using Gabor filters; 2) Automatic figure-ground TC pattern segmentation using Neural Oscillatory model; 3) Invariant market pattern matching using Elastic Graph Dynamic Link Model (EGDLM).

Actually NOEGM model has been applied to various invariant pattern recognition applications including face recognition [Lee *et al.*, 1999], scene analysis [Lee and Liu, 1999a] and the latest research on tropical cyclone (TC) pattern recognition from the satellite pictures [Lee and Liu, 1999b; 2000], based on composite neural oscillators as neural framework. A schematic diagram for the NOEGM model is shown in Figure 1.

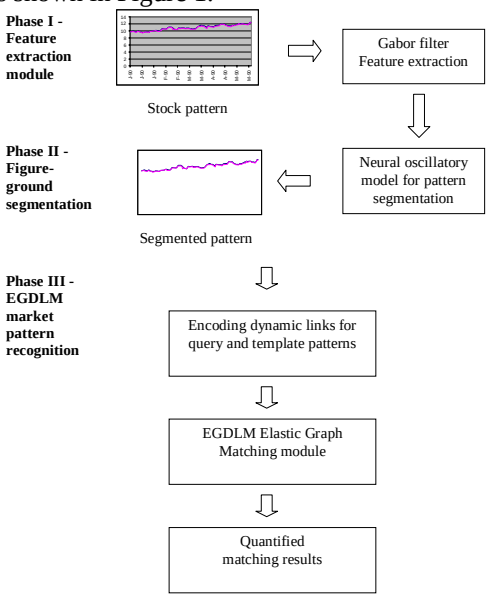


Figure 1 – Schematic diagram for the NOEGM model for market pattern recognition

Feature extraction module

In this module, Gabor filters of 10 different frequency bands (ϕ) and 5 different orientations (θ) are being used, a total of 50 feature vectors of different attributes are extracted from the stock patterns. The filter function is given:

$$g_{\phi,\theta}(x,y)=\frac{1}{\sigma\sqrt{\pi}}e^{(-\frac{x^2+y^2}{2\sigma^2})}e^{2\pi i\phi(x\cos\theta+y\sin\theta)} \quad (1)$$

Pattern segmentation – neural oscillatory model

In the neural oscillatory model, neurons in our visual cortex are presented by numerous neural oscillators. A typical neural oscillator consists of an excitatory neuron (u_i) and an inhibitory neuron (v_i), which interacts and oscillates with other neurons according to external stimulus. In the model, each layer represents the whole visual horizon in 2D perspective. In each location, the "column" neural oscillators represent the set of "visual sensors", which is modeled by an array of feature vectors.

According to the model, with the presence of a subject in an image (eg. stock pattern), neurons that belong to the same segment are oscillated in a phase-locked mode with zero phase shift after sufficient cycles of neural oscillations, while neurons in different segments are oscillated out of phase and uncorrelated [Lee and Liu , 1999a].

In the neural oscillatory model, each stock pattern graph is broken into a 2D mesh of $N = N_1 \times N_2$ composite neural oscillator sites. Each site (column) consists of M layers of neural oscillators which denote the neural oscillation from each local feature response (e.g. using Gabor Filters). In addition to the local excitatory and inhibitory neurons, the neural dynamics of the composite neural oscillators are activated/deactivated by the oscillatory model.

"Vertical & horizontal" excitatory connections

Neural oscillators within the same "column" are mutually activated with strength $W_{|}$. In each feature layer, each neural oscillator is activated by the eight "closest" neighboring excitatory composite neurons with strength $W_{\_}$.

"Vertical & horizontal" inhibitory neurons (v_i & v^q)

To control the unexpected local phase locking, "Vertical" and "Horizontal" inhibitors are introduced, which are governed by the inhibitory strength $T_{|}v_i$ & $T_{\_}v^q$ respectively.

"Global" inhibitory neurons (v)

A global inhibitory neuron which receives excitation from all excitatory neural oscillators, and it inhibits all excitatory units with the signal T_v. The neural dynamics are shown as follows:

$$\frac{du_i}{dt}=-u_i^q+S_\lambda(u_i^q-\beta v_i^q-\theta_u+I_{ci}^q+W_{i_|}^q u_i^q+W_{i_\_}^q u_i^q-T_{i_|}^q v_i-T_{i_\_}^q v^q-T_i^q v) \quad (2)$$

$$\tau\frac{dv_i^q}{dt}=-v_i^q+S_\lambda(\alpha u_i^q-\gamma v_i^q-\theta_v) \quad (3)$$

$$\tau\frac{dv_i}{dt}=-v_i+S_\lambda(W_{i_|}v_i-\theta_v) \quad (4)$$

$$\tau\frac{dv^q}{dt}=-v^q+S_\lambda(W_{\_}^q v^q-\theta_v) \quad (5)$$

$$\tau\frac{dv}{dt}=-v+S_\lambda(W^{'}v-\theta_v) \quad (6)$$

C) Segmentation criteria: correlation function σ_{xy}

The segmentation criteria is governed by the "Correlation Factors", which are a measurement of the binding strength (phase relationship) between the composite neural oscillators and their "nearest" neighbors, given by:

$$\sigma(\bar{x},\bar{y})=\frac{<\bar{x}\,\bar{y}>-<\bar{x}><\bar{y}>}{\sqrt{<\bar{x}^2>-<\bar{x}>^2}\cdot\sqrt{<\bar{y}^2>-<\bar{y}>^2}} \quad (7)$$

Market pattern recognition – elastic graph dynamic link model (EGDLM)

Object recognition based on EGDLM which described the recognition problem as an elastic graph matching mechanism between the attribute graphs of the image vectors (input layer) with the set of "memory" graphs in the object gallery (memory layer). The schematic diagram of

EGDLM network architecture for market pattern matching is shown in Figure 2.

One of the most striking features of the EGDLM is its "invariant" property. In the network model, only the topological relations between the neural oscillators (i.e. dynamic links) are encoded into the network. The pattern matching process is resembled to the 'Elastic Graph Matching', which is invariant under various transformations such as translation, rotation, reflection, dilation and occlusion, commonly occurred in pattern recognition.

EGDLM object recognition model consists of two processes: 1) dynamic link initialization and 2) elastic graph matching module. In dynamic link initialization process, dynamic links ($z_{ij,kl}$) between "memory" object graphs (obtained from market pattern templates) and query pattern are initialized according to the following rules:

$$z_{ij,kl} = \varepsilon J_{ij} J_{kl} \tag{8}$$

for $J_{ij} \in A$, $J_{kl} \in B$.

In the elastic graph matching module, attribute graph of the figure pattern (with neural oscillator vectors as nodes and correlation links as edges) is matched with each of the attribute graphs in the object gallery by minimizing energy function H(z):

$$H(z) = -\sum_{i,j\in B;k,l\in A} z_{ij} z_{jl} z_{ik} z_{kl} + \gamma \sum_{i\in B}\left(\sum_{k\in A} z_{ik} - 1\right)^2 + \gamma \sum_{k\in A}\left(\sum_{i\in B} z_{ik} - 1\right)^2 \tag{9}$$

where H(z) is minimized using the gradient descent.

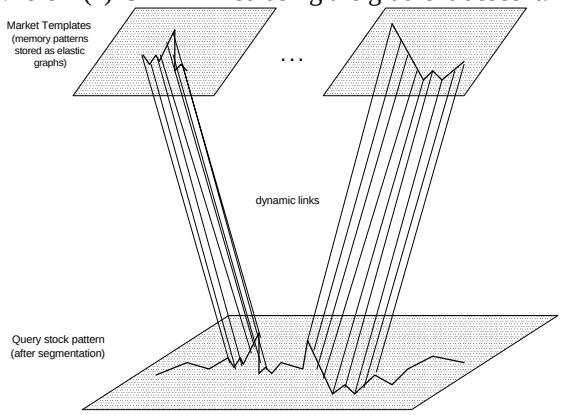


Figure 2 – EGDLM Network architecture for market pattern matching

2.3 Hybrid radial basis function networks for nonlinear time series stock prediction

Hybrid RBF network (HRBFN) – system overview

The proposed Hybrid RBF Network (HRBFN) [Lee and Liu, 2000] incorporates with two main technologies into the conventional RBF network for temporal time series prediction problem: 1) Structural learning technique that integrates the "forgetting" factor into the RBF BP algorithm [Ishikawa; 1996]; 2) A 'Time Difference with Decay' (TDD) method is corporated into the network to strengthen the temporal time series relation of the input data sequence for network training.

The HRBFN consists of three layers. The first layer is the input layer which consists of two portions: 1) Past network outputs that feedback into the network; 2) Major co-relative variables are concerned with the prediction problem. Past network outputs enter into the network by time-delay unit as the first inputs. These outputs are also affected by a decay factor γ that is governed by the following equation.

$$\gamma = \alpha e^{-\lambda k} \tag{10}$$

In general, the time series prediction of the proposed network is to predict the outcome of the sequence x_1^{t+k} at the time of t+k that is based on the past observation sequence of size n, i.e. x_1^t, x_1^{t-1}, x_1^{t-2}, x_1^{t-3},..., x_1^{t-n+1} and the major variables that influence the outcome of the time series at time t. For convenience, the following notations are used throughout the following network description: The numbers of input nodes in the first and second portions are set to n and m respectively. The number of hidden nodes is set to p. The predictive steps are set to k, so the number of output nodes is k. At the time t, the input will be $[x_1^t, x_1^{t-1}, x_1^{t-2}, x_1^{t-3},\ldots,x_1^{t-n+1}]$ and $[x_2^1, x_2^2,\ldots,x_2^m]$ respectively. The output is given by x^{t+k}, denoted by p_k^t for simplicity, w_{ij}^t denotes the connection weight between the i-th node and the j-th node at time t.

HRBFN – structural learning algorithms

The main idea of RBF learning algorithm with "forgetting" factor is to introduce a constant decay to connected weights that make the redundancy weight(s) fade out quickly. The cost function of the structural learning algorithm is given by equation (11).

$$E^t = E_1^t + \varepsilon' \sum_{i,j} \left| w_{ij}^t \right| \tag{11}$$

where E_1^t denotes the error square in traditional RBF learning, the second term is the penalty criteria.

If delta rule is used, the learning rule of the weights is given by:

$$\Delta w_{ij}^{t+1} = -\eta \frac{\partial E^t}{\partial w_{ij}^t} + \alpha \Delta w_{ij}^t \tag{12}$$

HRBFN – TDD method

The structural learning algorithm discussed above does provide a "dynamic" structure building of the neural network, but it cannot adapt the temporal time series relations of the input and output feedback data sequences into the model. In order to code with this problem, a temporal difference method with decay feedback is hybridized into the learning algorithm of the proposed model. The basic concepts are presented as follows.

In a typical time series prediction problem, given a series of past observations of time-step n at time t, i.e. $[x_1^t, x_1^{t-1}, x_1^{t-2}, x_1^{t-3},\ldots,x_1^{t-n+1}]$, with the predictive time-step of k, we not only obtain the predicted output at time t+k, i.e. p_k^t, but more importantly is the sequence of future events started from time t, i.e. $[p_1^t, p_2^t, p_3^t \ldots, p_k^t]$. In other words, the network can provide an overlapping and inter-related event

sequence as an additional "hint" for network learning, which can be implemented by using Temporal Difference technique [Sutton, 1983]. Besides, consider a sequence of temporal difference operations from time t+1 to t+k, the prediction from a "nearer" future normally has a higher level of confidence than a "far" future, so a decay operator is integrated into the learning algorithm in order to reflect the situation. With the integration of TDD methodology, the learning algorithm discussed in equation (12) will be modified into:

$$\Delta w_{ij}^{t+1} = \eta(\sum_{h=2}^{k}\gamma(h)\cdot(p_{h-1}^{t+1}-p_h^t)\sum_{l=1}^{t}\lambda^{t-1}\frac{\partial p_h^l}{\partial w_{ij}^t} + (p_1^t - p_0^t)\cdot \qquad (13)$$

$$\sum_{l=1}^{t}\left[\gamma(l)\cdot\lambda^{t-1}\frac{\partial p_1^l}{\partial w_{ij}^t}\right] + \alpha\Delta w_{ij}^t - \varepsilon''\mathrm{sgn}(w_{ij}^t)$$

where ε" is defined as:

$$\varepsilon'' = \begin{cases} \varepsilon & |w_{ij}^t| =< \theta \\ 0 & otherwise \end{cases} \qquad (14)$$

3. Implementation

3.1 NORN for stock prediction

Introduction

In section 2 we have discussed the critical components of NORN for automatic time series prediction and pattern recognition. In this section, we illustrate how these technologies can be integrated to stock prediction in the following ways: 1) long-term stock trend prediction; 2) short-term stock price prediction. A schematic diagram for the integrated NORN model for stock prediction is shown in Figure 3.

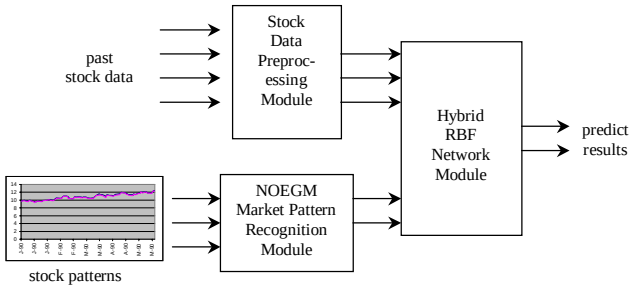


Figure 3 – Schematic diagram of the integrated NORN model for stock prediction

The integrated NORN model consists of the following main modules:

1) Stock data pre-processing module;
2) NOEGM market pattern recognition module;
3) Hybrid RBF network module for stock prediction.

Stock data pre-processing module

From the implementation point of view, ten-year daily stock data (1990-1999) of 33 major Hong Kong stocks are being adopted. Daily information includes daily high, low and closing stock value. The inputs of the HRBF recurrent network is generated by 1) pre-processed time series stock data, and 2) market pattern recognition results.

According to the different requirements and forecasting purpose of NORN for stock prediction, the time series stock data are processed differently for the long-term trend prediction and short-term price forecasting.

A) Data pre-processing (feature extraction) for long-term trend prediction

For long-term trend prediction, since we are interested in stock pattern variation instead of daily price fluctuation, two types of inputs (pre-processed) are being used: 1) normalized daily closing stock value (x_i); and 2) normalized 'desired-value' for days between a buy and a sell signal (d_i) given by:

$$x_j = \frac{c_i - c_{i-j}}{c_i\lambda_j}\ ,\ j=1...A, -1 \le x_j \le 1 \qquad (15)$$

where $\lambda_j = \frac{2}{P-A}\sum_{i=A+1}^{P}\frac{|c_i - c_{i-j}|}{c_i}\ ,\ j=1...A \qquad (16)$

$$d_i = \frac{c_i - c_{low}}{c_{high} - c_{low}} \qquad (17)$$

The main purpose of this data pre-processing (feature extraction) method given by equation (15) is to extract the difference between the current day's closing price (c_i) and the previous A closes (c_{i-j}). This difference is then normalized by dividing it with the closing price and the scaling factor λ.

In order to 'feed-in' the critical signal for trend prediction, the normalized 'desired-value' for the days between a buy and a sell signal (d_i) is being adopted. Actually, this signal is based on the concept of 'oscillator-based system' of the technical financial analysis discussed previously, a kind of 'neural-network oscillator' for stock prediction. For example, when this neural net oscillator rises above a sell threshold, it signals an alert that the market is approaching a trend change to the down side. On the other hand, when the neural net indicator falls below a buy threshold, it signals an alert that the market is approaching a trend changes to the up side. So, these oscillatory input nodes can provide an effective indicator for trend prediction.

B) Data pre-processing (feature extraction) for short-term price prediction

Unlike long-term trend prediction, short-term stock price prediction focuses on the fluctuations of the stock value on daily basis. So, the strategy for creating input patterns is to use the daily change in price (Δ values for open, close, high and low) to predict the next day's high, low and close.

Using similar normalization approach, for each different type of Δ values (open, close, high and low), the volatility factor λ can be calculated as follows:

$$\lambda = \frac{2}{P}\sum_{i=1}^{P}\frac{|\Delta|}{c_i} \qquad (18)$$

and each normalized input value is given by:

$$x_i = \frac{\Delta}{c_i\lambda}\ ,\ i=1...A, -1 \le x_i \le 1 \qquad (19)$$

The desired outputs ($d_{h,i}$, $d_{l,i}$, $d_{c,i}$) for the high, low and close values are also normalized in the similar approach as shown below: (20-22)

$$d_{h,i} = \frac{h_{i+1} - c_{i+1}}{c_i \lambda_h} \quad , d_{l,i} = \frac{c_{i+1} - l_{i+1}}{c_i \lambda_l} \quad , d_{c,i} = \frac{c_{i+1} - c_i}{c_i \lambda_c}$$

Another important fact for this kind of normalization is, once the predicted outputs (o_{h+1}, o_{l+1}, o_{c+1}) are obtained, the predicted stock prices can be transformed easily based on the following equations:

$$c_{i+1} = 2\, c_i\, \lambda_c\, (o_c - 0.5) + c_i \quad (23)$$

$$h_{i+1} = c_{i+1} + c_i\, \lambda_h o_h \quad (24)$$

$$l_{i+1} = c_{i+1} - c_i\, \lambda_l o_l \quad (25)$$

NOEGM market pattern recognition module

By using the neural oscillatory-based elastic graph matching (NOEGM), market patterns from the time series stock pattern for each stock can be extracted and recognition based on the market pattern templates for the 1) major reversal patterns and 2) major continuation patterns being shown in figures 1 and 2 respectively. From the implementation point of view, according to the type of market patterns being extracted from the time series stock price pattern, they are grouped into a set of three different input nodes: 1) reversal Top pattern node; 2) reversal Bottom pattern node; and 3) continuation pattern node.

Hybrid RBF network module for stock prediction

According to the schematic diagram of NORN model, time series stock information of 33 Hong Kong major stocks in a period from 1990 to the end of 1999 are 'fed in' to the hybrid RBF network in the following two ways: 1) pre-processed time series stock data for each stock based on the normalization and transformation techniques discussed previously, with windows size ranging from 10 to 45 days (for long-term trend prediction) and 1 to 10 days (for short-term stock prediction); 2) market patterns being extracted and identified from the time series stock patterns, which is quantified by the correlation values discussed previously.

3.2 Experimental results

From the system validation and performance evaluation point of view, three different types of experimental tests are conducted:

1) Market pattern recognition test for NOEGM validation;
2) Long-term/short-term window size evaluation test;
3) NORN model performance test.

Market pattern recognition test

In the test, four representative stocks from the four main business sectors (i.e. banking, finance investment, public utility and property) were used for system evaluation. They were: HSBC Holdings (banking), Tian On China Investment (finance investment), Hong Kong Telecom (public utility) and Cheung Kong Holdings Ltd. (property). Based on the major market patterns being shown in figures 1 and 2 and the stock pattern for these stocks in the period from 1990 to 1993. The automatic pattern recognition result based on CNOM model is shown in table 1.

Table 1 - Market pattern recognition test results

	Market patterns recognition rate *				
Market Patterns	HSBC Holdgs.	Tian On China Invest.	Hong Kong Telecom	Cheung Kong Holdgs.	Overall
A) Reversal patterns (Top)	182/189 (96%)	160/170 (94%)	169/182 (93%)	170/184 (92%)	93.93%
B) Reversal patterns (Bottom)	154/167 (92%)	159/173 (92%)	163/182 (90%)	184/207 (89%)	90.53%
C) Continuation patterns	318/349 (91%)	296/327 (91%)	296/335 (88%)	266/301 (88%)	89.63%
Overall	**654/*705* *(93%)***	**615/670 (92%)**	**628/699 (90%)**	**620/692 (90%)**	**91.00%**

Note * The pattern recognition rates from different patterns are denoted by: Matched cases / Total cases (recognition rate %)

As illustrated in Table 1, out of a total of 2766 market patterns being identified for these four stock pattern series, the correct matching rate is over 90%, with a total of 2517 correct matched patterns. The overall recognition rate ranging from 89% (for continuation patterns) to about 94% (for reversal Top patterns).

Actually, in view of the detailed recognition rate of the sub-category pattern for each type of market patterns, the recognition rates maintain an acceptable level of around 85 to 96%, except for some particular categories such as the 'Round bottom' pattern in the reversal bottom patterns and the 'Rectangles' pattern in the continuation patterns.

There is one important point needed to bring out and is it the 'basis' of the recognition test - that is, the 'Total cases' figures are identified by human subjective justification which can be explained why some categories have scored a lower pattern, as there exist a certain degree of ambiguity in some template patterns.

For the processing speed of pattern recognition in our test, the average time for pattern segmentation and recognition are 35s and 12s respectively, as compared with the previous work based on Composite Neural Oscillatory Model (CNOM) for cloud pattern recognition [Lee and Liu, 2000], the pattern segmentation and recognition are 50s and 20s corresponding to an improvement of 30 and 40% respectively due to lesser degree of complexity of the proposed model [Lee and Liu, 1999b].

Long-term/short-term window size evaluation test

As the window size of the stock prediction model can provide a deterministic factor for the network efficiency and accuracy, this experimental test aimed at the evaluation of the optimized window size for 1) long-term trend prediction and 2) short-term stock price prediction.

In the test, window sizes of 10 to 45 days (for long-term trend prediction) and 2 to 10 days (for short-term price prediction) were used respectively. The predicted output for these two sets of predictions are as follows:

1) Long-term trend prediction:
   -'Predicted desired-value (d_i)' for days between a buy and a sell signal.
2) Short-term price prediction:
   - 'Predicted next-day High (h_{i+1})
   - 'Predicted next-day Low (l_{i+1})
   - 'Predicted next-day Close (c_{i+1})

Based on the four typical stock items used in the previous test. Table 2 shows the experimental results in terms of average percentage error of the NORN model under different window sizes.

Table 2 - NORN model window size evaluation test results

Long-term trend prediction		**Short-term price prediction**	
Window size (days)	Av. % error	Window size (days)	Av. % error
10	2.345%	2	2.347%
15	1.075%	3	1.473%*
20	1.073%*	4	1.476%
25	1.238%	6	1.487%
30	2.487%	8	2.746%
40	5.347%	10	3.248%

As illustrated in Table 2, the optimal window size for long-term and short-term stock prediction are 20 days and 3 days respectively, with average percentage errors of 1.075% and 0.473% being achieved.

NORN model performance test

In this test, the ten-year (1990-1999) stock information of 33 major Hong Kong stocks were adopted. For comparison purpose, a neural network based forecasting aid 'Neuro™ Forecaster' from Neuro Intelligent Business Software was adopted in the test. Actually, Neuro™ Forecaster provides the following neural network models:

1) Time series feed-forward back-propagation model (FFBP) with different choices of transfer functions, which includes: standard sigmoid function, hyperbolic tangent function, Neurofuzzy function, etc.
2) 'Genetica Net Builder' - based on Genetic Algorithms (GA) for the construction and optimization of the network model.

Table 3 - NORN model performance test results
(For next-day stock price prediction)

Business Type	**Neuro Forecaster (Average % error)**				**NORN (Av. % error)**
	Sigmoid	Hyberbolic tangent	Neuro-Fuzzy	Genetica	
A) Banking	4.647%	3.457%	8.451%	3.427%	1.427%
B) Finance & investment	3.745%	3.758%	7.845%	4.747%	1.573%
C) Public utility	3.412%	4.285%	10.457%	2.747%	1.347%
D) Property	4.452%	5.474%	7.457%	3.417%	1.417%
E) Others	5.124%	4.789%	8.982%	3.746%	1.379%
Overall	**4.217%**	**4.648%**	**8.975%**	**3.417%**	**1.401%**

For the ease of comparison, these 33 stock items are grouped under the four critical business sectors, namely: banking, finance and investment, public utility, property and others. The experimental results for the next-day stock price prediction are shown in Table 3.

4. Conclusion

In this paper, we have presented an innovative, fully automatic and integrated stock prediction model to serve the purposes of 1) long-term stock trend prediction, and 2) short-term stock price forecast. Unlike contemporary neural network models for financial prediction [Vellido *et al.*, 1999] which focus on the 'replacement' of the conventional financial analysis and prediction techniques with various neural net technologies, this paper explores a new era for the implementation of neural networks in financial engineering in which neural networks can be successfully integrated with conventional financial analysis techniques such as technical analysis tools to improve the efficiency and effectiveness of the financial prediction results.

Acknowledgment

The authors are grateful to the partial supports of the Departmental Grant #H-ZJ87, Central Research Grant #PolyU 5091/00E and the iJADE projects of Hong Kong Polytechnic University.

References

[Ishikawa, 1996b] M. Ishikawa. Structural Learning with Forgetting. *Neural Networks*, 9(3) 509-521, 1996.

[Lee *et al.*, 1999] R. Lee, J. Liu and Y. You. Face Recognition: Elastic Relation Encoding and Structural Matching. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics (SMC'99)*, Tokyo, Japan, pages 172-177, 1999.

[Lee and Liu, 1999a] R. S. T. Lee and J. N. K. Liu. An Oscillatory Elastic Graph Matching Model for Scene Analysis. In *Proceedings of International Conference on Imaging Science, Systems, and Technology (CISST'99)*, Las Vegas, USA, pages 42-45, 1999.

[Lee and Liu, 1999b] R. S. T. Lee and J. N. K. Liu. An Automatic Satellite Interpretation of Tropical Cyclone Patterns Using Elastic Graph Dynamic Link Model. *International Journal of Pattern Recognition and Artificial Intelligence*, 13(8) 1251-1270, 1999.

[Lee and Liu, 2000] Raymond. S. T. Lee and James N. K. Liu. Tropical Cyclone Identification and Tracking System Using Integrated Neural Oscillatory Elastic Graph Matching and Hybrid RBF Network Track Mining Technique, *IEEE Transaction on Neural Networks*, 11(3) 680-689, 2000.

[Murphy, 1986] J. J. Murphy. Technical Analysis of the Future Markets. *The New York Institute of Finance*, Prentice Hall, New York, 1986.

[Ritchie, 1996] J. Ritchie. *Fundamental analysis: a back-to-the-basics investment guide to selecting quality stocks*, Chicago, Irwin Professional Pub, 1996.

[Sutton, 1983] R. S. Sutton. Learning to Predict by the Methods of Temporal Difference. *Machine Learning*, No. 3, pages 8-44, 1983.

[Vellido *et al.*, 1996] A. Vellido, P. J. G. Lisboa and J. Vaughan. Neural Networks in Business: A Survey of Applications (1992 - 1998). *Expert Systems with Application*, 17(1) 51-70, 1999.

[Zirilli, 1997] J. S. Zirilli. *Financial Prediction using Neural Networks*, International Thomson Computer Press, 1997.

A General Updating Rule for Discrete Hopfield-Type Neural Network with Delay

Shenshan Qiu[1,a], Eric C.C. Tsang[b], Daniel S. Yeung[b], Xizhao Wang[b]
[b]Dept. of Computing, The Hong Kong Polytechnic University
[a]South China University of Technology, Guangzhou, 510640 China
ausqiu@scut.edu.cn; ssqiu@21cn.com, csetsang, csdaniel, csxzwang@comp.polyu.edu.hk

Abstract

In this paper, the Hopfield neural network with delay (HNND) is studied from the standpoint of regarding it as an optimized computational model. Two general updating rules for network with delay (GURD) are given based on Hopfield-type neural networks with delay for optimization problems and characterized dynamic thresholds. It is proved that in any sequence of updating rule modes, the GURD monotonously converges to a stable state of the network. The diagonal elements of the connection matrix are shown to have an important influence on the convergence process, and they represent the relationship of the local maximum value of the energy function with the stable states of the networks. All ordinary DHNN algorithms are instances of GURD. It can be shown that the convergence conditions of GURD may be relaxed in the context of applications, for instance, the condition of nonnegative diagonal elements of the connection matrix can be removed from the original convergence theorem. New updating rule mode and restrictive conditions can guarantee the network to achieve a local maximum of the energy function.

1 Introduction

Recently the convergence of various connectionist models including discrete Hopfield neural network (DHNN) [Hopfield, 1982; Wilson and Pawley, 1988; Bruck, 1990], continuous Hopfield neural network (CHNN) [Hopfield, 1982] and network with delay [Clouse and Lee, 1997] has been studied extensively. Since Hopfield and Tank [Hopfield, 1982] first applied DHNN to the traveling salesman problem, DHNN (which has a very simple and efficient hardware implementation) has demonstrated the capability to solve a wide variety of combinatorial optimization problems. However, Hopfield and Tank's approach has been subjected to severe criticism on the quality of its solutions, especially for large-scale problems [Wilson and Pawley, 1988]. Other issues concerning DHNN such as computational complexity [Orponen, 1992], computational power, finite automata equivalency and dynamic system simulation by DHNN are beyond the scope of this paper.

It is well known that the DHNN without delay has an important property, i.e., it always converges to a stable state when operating in a serial mode and to a cycle of length at most 2 when operating in a parallel mode [Hopfield, 1982; Wilson and Pawley, 1988; Bruck, 1990; Xu *et al*., 1998]. The problem of convergence is considered one of the most important theoretical issues in the study of the neural networks. However, in most optimization problems, the corresponding matrix and updating rule mode do not satisfy convergence conditions. Therefore its application domain is rather limited. To correct the deficiencies of the Hopfield network, various attempts such as optimal parameter selection and infeasible solution elimination have been made. Among these attempts, some are to maintain the structure of the Hopfield network, while others are to modify the network structure. In [Qiu *et al*., 2000; Qiu *et al*., 1999a; Qiu *et al*., 1999b] a DHNN with delay is constructed and a serial (updating rule) mode similar to one in Hopfield network without delay is also presented. But the corresponding convergence conditions are strictly constrained. It is also shown that the energy function is dependent on the updating rule mode. It is well known that a detailed specification for the convergence of DHNN is very critical to most of the applications and it remains an open problem on how much could the convergence conditions be relaxed.

In fact, the convergence conditions, network architecture, network initial conditions as well as updating rule modes are

[1] Corresponding to: Shenshan Qiu, Department of Automatic Control Engineering, South China University of Technology, Guangzhou, 510640 China;

intimately related. The aim of this paper is to extend the convergence conditions given in [Qiu *et al*., 2000] and to specify the updating rule mode which will make the energy function decrease monotonically with respect to the evolution time. The energy function is modified by means of a modified DHNN as well as its updating rule mode, so that the convergence of DHNN with delay is obtained.

This paper has the following organization. Section 1 is the introduction. Section 2 gives a brief review for DHNN with delay. Section 3 introduces some definitions and notations. Section 4 discusses the convergence of DHNN with delay and presents the main result. The last section concludes the paper.

2 Hopfield Networks with Delay

Discrete Hopfield neural network (DHNN) [Hopfield, 1982; Wilson and Pawley, 1988; Bruck, 1990; Xu *et al*., 1998] is described as follows. The state of the network at time *t* is the vector $X(t)=[x_1(t),x_2(t),...,x_n(t)]^T$. In general, the state of the network at time *t*+1 is a function of $\{W,q\}$ and the state of network at time *t*. This network is thus completely determined by the parameters $\{W,q\}$, the initial state values $X(0)$, and the manner in which the nodes are updated (evolved). If at time step *t*, node x_i is chosen for updating, then at the next time step $t+1$,

$$x_i(t+1)=\operatorname{sgn}(h_i(t))=\begin{cases}1 & ,h_i(t)\ge 0\\ -1 & ,h_i(t)<0\end{cases} \tag{1}$$

where $h_i(t)=\sum_{j=1}^{n} w_{ij}x_j(t)-q_i$.

If only one neuron is allowed to change state at each time step by performing the state evolving equation (1) while the states of the other nodes are unchanged, then the network is said to be operating in a serial mode. If all the nodes are chosen to be updated at each time step by performing the state evolving equation (1), then the network is said to be operating in a parallel mode.

A DHNN with delay is a computational model, similar to a multilayer perceptron in which all connections are feed-forward. The difference between a DHNN with delay and a multilayer perceptron is as follows. The inputs to any node x_i for DHNN with delay may consist of the outputs of earlier nodes not only during the current step *t*, but also during previous *d* time steps $(t-1,t-2,...,t-d)$. Following [Bruck, 1990; Orponen, 1992], the state of a neuron is updated according to the following equation:

$$x_i(t+1)=s_i(H_i(t))\in\{1,-1\} \tag{2}$$

where $H_i(t)=\sum_{k=0}^{d}\sum_{j=1}^{n} w_{ij}^k x_j(t-k)-q_i(t)$.

In order to simplify the analysis presented later in this paper, only the special case *d*=1, $q(t)\equiv q$ will be discussed here. $s_i(\cdot)$ represents the content of the output function of the neuron and the updating rule mode. The quadruple $N=(w^0,w^1,q,s)$ is used to represent the model (2). For instance, if $w^1=O_{n\times n}$, $s_i(\cdot)=\operatorname{sgn}(\cdot)$, then the quadruple $N=(w^0,w^1,q,\operatorname{sgn})$ represents the DHNN model.

3 Updating Rule of DHNN with Delay

Let $B^n\equiv\{-1,1\}^n$ represent the set of all vectors $X=(x_1,x_2,...,x_n)^T$ where each component x_i assumes the value +1 or –1.We propose a new updating rule for DHNN with delay in order to extend the convergence conditions and establish an algorithm that will enable the network to escape from the local maximum points. The following is the definition of a generalized updating rule.

Definition 1. Let $N=(w^0,w^1,q,s)$ represent a DHNN with delay, starting from any initial vector $X(1)=X(0)\in B^n$. Given a sequence of neural index subsets $L(k)\subseteq\{1,2,...,n\}$ for $k\ge 1$, the neural network is updated in the following form: When $k=1$, $L(1)$ is a neural index subset by means of random selection (in general $L(1)\equiv\{1,2,..,n\}$), $x_i(k+1)=x_i(k)$, if $i\notin L(k)$ and for $i\in L(k)$, $k>1$, the following defines updating rule s :

$$x_i(k+1)=\begin{cases}-x_i(k), & if\ x_i(k)H_i(k)>x_i(k)T_i(k)\\ x_i(k), & otherwise\end{cases} \tag{3}$$

where $H_i(k)\equiv\sum_{j=1}^{n}\left(w_{ij}^0x_j(k)+w_{ij}^1x_j(k-1)-q_i\right)$, and

$$T_i(k)\equiv -w_{ii}^0+\sum_{\substack{j\in L(k)\\ j\ne i}}\left|w_{ij}^0\right|+2\sum_{j\in L(k-1)}\left|w_{ij}^1\right|$$

The updating mode is called the general updating rule for the neural network with delay, GURD for short. A sequence of neural index subsets $L(k)\subseteq\{1,2,...,n\}$ satisfying the above conditions is generalized in an evolution mode. It includes a serial mode for neural network with delay when the sequence of neural index subsets is chosen which satisfy $|L(k)|=1$ [Xu *et al*., 1998; Sun, 1998]. The following gives an illustrated example.

Example **1.1**

$$w^0=\begin{pmatrix}-7&-2&-2&4\\-2&-5&-4&6\\-2&-4&11&-3.5\\4&6&-3.5&0\end{pmatrix},\ w^1=\begin{pmatrix}10&-2&2&-4\\1&5&2.5&-1\\-2&-2.5&10&5\\4&1&-5&10\end{pmatrix},\ q=\begin{pmatrix}-12.8294\\-29.4529\\-10.8275\\-4.8314\end{pmatrix}$$

The updating results of example 1.1 are presented in Table 1. It is noted that, for each updating mode, the threshold of a neuron is different, depending on the given

sequence of neural index subsets $L(k)$ which determines the sequence of updating mode. The threshold $T_i(k)$ changes with the updating time k. However, for a Hopfield Neural network, its threshold is a constant. So the stable state of the network is constrained. In other words, when the weights and threshold parameters of a network are given, its fixed points or stable states have been fixed, and they can not change with different updating modes, such as serial mode or parallel mode. On the other hand, the stable state of GURD mode is dynamically distributed based on a given sequence of neural index subsets $L(k)$. Of course, the global maximum point will not change with respect to different updating modes. Only the local maximum points will be affected by these updating modes. Thus this dynamically distributed mechanism and delay relation enable the network to escape from local maximum points and finally reach the global maximum point (similar to LME algorithm [Peng, *et al*., 1996], a strategy to identify which delay item can be regarded as a certain noise). This is what motivates the investigation on the updating mode and the delay neural network model.

lvs code	State vector code; Updated neuron index								SPs	E.Value
	1	2	3	4	1	2	3	4		
0→	8	12	12	13	13	13	13	13	13	Submax
1→	9	13	13	13	13	13	13	13	13	Submax
2→	10	14	14	15	15	15	15	15	15	Max
3→	11	15	15	15	15	15	15	15	15	Max
4→	4	4	4	5	5	5	5	5	5	C
5→	5	5	5	5	5	5	5	5	5	C
6→	6	6	6	6	6	6	6	6	6	B
7→	7	7	7	7	7	7	7	7	7	A
8→	8	12	12	13	13	13	13	13	13	Submax
9→	9	13	13	13	13	13	13	13	13	Submax
10→	10	14	14	15	15	15	15	15	15	Max
11→	11	15	15	15	15	15	15	15	15	Max
12→	12	12	12	13	13	13	13	13	13	Submax
13→	13	13	13	13	13	13	13	13	13*	Submax
14→	14	14	14	15	15	15	15	15	15	Max
15→	15	15	15	15	15	15	15	15	15*	Max

Table 1. Updating procedure from all initial values to SP for example 1.1

Definition 2. The function $E(u,v)$ defined below is called a bivariate energy function on B^n for $N=(w^0,w^1,q,s)$:

$$E(u,v)=u^T w^0 u+2u^T w^1 v-2u^T q, \tag{4}$$

where $u=x(t), v=x(t-1)$. In short, the bivariate energy function is written as $E(t)\equiv E(x(t),x(t-1))$.

4 Main Results

Based on the GURD mode, the neural network with delay has the following convergence properties.

Theorem 1. Let $N=(w^0,w^1,q,s)$ be a DHNN with delay, operating in GURD mode s. Suppose w^0 is a symmetric matrix and w^1 is a diagonal dominant matrix, i.e., $w^1_{ii}\geq\sum_{j\neq i}|w^1_{ji}|$. Then for any $k\geq 1$, $\Delta x(k)\neq 0$ or $\Delta x(k-1)\neq 0$, it is guaranteed that

$$\Delta E(k)\equiv E(x(k+1),x(k))-E(x(k),x(k-1))\geq 0. \tag{5}$$

For any $L(k)\subseteq\{1,2,...,n\}$ the equality holds if only and if $x(k+1)=x(k)=x(k-1)$.

Proof: According to definition 2

$$E(u,v)=u^T w^0 u+2u^T w^1 v-2u^T q$$

where $u=x(k), v=x(k-1)$, q is a threshold vector $q=(q_1,q_2,...,q_n)^T$. Then

$$|E(k)|\equiv|E(x(k),x(k-1)|\leq\sum_{i=1}^{n}\sum_{j=1}^{n}\left(|w^0_{ij}|+2|w^1_{ij}|\right)+2\sum_{i=1}^{n}|q_i|.$$

For any $k\geq 2, i\in L(k)\subseteq\{1,2,...,n\}$ and GURD in the operating mode, let $\Delta x_i(k)\equiv x_i(k+1)-x_i(k), \Delta E(k)\equiv E(k+1)-E(k)$, we have

$$\begin{aligned}\Delta E(k)&\equiv x^T(k+1)w^0x(k+1)+2x^T(k+1)w^1x(k)-2x^T(k+1)q\\&\quad-x^T(k)w^0x(k)-2x^T(k)w^1x(k-1)+2x^T(k)q.\\&=\sum_{i=1}^{n}2\Delta x_i(k)H_i(x(k))+\sum_{i=1}^{n}\sum_{j=1}^{n}(\Delta x_i(k))(\Delta x_j(k))w^0_{ij}\\&\quad+2\sum_{i=1}^{n}\sum_{j=1}^{n}x_i(k+1)\Delta x_j(k-1)w^1_{ij}\\&=\sum_{i=1}^{n}2\Delta x_i(k)\left\{H_i(x(k))+\frac{1}{2}\sum_{j=1}^{n}(\Delta x_j(k))w^0_{ij}+\sum_{j=1}^{n}(\Delta x_j(k-1))w^1_{ij}\right\}\\&\quad+2\sum_{j\in L(k-1)}\Delta x_j(k-1)\left\{\sum_{i=1}^{n}x_i(k)w^1_{ij}\right\}\\&=\sum_{i\in L(k)}2\Delta x_i(k)\left\{H_i(x(k))+\frac{1}{2}\sum_{j\in L(k)}(\Delta x_j(k))w^0_{ij}+\sum_{j\in L(k-1)}(\Delta x_j(k-1))w^1_{ij}\right\}\\&\quad+2\sum_{i\in L(k-1)}\Delta x_i(k-1)\left\{\sum_{j=1}^{n}x_j(k)w^1_{ji}\right\}\end{aligned}$$

(case 1) If $\Delta x_i(k)>0$, then $x_i(k)=-1$ and $x_i(k+1)=1$.

Thus we have

$$H_i(k)>-w^0_{ii}+\sum_{\substack{j\in L(k)\\ j\neq i}}|w^0_{ij}|+2\sum_{j\in L(k-1)}|w^1_{ij}|$$

being a sufficient condition for $(\Delta E(k))_i>0$;

(case 2) If $\Delta x_i(k)<0$, then $x_i(k)=1$ and $x_i(k+1)=-1$.

Thus we have

$$H_i(k)<w^0_{ii}-\sum_{\substack{j\in L(k)\\ j\neq i}}|w^0_{ij}|-2\sum_{j\in L(k-1)}|w^1_{ij}|$$

being a sufficient condition for $(\Delta E(k))_i>0$;

(case 3) as w^1 is a column diagonal dominant matrix, then

$$\Delta E(k) \geq 0 \text{ when } \Delta x_i(k) = 0, \Delta x_i(k-1) \neq 0.$$

Any nonzero update with $\Delta x(k) \neq 0$ or $\Delta x(k-1) \neq 0$ in the GURD is of these three cases. So $\Delta E(k) \geq 0$.

Tables 1 and 2 present the updating results of example 1.1. Especially in Table 2, the experimental results support our main theorem. It shows that our method is more likely to converge to the maximum or the submaximum values than the Hopfield neural network method does (the Hopfield neural network could be applied to the energy function of the example 1.1, after modifying the diagonal elements of the weight matrix). The three stable points (coded SPs: 5,6,7) which are neither max nor submax value points, have the greatest energy function values.

Ivs code	Energy F.Value	Stable Points	Relation SP with (EFValue)
0	-51.8824	13	Submax
1	-58.5569	13	Submax
2	29.4276	15	Max
3	-5.24686	15	Max
4	69.9292	5	C
5	111.255	5	C
6	119.239	6	B
7	132.565	7	A
8	3.43518	13	Submax
9	28.7607	13	Submax
10	68.7452	15	Max
11	66.0707	15	Max
12	101.247	13	Submax
13	174.572	13	Submax
14	134.557	15	Max
15	179.882	15	Max

Table 2. Updating results to all initial values for example 1.1

The following example 1.2 is a revised version of example 1.1. After modifying the original threshold vector (by adding a certain noise or standard Gaussian noise) and the diagonal elements of the weight w^0. The updating results of example 1.2 are given in Table 3. The experimental results of example 1.2 show a much better performance than example 1.1. There are only two stable points. SP 13 is the maximum value point and SP 15 is the submaximum value point.

Example **1.2**

$$w^0 = \begin{pmatrix} 0 & -2 & -2 & 4 \\ -2 & 0 & -4 & 6 \\ -2 & -4 & 0 & -3.5 \\ 4 & 6 & -3.5 & 0 \end{pmatrix} \quad w^1 = \begin{pmatrix} 10 & -2 & 2 & -4 \\ 1 & 5 & 2.5 & -1 \\ -2 & -2.5 & 10 & 5 \\ 4 & 1 & -5 & 10 \end{pmatrix} \quad q = \begin{pmatrix} -33.9882 \\ -40.3587 \\ -8.9825 \\ -4.99416 \end{pmatrix}$$

It is no surprise that the convergence performance of example 1.2 is better than that of example 1.1. This shows that by adjusting the original threshold values of the network parameters, one can change or decrease the local maximum value points. However, this improvement comes as a result of 20 times updating tests and 50 random selection of parameter threshold values. It remains an important open problem on how such threshold values can be adjusted to produce a better performance.

Ivs code	Energy F. Value	Stable Points	E.Value
0	-111.647	13	Max
1	-117.671	13	Max
2	-37.7172	15	Submax
3	-71.7406	15	Submax
4	53.7878	13	Max
5	95.7644	13	Max
6	95.7178	15	Submax
7	109.694	15	Submax
8	28.3057	13	Max
9	54.2822	13	Max
10	86.2356	15	Submax
11	84.2122	15	Submax
12	169.741	13	Max
13	243.717	13*	Max
14	195.671	15	Submax
15	241.647	15*	Submax

Table 3. Updating results to all initial values for example 1.2

All the updating rules are summarized in Table 4. The following illustration is used to explain the entries in Table 4.

Ivs code	State vector code/updated neuron index								SPs	E.Value
	1	2	3	4	1	2	3	4		
0→	8	12	12	13	13	13	13	13	13	Max
1→	9	13	13	13	13	13	13	13	13	Max
2→	10	14	14	15	15	15	15	15	15	Submax
3→	11	15	15	15	15	15	15	15	15	Submax
4→	12	12	12	13	13	13	13	13	13	Max
5→	13	13	13	13	13	13	13	13	13	Max
6→	14	14	14	15	15	15	15	15	15	Submax
7→	15	15	15	15	15	15	15	15	15	Submax
8→	8	12	12	13	13	13	13	13	13	Max
9→	9	13	13	13	13	13	13	13	13	Max
10→	10	14	14	15	15	15	15	15	15	Submax
11→	11	15	15	15	15	15	15	15	15	Submax
12→	12	12	12	13	13	13	13	13	13	Max
13→	13	13	13	13	13	13	13	13	13*	Max
14→	14	14	14	15	15	15	15	15	15	Submax
15→	15	15	15	15	15	15	15	15	15*	Submax

Table 4. Updating procedure from all initial values to SPs for example 1.2

Let the initial value $X(1) = X(0)$ be $(-1,-1,-1,-1)^T \in B^4$. The evolving operational process can be interpreted as follows. First choose a sequence of neural index subsets $L(1) = \{1\}$,

$L(2)=\{2\}$, $L(3)=\{3\}$, $L(4)=\{4\}$, $L(k)=L(i)$, $i=k-[(k-1)/4]\cdot 4$, $k>4$,

where $[s]$ denotes the largest integer which is smaller than s; bivariant $\{st\_code, EFvalue=m_1\}$ represents the neuron state vector code(*st_code*) and the corresponding energy function value(*EFvalue*); and $\xrightarrow{L(i)}$ means updating all elements of the neural index subsets in $L(i)$.

Initial value $\{0, EFvalue=-111.647\} \xrightarrow{L(1),L(2),L(3),L(4)} \{13, EFvalue=243.717\} \xrightarrow{L(4)} \ldots$. It converges to the stable state 13.

Theorem 2. Let $N=(w^0,w^1,q,s)$ be a DHNN with delay, operating in a GURD mode. Suppose w^0 is a symmetric matrix and w^1 is a diagonal dominant matrix, i.e., for all i, $w_{ii}^1 \geq \sum_{j\neq i}|w_{ji}^1|$. Then for any $k\geq 1$, we have

$$\Delta E(k)\equiv E(u,u)-E(u,v)\geq 0 \text{ , for } u,v\in B^n,\quad v\neq u \qquad (6)$$

The inequality holds if only and if $\bigcap_{k=1}^{\infty} L(k)$ is a nonempty set, and there is an index $i\in\bigcap_{k=1}^{\infty}L(k)$ such that $w_{ii}^1>\sum_{j\neq i}|w_{ij}^1|$, where $u=x(k+1), v=x(k)$.

Proof: According to definition 2

$$E(u,v)=u^T w^0 u+2u^T w^1 v-2u^T q,$$

where $u=x(k), v=x(k-1)$, and q is a threshold vector $q=(q_1,q_2,\ldots q_n)^T$. Let $\Delta x_i(k)\equiv x_i(k+1)-x_i(k)$, $\Delta E(k)\equiv E(k+1)-E(k)$. Then

$$\begin{aligned}\Delta E(k)&\equiv 2x^T(k+1)w^1x(k)-2x^T(k+1)q\\&\quad -2x^T(k)w^1x(k-1)+2x^T(k)q\\&=2x^T(k)w^1\Delta x(k-1)=2\sum_{i=1}^{n}\Delta x_i(k-1)\left\{\sum_{j=1}^{n}w_{ji}x_j(k)\right\}.\end{aligned}$$

Without loss of generality, we assume that $L(k-1)\equiv\{1,2,\ldots,n\}$, $\Delta x_i(k-1)\neq 0$, $i\in L(K-1)$. Since w^1 is a column diagonal dominant matrix, it implies that $\Delta E\geq 0$.

If $\bigcap_{k=1}^{\infty}L(k)$ is a nonempty set of $\{1,2,\ldots,n\}$, w^1 is a diagonal dominant matrix and there is an index $i\in\bigcap_{k=1}^{\infty}L(k)$ such that $w_{ii}^1>\sum_{j\neq i}|w_{ji}^1|$, we can conclude

$$\Delta E\equiv E(u,u)-E(u,v)>0.$$

On other hand, if $\Delta E\equiv E(u,u)-E(u,v)\leq 0$, i.e.,

$$\begin{aligned}0\geq\Delta E=u^T w^1(u-v)&=2\left\{2\sum_{i=1}^{n}w_{ii}^1+\sum_{j=1}^{n}\sum_{i\neq j}^{n}u_i w_{ij}^1(u_j-v_j)\right\}\\&\geq 4\sum_{i=1}^{n}(w_{ii}^1-\sum_{j\neq i}^{n}|w_{ji}^1|)\geq 0\end{aligned}$$

Since w^1 is a diagonal dominant matrix, hence $w_{ii}^1=\sum_{j\neq i}|w_{ji}^1|$ is satisfied for all $i\in\{1,2,\ldots n\}$. It is a contradiction as there is an index i such that $w_{ii}^1>\sum_{j\neq i}|w_{ji}^1|$. This completes the proof of theorem 2.

From the general updating rule (5) in theorem 1, it is noted that the network trajectory evolves in such a way that the energy function is always nondecreasing; that is,

$$\Delta E(k)\equiv E(x(k+1),x(k))-E(x(k),x(k-1))\geq 0$$

Using the inequality (6) in theorem 2

$$E(x(k+1),x(k+1))-E(x(k+1),x(k))\geq 0.$$

Hence

$$E(x(k+1),x(k+1))\geq E(x(k+1),x(k))\geq E(x(k),x(k-1)) \qquad (7)$$

The network trajectory now has a new step, i.e., at the $(k+1)th$ step, instead of updating from the $k\ th$ step with initial values $x(k),x(k-1)$, now updating from the $k\ th$ step with initial values $x(k),x(k)$. The modification allows the energy function to accelerate the rate of convergence and simplify the GURD. In the following, we define a new GURD rule.

Definition 3. Let $N=(w^0,w^1,q,s)$ represent DHNN with delay, starting from any initial vector $X(1)=X(0)\in B^n$, given a sequence of neural index subsets $L(k)\subseteq\{1,2,\ldots,n\}$ for $k\geq 1$, the neural network is updated as follows:

1) When $k=1$, $L(1)\subseteq\{1,2,\ldots,n\}$ is a neural index subset by means of random selection (in general $L(1)\equiv\{1,2,\ldots,n\}$),
2) For $L(k)\subseteq\{1,2,..,n\}$, if $i\notin L(k)$, let $x_i(k+1)=x_i(k)$ and for $i\in L(k)$. The updating rule s is given by:

$$x_i(k+1)=\begin{cases}-x_i(k), & if\ x_i(k)\left(w_{ii}^0+2\sum_{\substack{j\in L(k)\\ j\neq i}}|w_{ij}^0|\right)>x_i(k)H_i(k)\\ x_i(k), & otherwise\end{cases} \qquad (8)$$

where $H_i(k)=\sum_{j=1}^{n}\left(w_{ij}^0x_j(k)+w_{ij}^1x_j(k-1)-q_i\right)$.

This updating mode is also called the simplified general updating rule for neural network with delay, SGURD for short.

It is noted that both w^0 being a symmetric matrix and w^1 being a diagonally dominant matrix are required for the

SGURD. Comparing the two constrained conditions (6) and (8), one may easily conclude that the range region of x invariable condition (otherwise case in (8)) decreases. It means that the energy function can reach the local maximum value in a short updating time. In other words, we only need to execute the SGURD rule $2n+1$ steps at most [Qiu *et al*., 1999a]. But for the Hopfield network, the updating rule may require up to n^2 steps in order to obtain a stable state when the energy function has reached to a local maximum [Bruck, 1990].

It is noted that theorem 2 generalizes the results in [Qiu *et al*., 2000; Qiu *et al*., 1999a; Qiu *et al*., 1999b] which extend the constrained conditions of the weight matrices w^0, w^1. The requirement that the diagonal elements in w^0 are non-negative (serial mode) and w^0 is nonnegative definite (parallel mode) are not necessary for the GURD mode. It has been shown that the updating rule process of the network and the activation process of the neurons are characterized by a dynamical process. But the sigmoid function still remains as the most popular choice for the network's activation function, and the serial, parallel and partially parallel modes are considered as important updating rule modes. There are no other modes.

Corollary 2.1 After a finite number of updates, under the GURD mode, the neural network with delay will converge to a fixed point.

In fact the fixed point is a local minimum point of the energy function. On the other hand, the convergence property is indispensable for any practical application design of neural network with delay.

5 Conclusion

In this article we have demonstrated how a generalized updating rule can cause the energy function to increase monotonously for neural network with delay. The GURD mode can operate in any sequence of updating modes. We have shown that the neural network with delay converges when operating in the GURD mode. The algorithm presented in [Qiu *et al.*, 2000; Qiu *et al*., 1999a; Qiu *et al*., 1999b], including the original MHNN algorithm and the partially parallel mode algorithm [Hopfield, 1982; Wilson and Pawley, 1988; Bruck, 1990; Xu *et al*., 1998;Sun, 1998; Lee, 1999], can be easily derived from our main result which is supported by the experimental results of examples 1.1 and 1.2.

Acknowledgements

The authors would like to thank reviewers for their comments and suggestions which helped in improving the manuscript. This research was partially supported by the national nature sciences foundation, China (Grant:69934030) and nature sciences foundation of South China of University Technology.

References

[Hopfield, 1982] J J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. Nat. Acad. Sci*. USA, 1979: 2554-2558, 1982.

[Wilson and Pawley, 1988] G. N. Wilson, and G. S. Pawley. On the Stability of the Traveling Salesman Problem Algorithm of Hopfield and Tank. *Biological Cybernetics*, 58:63-70, 1988.

[Bruck, 1990] Bruck J, On the convergence properties of the Hopfield model. *Proceedings of IEEE,* 78(10):1579-1585, 1990.

[Clouse and Lee, 1997] Daniel S. Clouse and Giles C.Lee. Time-delay neural network: Representation and induction of finite-state machines. *IEEE Transactions on Neural Networks*, 8(5):1065-1070, 1997.

[Orponen, 1992] P. Orponen. Neural networks and complexity theory. *Proc.17th International Symposium on Mathematical Foundations of Computer Science*, pages 50-61,Lecture Notes in Computer Science 629, Springer-Verlag, Berlin ,Germany, 1992.

[Xu *et al*., 1998] Zongbin Xu, Guoqing Hu and Chungping Kwong**.** Asymmetric Hopfield-type networks: theory and applications. *Neural Networks*, 9(3):483-501,1996.

[Sun, 1998] Yi Sun. A generalized updating rule for modified Hopfield neural network for quadratic optimization. *Neurocompting*, 19:133-143, 1998.

[Qiu *et al*., 2000] Shenshan Qiu, Eric C.C. Tsang and Daniel S. Yeung. Stability of discrete Hopfield neural networks with time-delay, In *Proceedings of the 2000 IEEE International Conference on system, Man Cybernetics*, pages 2545-2550, Nashville, Tennessee, October, 2000.

[Qiu *et al*., 1999a] Shenshan Qiu, Xiaofei Xu, Mingzhu Liu and Yadong Wang. Convergence of discrete Hopfield-type neural network with time-delay in a serial mode. *Journal of Computer Research & Development* 36 (5):557-563, 1999.

[Qiu *et al*., 1999b] Shenshan Qiu, Xiaofei Xu, Chunsheng Li and Mingzhu Liu. Matrix criterion of Dynamic analysis in discrete neural networks with delay. *Journal of Software,*10(10):1108-1113, 1999.

[Lee, 1999] Donq-liang Lee. New stability conditions for Hopfield networks in partial simultaneous update mode. *IEEE Transactions on Neural Networks*, 10(4):975-978, 1999.

[Peng, *et al*., 1996] Mengkang Peng, Narendra K. and Allistair F. Armitage. An investigation into the improvement of local minima of the Hopfield network. *Neural Networks,* 9(7):1241-1253, 1996.

NEURAL NETWORKS AND GENETIC ALGORITHMS

NEURAL NETWORKS AND
GENETIC ALGORITHMS

Genetic Algorithm based Selective Neural Network Ensemble

Zhi-Hua Zhou Jian-Xin Wu Yuan Jiang Shi-Fu Chen

National Laboratory for Novel Software Technology
Nanjing University, Nanjing 210093, P.R.China
zhou@nju.edu.cn {wujx, jy}@ai.nju.edu.cn chensf@nju.edu.cn

Abstract

Neural network ensemble is a learning paradigm where several neural networks are jointly used to solve a problem. In this paper, the relationship between the generalization ability of the neural network ensemble and the correlation of the individual neural networks is analyzed, which reveals that ensembling a selective subset of individual networks is superior to ensembling all the individual networks in some cases. Therefore an approach named GASEN is proposed, which trains several individual neural networks and then employs genetic algorithm to select an optimum subset of individual networks to constitute an ensemble. Experimental results show that, comparing with a popular ensemble approach, i.e. *averaging all*, and a theoretically optimum selective ensemble approach, i.e. *enumerating*, GASEN has preferable performance in generating ensembles with strong generalization ability in relatively small computational cost.

1 Introduction

Since neural computing has no rigorous theoretical framework until now, whether a neural network based application will be successful or not is almost fully determined by that who is the practitioner. In general, the more experiences the practitioner has on neural computing, the more chances the application will have in gaining success. However, in real-world applications, the users are often those with little knowledge on neural computing. Therefore the rewards that neural network techniques may return do not always appear.

In the beginning of the 1990's, Hansen and Salamon [1990] showed that the generalization ability of a neural network system can be significantly improved through ensembling neural networks, i.e. training several neural networks and combining their results in some way. Later, Sollich and Krogh [1996] defined neural network ensemble as a collection of a (finite) number of neural networks that are trained for the same task. Since it behaves remarkably well and is easy to use, neural network ensemble is regarded as a promising methodology that can profit not only experts in neural computing but also ordinary engineers in real-world applications. And neural network ensemble has already been used in many real domains such as handwritten digit recognition [Hansen *et al.*, 1992], scientific image analysis [Cherkauer, 1996], face recognition [Gutta and Wechsler, 1996; Huang *et al.*, 2000], OCR [Mao, 1998], seismic signals classification [Shimshoni and Intrator, 1998], etc. Many works have been done in investigating why and how neural network ensemble works. The classical one is Krogh and Vedelsby [1995] 's work, in which they derived a famous equation $E = \overline{E} - \overline{A}$ that clearly demonstrates that the generalization ability of the ensemble is determined by the average generalization ability and the average ambiguity of the individual neural networks that constitutes the ensemble.

In this paper, the relationship between the generalization ability of the neural network ensemble and the correlation of the individual neural networks is analyzed, which reveals that in some cases ensembling an appropriate subset of individual networks is superior to prevailing ensemble schemes, i.e. ensembling all the individual networks at hand. Based upon the recognition that the appropriate subset of individual networks is difficult to be found out directly, a genetic algorithm based approach named GASEN (Genetic Algorithm based Selective ENsemble) is proposed, which trains several individual neural networks and then employs genetic algorithm to select an optimum set of individual networks to constitute an ensemble. Experiments show that

GASEN is superior to a popular ensemble approach, i.e. *averaging all* that averages the outputs of all the individual networks at each output unit. Experiments also show that GASEN is superior to a selective ensemble approach that is theoretically optimum, i.e. *enumerating* that estimates the generalization ability of every possible subset of individual networks and then selects the best subset to make an ensemble. Moreover, most ensemble approaches require their individual networks be independently trained. But at present there is no method can guarantee the independence when there are many individual networks. Since GASEN can increase the ambiguity of the ensemble through selecting the appropriate subset of individual networks, it does not claim independent training, which makes it more easily to use than many other ensemble approaches.

The rest of this paper is organized as follows. In Section 2, the relationship between the generalization ability of the ensemble and the correlation of the individual neural networks is analyzed. In Section 3, GASEN is proposed to find out the optimum subset of individual networks. In Section 4, experiments on *averaging all*, *enumerating*, and GASEN are reported. In Section 5, related works are overviewed. Finally in Section 6, conclusions are drawn and several issues for future works are indicated.

2 Generalization and Correlation

Suppose the learning task is to use an ensemble that comprises N individual neural networks to approximate a function f: $\mathbf{R}^m \rightarrow \mathbf{R}^n$. The predictions of the individual networks are combined through *weighted averaging*, where a weight w_i (i = 1, 2, …, N) is assigned to the individual network f_i, and w_i satisfies equation (1) and (2):

$$0 < w_i < 1 \tag{1}$$

$$\sum_{i=1}^{N} w_i = 1 \tag{2}$$

Therefore the k-th output component of the ensemble is computed according to equation (3), where $f_{i,k}$ is the value of the k-th output component of the i-th individual network.

$$\overline{f}_k = \sum_{i=1}^{N} w_i f_{i,k} \tag{3}$$

For convenience of discussion, here we assume that each individual network has only one output component, i.e. the function to be approximated is f: $\mathbf{R}^m \rightarrow \mathbf{R}$. But note that following derivation can be easily generalized to situations where each individual network has multiple output components.

Suppose $x \in \mathbf{R}^m$ is randomly sampled according to a distribution $p(x)$. The expected output of x is $d(x)$. The actual output of the i-th individual neural network is $f_i(x)$. Then the output of the neural network ensemble is:

$$\overline{f}(x) = \sum_{i=1}^{N} w_i f_i(x) \tag{4}$$

The generalization error $E_i(x)$ of the i-th individual network on input x and the generalization error $E(x)$ of the ensemble on input x are respectively:

$$E_i(x) = (f_i(x) - d(x))^2 \tag{5}$$

$$E(x) = (\overline{f}(x) - d(x))^2 \tag{6}$$

Then the generalization error E_i of the i-th individual neural network on the distribution $p(x)$ and the generalization error E of the ensemble on the distribution $p(x)$ are respectively:

$$E_i = \int dx p(x) E_i(x) \tag{7}$$

$$E = \int dx p(x) E(x) \tag{8}$$

The average generalization error of the individual neural networks on input x is:

$$\overline{E}(x) = \sum_{i=1}^{N} w_i E_i(x) \tag{9}$$

Then the average generalization error of the individual neural networks on the distribution $p(x)$ is:

$$\overline{E} = \int dx p(x) \overline{E}(x) \tag{10}$$

Now we define the correlation between the i-th and the j-th individual neural networks as:

$$C_{ij} = \int dx p(x) (f_i(x) - d(x)) (f_j(x) - d(x)) \tag{11}$$

Note that C_{ij} satisfies equation (12) and (13):

$$C_{ii} = E_i \tag{12}$$

$$C_{ij} = C_{ji} \tag{13}$$

Considering equation (4) and (6) we get:

$$E(x) = \left(\sum_{i=1}^{N} w_i f_i(x) - d(x) \right) \left(\sum_{j=1}^{N} w_j f_j(x) - d(x) \right) \tag{14}$$

Then considering equation (14) and (11) we get:

$$E = \sum_{i=1}^{N} \sum_{j=1}^{N} w_i w_j C_{ij} \tag{15}$$

Different to Krogh and Vedelsby [1995] 's result $E = \overline{E} - \overline{A}$, equation (15) utilizes the correlation between the individual neural networks to represent the generalization error of the ensemble. Since the computation of C_{ij} only refers to f_i and f_j, equation (15) is easier to use

than $E = \overline{E} - \overline{A}$ in real-world applications.

Suppose that $w_i = 1/N$ ($i = 1, 2, \ldots, N$), i.e. the predictions of the individual neural networks are combined via *averaging*. Then equation (15) becomes:

$$E = \sum_{i=1}^{N}\sum_{j=1}^{N} C_{ij} / N^2 \tag{16}$$

Let's assume that the k-th individual neural network is deleted from the ensemble. Then the generalization error of the new ensemble is:

$$E' = \sum_{\substack{i=1 \\ i\neq k}}^{N}\sum_{\substack{j=1 \\ j\neq k}}^{N} C_{ij} / (N-1)^2 \tag{17}$$

Considering equation (16) and (17) we get:

$$E - E' = \frac{2\sum_{\substack{i=1 \\ i\neq k}}^{N} C_{ik} + E_k - (2N-1)E}{(N-1)^2} \tag{18}$$

It is obvious that $E > E'$ when equation (19) is satisfied, which means that the new ensemble that omits f_k is more accurate than the original one that includes f_k.

$$E < \left(2\sum_{\substack{i=1 \\ i\neq k}}^{N} C_{ik} + E_k \right) / (2N-1) \tag{19}$$

Considering equation (19) and (17) we get the constraints on f_k:

$$(2N-1)\sum_{\substack{i=1 \\ i\neq k}}^{N}\sum_{\substack{j=1 \\ j\neq k}}^{N} C_{ij} < 2(N-1)^2 \sum_{\substack{i=1 \\ i\neq k}}^{N} C_{ik} + (N-1)^2 E_k \tag{20}$$

Now a conclusion is arrived that after the individual neural networks are trained, in some cases ensembling an appropriate subset of individual neural networks is superior to ensembling all the individual networks. The individual networks that should be omitted satisfy equation (20).

3 GASEN

Note that the individual neural networks to be omitted are hard to be found out directly by equation (20) due to the extensive computation required. Moreover, following observation is noteworthy that in real-world applications the generalization error of the individual neural networks and that of the ensemble are all unknown. However, we can employ cross-validation to get their generalization error on a validation set, which can be used to approximate the actual generalization error.

An approach named *enumerating* can be utilized to find out the appropriate subset of individual networks, which estimates the generalization error of all the possible subsets of $\{f_1, f_2, \ldots, f_N\}$ and then selects the best subset to make an ensemble. When N is a small number, *enumerating* can achieves optimum results. However, if N is a big number, such as $N > 30$, *enumerating* is nearly impossible to be realized due to its excessive computational cost (it will estimates the generalization error of $2^N - 1$ number of ensembles).

Here we present a practical routine to find out the appropriate subset of individual neural networks. Assume we can assign to each individual neural network an optimum weight that exhibits its importance in the ensemble. Then we can select the individual networks whose weight is bigger than a pre-set threshold λ to constitute the ensemble via *averaging*. Suppose the weight corresponding to the i-th individual neural network is w_i, which satisfies both equation (1) and (2). Then we have a weight vector $w = (w_1, w_2, \ldots, w_N)$. Since the optimum weight should minimize the generalization error of the ensemble, considering equation (15), the optimum weight vector w_{opt} can be expressed as:

$$w_{opt} = \arg\min\left(\sum_{i=1}^{N}\sum_{j=1}^{N} w_i w_j C_{ij} \right) \tag{21}$$

$w_{opt.k}$, i.e. the k-th ($k = 1, 2, \ldots, N$) component of w_{opt} , can be solved by *lagrange* multiplier. $w_{opt.k}$ satisfies:

$$\frac{\partial\left(\sum_{i=1}^{N}\sum_{j=1}^{N} w_i w_j C_{ij} - 2\lambda\left(\sum_{i=1}^{N} w_i - 1 \right) \right)}{\partial w_{opt.k}} = 0 \tag{22}$$

Equation (22) can be simplified as:

$$\sum_{\substack{j=1 \\ j\neq k}}^{N} w_{opt.k} C_{kj} = \lambda \tag{23}$$

Considering that $w_{opt.k}$ satisfies equation (2), we get:

$$w_{opt.k} = \frac{\sum_{j=1}^{N} C_{kj}^{-1}}{\sum_{i=1}^{N}\sum_{j=1}^{N} C_{ij}^{-1}} \tag{24}$$

Although equation (24) is enough to solve w_{opt} in theory, it rarely works well in real-world applications. This is because in the ensemble of real-world applications there are often some individual neural networks that are quite similar in performance, which makes the correlation matrix $(C_{ij})_{N\times N}$ of the ensemble be an inreversible or ill-conditioned matrix so that equation (24) cannot be solved.

Since equation (21) can be viewed as an optimization problem, considering the success that has been obtained by genetic algorithms in optimization area [Goldberg, 1989], GASEN is proposed to find out the appropriate subset of the individual networks. After the individual networks being

trained, GASEN employs genetic algorithm to evolve the optimum weight vector w_{opt}. Then GASEN selects the individual networks whose corresponding optimum weight component is bigger than the pre-set threshold λ to constitute the ensemble. Note that if no individual network is washed out, i.e. every component of the evolved optimum weight vector is bigger than λ, all the individual networks are used to constitute the ensemble. We believe that this is corresponding to the situation that no individual networks satisfying equation (20). Following observation is noteworthy that the output of the ensemble is generated via *averaging*. In other words, the evolved optimum weight vector is only used in the selection of the individual networks. This is because we believe that using the weight vector both in the selection of the individual networks and the combination of the individual predictions is easy to cause overfitting.

Here GASEN is realized by utilizing the standard genetic algorithm [Goldberg, 1989] and a floating coding scheme that represents every component of the weight vector in 64 bits. Therefore each individual in the evolving population is coded in $8N$ bytes where N is the number of the individual networks. Note that since GASEN can be viewed as an abstract approach rather than a concrete algorithm, it can be easily realized by employing diversified kind of genetic algorithms and coding schemes.

Let V denotes the validation set. The estimated value of the correlation between the i-th and the j-th individual neural networks is:

$$C_{ij}^{V} = \frac{\sum_{x \in V} \left(f_i(x) - d(x)\right)\left(f_j(x) - d(x)\right)}{|V|} \qquad (25)$$

Considering equation (15), the estimated generalization error of the neural network ensemble corresponding to the individual w in the evolving population is:

$$E_w^V = \sum_{i=1}^{N}\sum_{j=1}^{N} w_i w_j C_{ij}^V = w^T C^V w \qquad (26)$$

It is obvious that $E_W{}^V$ expresses the goodness of w. The smaller $E_W{}^V$ is, the better w is. So we use $f(w) = 1/ E_W{}^V$ as the fitness function. Note that w may violate equation (2) during its evolving. Therefore it is necessary to do normalization on the evolved optimum w so that its components can be compared with λ. Here we use a simple normalization scheme:

$$w_{opt.i} = w_i / \sum_{i=1}^{N} w_i \qquad (27)$$

4 Experiments

We use four regression problems to compare the performance of three ensemble approaches, i.e. *averaging all*, *enumerating*, and GASEN.

The first problem is *Friedman#1* proposed by Friedman [1991]. There are 5 continuous attributes. The data set is generated according to equation (28) where the noise item ε satisfies normal distribution $N(0, 1)$ and x_i (i =1, 2, …, 5) satisfies uniform distribution $U[0, 1]$. In our experiments the size of the training set and the test set are respectively 200 and 1000.

$$t = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \varepsilon \qquad (28)$$

The second problem is *Boston Housing* from UCI machine learning repository [Blake *et al.*, 1998]. There are 11 continuous attributes and 1 categorical attribute. The data set comprises 506 examples among which 400 examples make up the training set and the rest 106 examples make up the test set in our experiments.

The third problem is *Ozone* proposed by Breiman and Friedman [1985]. There are 9 continuous attributes. The data set comprises 366 examples. Since the intention of the experiments is not to compare the ability of dealing with missing values, 1 attribute and 36 examples that has missing values are omitted as Briedman [1996] did on the data set. Therefore in our experiments there are 8 continuous attributes and 330 examples among which 250 examples make up the training set and the rest 80 examples make up the test set.

The fourth problem is *Servo* from UCI machine learning repository. There are 4 categorical attributes. The data set comprises 167 examples among which 130 examples make up the training set and the rest 37 examples make up the test set in our experiments. Note that some researchers [Quinlan, 1993] believe that this problem is very difficult because it involves some kind of extreme nonlinearity.

For each problem we use *Bagging* on the training set to generate 20 single-hidden-layered BP [Rumelhart *et al.*, 1986] networks. During the training process, the generalization error of each network is estimated in each epoch on a validation set generated via bootstrap sampling from the training set. If the error does not change in consecutive 5 epochs, the training of the network is terminated in order to avoid overfitting. Then we use *averaging all*, *enumerating*, and GASEN to ensemble those trained BP networks. In our experiments the genetic algorithm employed by GASEN is implemented by the GAOT toolbox developed by Houck *et al.* [1995]. The genetic operators, including select, crossover, and mutation, and the system parameters, including crossover probability, mutation probability, and stopping criterion, are all set to the default values of GAOT. The pre-set threshold λ used by GASEN is set to 0.05. The validation set V used by GASEN is also generated via bootstrap sampling from the training set. For every problem we perform 20 runs and record the average mean squared error and the standard deviation on

Table 1 Experimental results on *averaging all*, *enumerating*, and GASEN

Data set	*averaging all*		*enumerating*		GASEN	
	error	deviation	error	deviation	error	deviation
Friedman#1	1.33	0.26	0.47	0.12	0.5	0.14
Boston Housing	12.26	0.97	10.6	0.55	10.68	0.8
Ozone	22.29	2.00	19.85	1.72	19.99	1.63
Servo	0.21	0.026	0.21	0.045	0.226	0.058

the test set of the ensembles. The experimental results are tabulated in Table 1.

Pairwise one-tailed *t*-tests indicate that the generalization error of GASEN is significantly less than that of *averaging all* on *Friedman#1*, *Boston Housing*, and *Ozone*, while there is no significant difference between the performance of those two approaches on *Servo*. We believe that GASEN is superior to *averaging all* because ensembles generated by it are more accurate than that generated by *averaging all* in most cases. Pairwise one-tailed *t*-tests also indicate that there is no significant difference between the performance of GASEN and *enumerating* on all those data sets. Considering that *enumerating* can hardly work when there are lots of individual networks due to its extensive computational cost, we believe that GASEN is superior to *enumerating* because it can generate ensembles with comparable generalization ability in the cost of much smaller computation.

Following observation is interesting. Through analyzing the ensembles, we find that GASEN and *enumerating* averagely select a subset comprises 5 networks out of 20 individual networks to constitute the ensemble. And the subset selected by GASEN is the same as that selected by *enumerating* in 6 runs out of 20 runs on *Friedman#1*, 2 runs out of 20 runs on *Boston Housing*, 3 runs out of 20 runs on *Ozone*, and 7 runs out of 20 runs on *Servo*. It is obvious that the frequency of the appearance of same selected results is much higher than that should exhibit in random selection. Considering that *enumerating* is an optimum approach when the size of ensemble is small, we believe that this observation verifies the goodness of GASEN from another aspect. However, the reason for explaining the high frequency of the appearance of same selected results should be further explored.

5 Related Works

Neural network ensemble has become a very active area and there are a large number of research groups working on it. Besides the achievements cited in the brief review presented in Section 1, some significant developments in this area can be found in [Sharkey, 1999]. Moreover, there are some works very related to this paper.

Yao and Liu [1998] employed genetic algorithm to evolve a population of neural networks. Instead of choosing the best neural network in the last generalization as the final result, they regarded the entire population as a neural network ensemble and combining all the individuals in the last generalization in order to make best use of all the information contained in the population. Although genetic algorithm is used in both their and our works, there are many differences among which a conspicuous one is that they intended to utilize the information contained in the genetic population rather than performing selection on the individual networks.

Liu and Yao [1999] proposed an ensemble learning approach, i.e. negative correlation learning, where all the individual networks are trained simultaneously through the correlation penalty terms in their error functions. Rather than generating unbiased networks whose errors are uncorrelated, negative correlation learning can generate negatively correlated networks to encourage specialization and cooperation among the individual networks. The main reason that negative correlation learning can improve the generalization ability of an ensemble is that it increases the ambiguity item $\overline{A}$ in the famous equation $E = \overline{E} - \overline{A}$. In GASEN, when individual neural networks are selected according to the evolved optimum weight vector, the ambiguity of the ensemble is also increased. This can be derived from the observation that the generalization ability of the ensemble generated by GASEN is better than the ensemble that comprises same number of individual networks that rank toppest in generalization ability.

6 Conclusion

In this paper, the relationship between the generalization ability of the neural network ensemble and the correlation of individual networks is analyzed, which reveals that in some cases ensembling a selective subset of individual networks is superior to ensembling all the individual networks. Then a genetic algorithm based ensemble approach named GASEN is proposed. Experimental results show that GASEN is a promising ensemble approach that is superior to both *averaging all* and *enumerating*.

There are many works left to do in the near future. Firstly, at present GASEN has only been compared with *averaging all* and *enumerating* on several data sets. We plan to do comparisons with more ensemble approaches on more data sets. Secondly, the pre-set threshold λ is an important

parameter of GASEN, which determines the individual neural networks that constitute the ensemble. We hope to find out the relationship between λ and the generalization ability of the ensemble so that we can set λ to appropriate values in real-world applications. Thirdly, we want to explore why there is such a high frequency that GASEN and *enumerating* select the same subset of individual networks to make an ensemble.

Acknowledgements

The comments and suggestions from the anonymous reviewers greatly improve this paper. The National Natural Science Foundation of P.R.China and the Natural Science Foundation of Jiangsu Province, P.R.China, supported this research.

References

[Blake *et al.*, 1998] C. Blake, E. Keogh, and C. J. Merz. UCI Repository of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.html], Department of Information and Computer Science, University of California, Irvine, California, 1998.

[Breiman, 1996] L. Breiman. Bagging predictors. *Machine Learning*, 24(2): 123-140, 1996.

[Breiman and Friedman, 1985] L. Breiman and J. Friedman. Estimating optimal transformations in multiple regression and correlation (with discussion). *Journal of the American Statistical Association*, 80(391): 580-619, 1985.

[Cherkauer, 1996] K. J. Cherkauer. Human expert level performance on a scientific image analysis task by a system using combined artificial neural networks. In *Proceedings of the 13th AAAI Workshop on Integrating Multiple Learned Models for Improving and Scaling Machine Learning Algorithms*, pages 15-21, 1996. AAAI.

[Friedman, 1991] J. Friedman. Multivariate adaptive regression splines (with discussion). *Annals of Statistics*, 19(1): 1-141, 1991.

[Goldberg, 1989] D. E. Goldberg. *Genetic Algorithm in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, 1989.

[Gutta and Wechsler, 1996] S. Gutta and H. Wechsler. Face recognition using hybrid classifier systems. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1017-1022, 1996. IEEE Computer Society.

[Hansen and Salamon, 1990] L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 12(10): 993-1001, 1990.

[Hansen *et al.*, 1992] L. K. Hansen, L. Liisberg L, and P. Salamon. Ensemble methods for handwritten digit recognition. In *Proceedings of the IEEE-SP Workshop on Neural Networks for Signal Processing*, pages 333-342, 1992. IEEE Computer Society.

[Houck *et al.*, 1995] C. R. Houck, J. A. Joines, and M. G. Kay. A genetic algorithm for function optimization: a Matlab implementation, NCSU-IE Technical Report 95-09, North Carolina State University, 1995.

[Huang *et al.*, 2000] F. J. Huang, Z.-H. Zhou, H.-J. Zhang, and T. Chen. Pose invariant face recognition. In *Proceedings of the 4th IEEE International Conference on Automatic Face and Gesture Recognition*, pages 245-250, Grenoble, France, 2000. IEEE Computer Society.

[Krogh and Vedelsby, 1995] A. Krogh A and J. Vedelsby. Neural network ensembles, cross validation, and active learning. In *Advances in Neural Information Processing Systems 7*, pages 231-238, 1995. MIT.

[Liu and Yao, 1999] Y. Liu and X. Yao. Ensemble learning via negative correlation. *Neural Networks*, 12(10): 1399-1404, 1999.

[Mao, 1998] J. Mao. A case study on bagging, boosting and basic ensembles of neural networks for OCR. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, vol.3, pages 1828-1833, 1998. IEEE Computer Society.

[Quinlan, 1993] J. R. Quinlan. *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, California, 1993.

[Rumelhart *et al.*, 1986] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by backpropagating errors. *Nature*, 323(9): 318-362, 1986.

[Sharkey, 1999] D. Sharkey, editor. *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*, Springer-Verlag, London, 1999.

[Shimshoni and Intrator, 1998] Y. Shimshoni and N. Intrator. Classification of seismic signals by integrating ensembles of neural networks. *IEEE Trans. Signal Processing*, 46(5): 1194-1201, 1998.

[Sollich and Krogh, 1996] P. Sollich and A. Krogh. Learning with ensembles: How over-fitting can be useful. In *Advances in Neural Information Processing Systems 8*, pages 190-196, 1996. MIT.

[Yao and Liu, 1998] X. Yao and Y. Liu. Making use of population information in evolutionary artificial neural networks. *IEEE Trans. Systems, Man and Cybernetics – Part B: Cybernetics*, 28(3): 417-425, 1998.

Neural Logic Network Learning using Genetic Programming

Chew Lim Tan, Henry Wai Kit Chia
School of Computing
National University of Singapore
3 Science Drive 2, Singapore 117543
{tancl, chiawaik}@comp.nus.edu.sg

Abstract

Neural Logic Network or *Neulonet* is a hybrid of neural network expert systems. Its strength lies in its ability to learn and to represent human logic in decision making using component *net rules*. The technique originally employed in *neulonet* learning is backpropagation. However, the resulting weight adjustments will lead to a loss in the logic of the *net rules*. A new technique is now developed that allows the *neulonet* to learn by composing *net rules* using genetic programming. This paper presents experimental results to demonstrate this new and exciting capability in capturing human decision logic from examples. Comparisons will also be made between the use of *net rules*, and the use of standard boolean logic of negation, disjunction and conjunction in evolutionary computation.

1 Introduction

There has been a fair amount of interest in training neural networks using genetic programming [Golubski and Feuring, 1999]. Gaudet [1996] applied genetic programming on logic-based neural networks which basically represent standard logic (negation, conjunction and disjunction). Similarly, an adaptive logic network [Armstrong and Thomas, 1996] uses linear threshold units at the leaf node level, while the parent nodes of the network are composed of "AND" and "OR" logic units. However, neural networks can also represent non-standard logic. A Neural Logic Network or simply *Neulonet* has been proposed that emulates human decision logic which are often too complex to be expressed neatly using standard logic [Teh, 1995; Tan *et al.*, 1996].

In this paper, we begin by focusing on the integration of *neulonets* into a genetically programmed environment. First, we introduce the concept of *neulonets* and show how they can be combined to form complex decisions. We then describe a GP system for evolving *neulonets*, and discuss how genetic operations of crossover and mutation can be adapted to *neulonet* evolution. Next, we demonstrate how the system can effectively solve a classification problem and extract rules from the evolved *neulonet*. Finally we provide experimental results to substantiate the use of *neulonets* as an improvement over the use of standard logic networks in genetic programming.

2 Neural Logic Network

A *Neulonet* has an ordered pair of numbers associated with each node and connection (figure 1). Let Q be the output node and $P_1, P_2, \ldots P_N$ be input nodes. Let values associated with the node P_i be denoted by $(a_i,\ b_i)$, and the weight for the connection from P_i to Q be $(\alpha_i,\ \beta_i)$. The ordered pair for each node takes one of three values, namely, (1,0) for true, (0,1) for false and (0,0) for "don't know". (1,1) is undefined. The following activation function determines the output at Q:

$$Act(Q) = \begin{cases} (1,0) & \text{if } \sum_{i=1}^{N}(a_i\alpha_i - b_i\beta_i) \geq \lambda \\ (0,1) & \text{if } \sum_{i=1}^{N}(a_i\alpha_i - b_i\beta_i) \leq -\lambda \\ (0,0) & \text{otherwise.} \end{cases} \quad (1)$$

where λ is the threshold, usually set to 1.

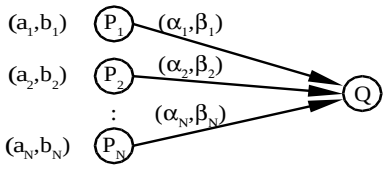


Figure 1: A basic Neural Logic Network - *Neulonet*.

By applying equation (1) on a network in figure 2, the network behaves like an expert system rule with an OR logic operation as shown in the truth table.

P	Q	P or Q
(1,0)	(1,0)	(1,0)
(1,0)	(0,1)	(1,0)
(1,0)	(0,0)	(1,0)
(0,1)	(1,0)	(1,0)
(0,1)	(0,1)	(0,1)
(0,1)	(0,0)	(0,0)
(0,0)	(1,0)	(1,0)
(0,0)	(0,1)	(0,0)
(0,0)	(0,0)	(0,0)

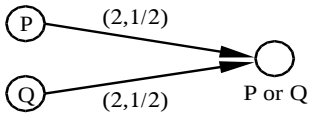


Figure 2: The above *neulonet* behaves like a disjunction rule.

We call a network in figure 2 a *net rule* – a rudimentary network that can be chained with other similar networks just like in a conventional rule-based expert system. The power of a *net rule*, however, is not limited to standard boolean logic operations. A wide range of different human logic in decision making can be represented with *net rules* using different sets of weights. Figure 3 shows some examples of *net rules* that emulate a human decision maker's behavior. Human decision processes are often too complex to be expressed neatly using standard logic. One such complexity is that human reasoning can be biased by giving different degrees of importance to different factors. A simple AND or OR operation will be inadequate in representing varying degrees of bias.

For instance, rule (4) "Priority" in figure 3 exhibits different degrees of influence on the outcome of Q with P_1 being the most important factor, while P_n will only be considered when all other factors are unknown, i.e. (0,0). Another example can be seen in rule (13) "Silence means consent" where *Def* represents the default action. If no one among P_1, P_2, ..., P_n has any opinion, or if there is an equal number of persons either for or against a motion, the default action is taken. In all other cases, the majority wins. *Net rules* can combine to form composite *net rules* to achieve more complicated decisions. For instance, using the output of rule (7) "Unanimity" as the input V to rule (10) "Veto", both with *n* set to 2, an XOR operation can be realized as shown in figure 4.

A *neulonet* combines the strengths of neural networks and expert systems [Tan *et al.*, 1996]. It was proposed that a knowledge engineer could first encode his knowledge into component *net rules* and later use real examples to do refinement training to adjust the weights in the *neulonet*. Training in this case is by means of backpropagation [Teh, 1995], which will pervasively modify all weights in the network. It was shown that the *neulonet*'s performance improves after the refinement training. However, one problem with backpropagation is that after training, the logic of the *net rules* may not be decipherable as the weights of the *net rules* have been altered. The trained *net rules* may not be reused easily. In view of this, a novel way of *neulonet* training by means of a genetic programming paradigm is now introduced.

3 Genetic Programming

Genetic programming [Koza, 1992] is an extension of the conventional genetic algorithm where instead of subjecting bit patterns to genetic evolution, the individuals in the gene population are computer programs. In the context of *neulonet* evolution, the problem statement may be stated as follows:

> "Given a set of input nodes, a library of *net rules*, an output node, and a set of examples containing instances of input values together with the corresponding output decision, apply genetic programming to construct a *neulonet* that represents a decision logic induced from the examples".

The solution may be carried out in three steps:

1. Generate an initial population of *neulonets* comprising of random compositions of *net rules* according to the available input nodes and the output node.

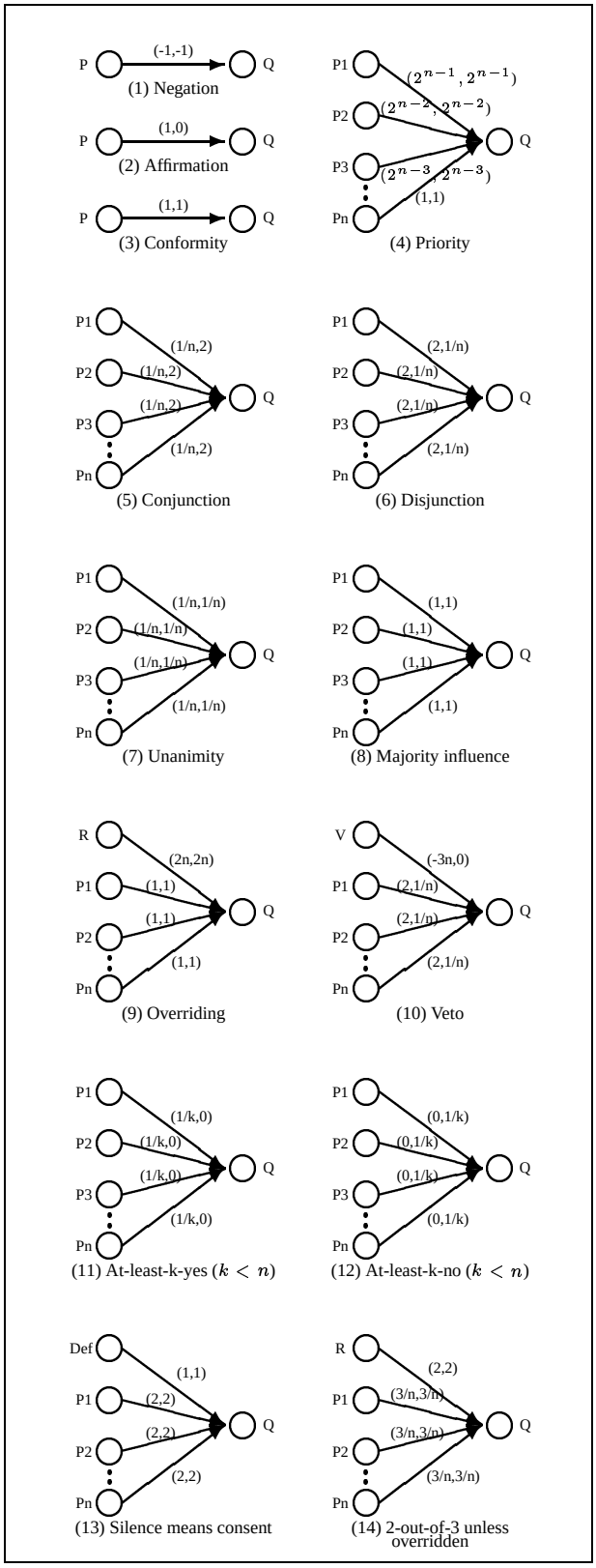


Figure 3: Examples of component *net rules*.

P	Q	P xor Q
(1,0)	(1,0)	(0,1)
(1,0)	(0,1)	(1,0)
(1,0)	(0,0)	(1,0)
(0,1)	(1,0)	(1,0)
(0,1)	(0,1)	(0,1)
(0,1)	(0,0)	(0,0)
(0,0)	(1,0)	(1,0)
(0,0)	(0,1)	(0,0)
(0,0)	(0,0)	(0,0)

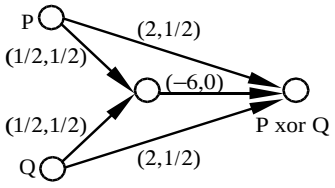


Figure 4: An XOR composite *net rule*.

2. Iteratively perform the following substeps until the termination criterion has been satisfied:
   (a) Fire each *neulonet* in the population and assign it a fitness value using the fitness measure.
   (b) Create a new population of *neulonets* by applying the following three operations to *neulonets* chosen with a probability based on fitness.
      i. Reproduce an existing *neulonet* by copying it into the new population.
      ii. Create two new *neulonets* from two existing ones by genetically recombining chosen parts using a crossover operation applied at a randomly chosen crossover point within each *neulonet*.
      iii. Create a new *neulonet* from an existing one using a mutation operation at a randomly chosen mutation point.
3. The *neulonet* that is identified to be the best individual is designated as the solution to the problem.

For efficiency, the *neulonet* structure that undergoes evolution is a labeled tree implemented using a prefix-ordered, jump-table approach [Keith and Martin, 1994]. The initial population of *neulonets* are generated in a similar fashion as Koza's [1992] "ramped-half-and-half" generative method. We use a normalized fitness measure $f(\epsilon, \sigma; \kappa)$ based on the errors produced by the *neulonet*, ϵ, as well as the size of the *neulonet*, σ. The factor, $\kappa \in [0, 1]$, is used to weigh the effects of accuracy over size in the fitness measure. A higher value for κ places more emphasis on finding an accurate solution at the expense of the size of the *neulonet*.

The crossover operation involves swapping the chosen parts of two *neulonets* with constraints imposed on the swapping process to preserve the syntactic integrity. For instance, swapping should not leave any dangling intermediate output nodes as such nodes will not be able to receive input values during firing. Figures 5 and 6 illustrate a crossover operation on two *neulonets* before and after the operation.

The mutation operation involves changes to a chosen *neulonet*. A random mutation point is picked such that the *neulonet* whose root is the mutation point is replaced by another randomly generated *neulonet*. Figure 7 shows the effect of the mutation operation on a *neulonet* with the "Conformity" *net rule* replaced by a "Priority" *net rule*.

The probabilities assigned are: (i) reproduction: 15%, (ii) crossover: 80%, and (iii) mutation: 5%.

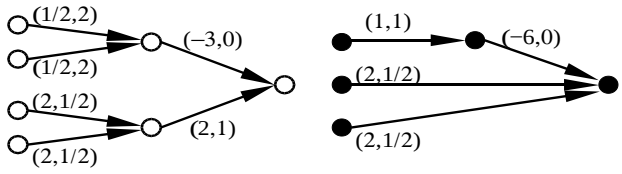


Figure 5: Two *neulonets* before crossover operation.

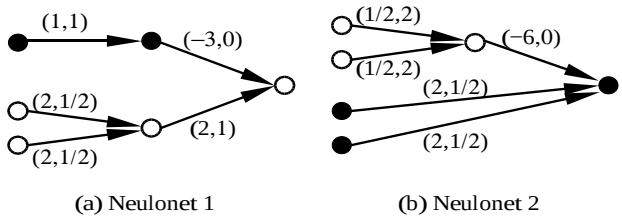


Figure 6: Two *neulonets* after crossover operation.

Note that the "Conformity" *net rule* appears redundant in the activation of a *neulonet* and might be deemed not to be doing anything significant in the evolutionary process. This kind of *net rule* is similar to an *intron* in biology, which is a chromosome that is never expressed and provides spacing between the genes [Angeline, 1994]. However, Levenick [1991] notes that *introns* are useful in genetic algorithms.

4 Proposed Method

In [Tan *et al.*, 1996], it was assumed that the knowledge engineer has some prior knowledge to construct a *neulonet*, which is later subjected to further refinement training. What if the knowledge engineer has only a set of examples without any other knowledge? One approach is to construct a *neulonet* with random weights and train it with the examples. The resultant *neulonet* is no different from an ordinary neural network, as it is difficult to interpret the logic semantics from the seemingly meaningless set of weights. Another approach is to solve a set of inequalities arising from the activation function in equation (1) [Teh, 1995]. The solution is not unique. Moreover, the process becomes more difficult with increasing number of inputs and it may not be always possible to find a set of interpretable weights to satisfy all the inequalities.

The genetic programming paradigm proposed here provides a third alternative for constructing a *neulonet* from examples. We shall illustrate the process using a simple example from the Space Shuttle Landing Domain [Michie, 1988].

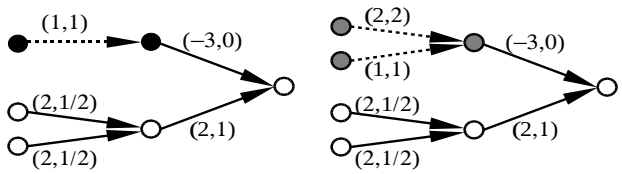


Figure 7: *Neulonet* with a mutated *net rule*.

This data set comprises 15 instances and 6 attributes. Table 8 shows each instance being classified as either to auto-land or not. To conform to the input requirements of the *neulonet* structure, every distinct attribute-value pair has a corresponding boolean attribute in a transformed data set. The valid values for these new attributes can either be yes, no or unknown. In our example, the 6-attribute data set transforms to a 16-attribute data set of boolean values.

Stable	Error	Sign	Wind	Magnitude	Vis	Class
?	?	?	?	?	no	auto
xstab	?	?	?	?	yes	noauto
stab	LX	?	?	?	yes	noauto
stab	XL	?	?	?	yes	noauto
stab	MM	nn	tail	?	yes	noauto
?	?	?	?	OutOfRange	yes	noauto
stab	SS	?	?	Low	yes	auto
stab	SS	?	?	Medium	yes	auto
stab	SS	?	?	Strong	yes	auto
stab	MM	pp	head	Low	yes	auto
stab	MM	pp	head	Medium	yes	auto
stab	MM	pp	tail	Low	yes	auto
stab	MM	pp	tail	Medium	yes	auto
stab	MM	pp	head	Strong	yes	noauto
stab	MM	pp	tail	Strong	yes	auto

Table 8: Space Shuttle Landing Domain data set.

For a particular *neulonet* evolution run, a 100% accurate solution was produced in generation 5 which consisted of 8 basic *net rules*. Further evolution produced another 100% accurate solution in generation 11 as shown in figure 9. This solution consisted of a "Priority" rule rooted at P, a "At-Least-2-Yes" rule rooted at L, and a "Negation" rule rooted at N.

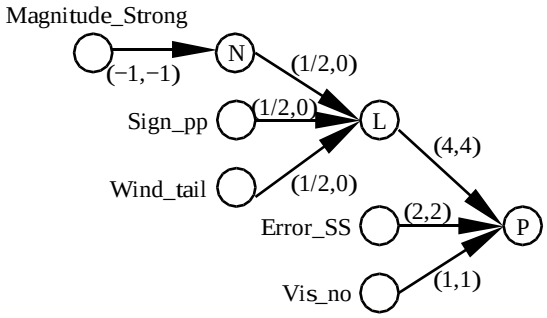


Figure 9: A solution to the Shuttle-Landing classification problem.

5 Extracting Rules from Neulonets

Extracting rules from neural networks has been studied by many researchers [Gallant, 1993; Gaudet, 1996; Setiono and Liu, 1996; Tan, 1997; Towell and Shavlik, 1993]. Existing work, however, basically utilizes standard logic, such as negation, conjunction and disjunction. The rules extracted are generally in the form of a decision tree equivalent to an AND/OR tree based on some classification of the example data. In our present work, the goal is to extract human decision logic from the *neulonets*. The extraction is in fact a straightforward process because the *neulonets* constructed are just a composition of *net rules* which by themselves fully express the human logic in use. As for the case of the Space Shuttle Landing Domain classification problem, the following *net rules* are easily identified from the solution in figure 9.

N : **Negation**(Magnitude_Strong)
L : **At-Least-Two-Yes**(N, Sign_pp, Wind_tail)
P : **Priority**(L, Error_SS, Visibility_no)

In layman terms, the decision to auto-land the space shuttle is biased towards any two or more positive factors for a "pp" sign, tail wind and a magnitude that is not strong. Otherwise, the presence (absence) of an "SS" error will result in a decision to auto-land (manual-land) the shuttle. If this error is unknown, then the decision to auto-land (manual-land) depends on a negative (positive) visibility.

6 Empirical Study and Discussion

Our analysis above shows that the proposed approach of using *net rules* in genetic programming should perform well for data sets which encompass some form of ordered logic reasoning. This analysis will be further confirmed via experiments. In particular, we wish to verify whether using an extended set of *net rules* as logical units is comparable to, if not better, than merely using a limited set of *net rules* comprising the standard boolean logic of negation, conjunction and disjunction (rules (1), (5) and (6) of figure 3). We selected data sets commonly used and publicly available from the UC Irvine data repository [Blake and Merz, 1998]. Data sets containing discrete attribute data types were transformed to an equivalent binary data set with one attribute for each attribute-value pair in the original set. Moreover, data sets containing continuous-valued attributes were pre-processed using a Chi2 Discretizer [Liu and Motoda, 1998] prior to the transformation. For the case of data sets having three or more class values, a separate data set that performs a boolean classification for each class value was created. Table 10 summarizes the data sets used in our experiments.

Data Set	# Instances	# Attributes
Shuttle-Landing-Control	15	16
Iris-setosa	150	123
Iris-versicolour	150	123
Iris-virginica	150	123
Monk-1	432	17
Monk-2	432	17
Monk-3	432	17
Voting-records	435	32
Breast-Cancer-Wisconsin	699	90
Mushroom	8124	125

Table 10: Data set summary. #Attributes denotes the number of boolean attributes in the transformed data set.

The entire library of *net rules* given in figure 3 was used for our experiments. Each of the *net rules* (4) to (13), were represented by their 2- and 3-arity equivalents, while a 4-arity "2-out-of-3 Unless Overridden" rule (14) was used. As a large initial population was required to cater for a wide variety of *neulonets* that could be evolved, we implemented a distributed parallel GP-system on an AP-3000 Fujitsu distributed memory parallel processing system consisting of 32 nodes using a ring-type connection [Niwa and Iba, 1996].

Data Set	*Neulonet*	Standard
Shuttle-landing	100% 3.0 (34.7)	100% 5.5 (48.3)
Iris-setosa	100% 1.0 (18.0)	100% 1.0 (12.0)
Iris-versicolour	99.8% 5.6 (230.0)	99.3% 8.0 (286.7)
Iris-virginica	99.6% 5.3 (154.0)	98.8% 3.0 (133.3)
Monk-1	100% 5.6 (31.6)	100% 4.7 (24.0)
Monk-2	99.8% 19.2 (456.4)	93.2% 20.8 (544.5)
Monk-3	100% 3.0 (26.6)	99.1% 4.0 (30.3)
Voting-records	99.6% 14.3 (356.3)	97.7% 13.0 (538.7)
Breast-cancer	99.5% 20.0 (552.0)	99.4% 20.2 (666.0)
Mushroom	100% 6.5 (46.2)	100% 6.25 (71.3)

Table 11: Experimental results depicting classification accuracy for the best individual. The numbers below the accuracy value denotes the number of decision nodes and (number of generations).

In our experiments, we used an initial population of 10,000, each having a depth of not more than four component *net rules*, and allowed to evolve to a depth of not more than 17. The evolutionary process would proceed until the classification accuracy and size of the fittest individual were unchanged for the last 100 generations. The weighting factor κ used in the fitness measure was set to 0.99. To further simplify the best evolved *neulonet*, a set of eight most commonly used simplification rules were applied recursively to identify component *net rules* for elimination or recombination. Results for evolving the best *neulonet* solution from an average of 10 runs using the set of *net rules* versus using standard boolean logic are presented in Table 11.

6.1 Classification Accuracy

In terms of the classification accuracy, *neulonet* evolution provided a comparable, if not better, result than standard logic evolution in all cases. This was especially true for data sets having an ordered logic reasoning as in the case of the Shuttle Landing Domain problem, or when the classification rules encompassed a "quantification" of standard boolean logic. For example, in the Monk-2 data set, the classification rule is given by [Thrun *et al.*, 1991] as follows:

$$\text{Exactly two of } a_i = 1 \;\; \forall i \in \{1, 2, 3, 4, 5, 6\}$$

Clearly, it would be difficult to achieve a high classification accuracy using only a composition of standard boolean logic units. However, in the case of *neulonet* evolution, the expressive power inherent in the set of *net rules* allowed for a more accurate solution tree to be evolved.

6.2 Solution Size

Due to the *neulonet's* ability to handle more complex decisions, *net rule* evolution provided more compact solutions than their standard boolean counterparts for the same classification accuracy, particularly for data sets having non-trivial classification rules. Using the best evolution runs for both empirical approaches in the Monk-2 data set, the profiles for the number of decision nodes is shown in figure 12 for accuracies between 70% to 100%. Observe that the profiles for both approaches are comparable in the case of classification accuracies of less than 90%, indicating that the classification rule learned thus far was still relatively simple. However, as the required accuracy increased, the apparent power of *neulonets* in deriving complex decision rules resulted in a significantly smaller solution size.

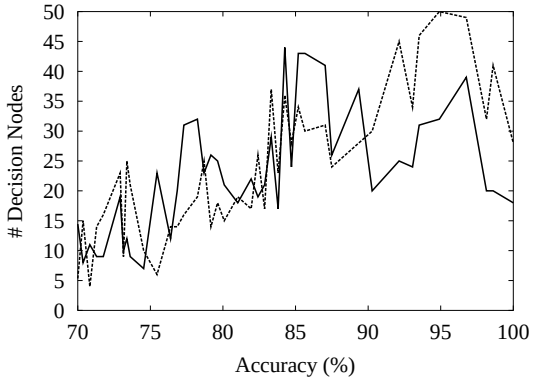


Figure 12: Profile for the number of decision nodes during *net rule*(solid-line) versus standard boolean logic(dotted-line) evolution.

6.3 Number of Generations Required

A general drawback in *net rule* evolution was the longer time needed to converge to an ideal solution due to the larger size of the component *net rule* library. It has to be noted that the experiment conducted actually gave the standard logic an advantage in that the rule set is small (only negation and 2- or 3-arity conjunctions and disjunctions), while the population sizes in both *neulonet* and standard logic evolution approaches were kept the same at 10,000 individuals. As a result, there were more opportunities for the small set of standard logic *rules* to quickly evolve to ideal individuals within the population. Thus for problems involving simpler decisions, standard logic evolution attained the required accuracy faster. However, there were cases in which *neulonet* evolution was faster. This was observed from the accuracy profile for the Monk-2 data set in figure 13.

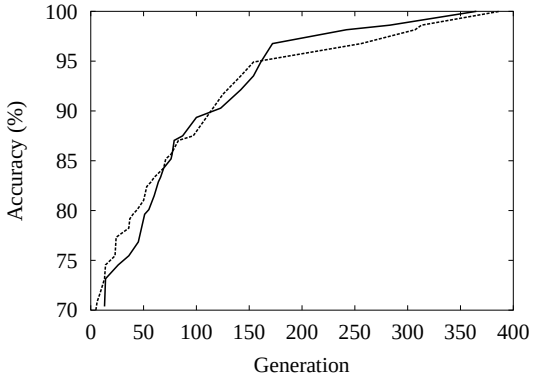


Figure 13: Accuracy profile for *net rule*(solid-line) versus standard boolean logic(dotted-line) evolution.

For accuracies of less than 95%, standard logic evolution required slightly less generations. However, as the demand in accuracy increased, it became increasingly difficult to evolve a solution using only standard logic rules. The added advantage in expressing complex rules during *neulonet* evolution, on the other hand, produced a faster rate of convergence.

7 Conclusion

Genetic programming proves to be an interesting paradigm in constructing *neulonets* with the prescribed human logic *net rules*. The paradigm is also amenable to refinement training. A knowledge engineer could start by constructing *neulonets* based on the human expert's prior knowledge. The constructed *neulonets* may then be subjected to the genetic programming evolutionary process. Thus step 1 of the genetic programming solution (i.e. generation of an initial random population) may be skipped and instead, the process will start from step 2 onwards.

This new mode of neural logic network learning, whether learning from scratch or learning by refining the existing network, preserves the logic semantics understandable for general human decision making. The size of the *net rule* library determines the granularity of the decision steps and the processing time. A large library enhances the capability in expressing nuances among different decisions but at the expense of a longer time to converge.

Further experiments will be carried out to investigate this trade-off issue. Variants of crossover and mutation processes, and fitness measures will also be studied. A long term plan in future is to apply genetic programming on fuzzy neural logic networks [Teh, 1995]. For such networks, the values for input and output ordered pairs are real-valued between 0 and 1. Genetic programming will thus be an attempt to evolve the best fuzzy decision rules. We envision an even more exciting horizon of fuzzy *neulonet* learning with genetic programming.

References

[Angeline, 1994] Peter J. Angeline. Genetic Programming and Emergent Intelligence, in Kinnear, K.E, ed., 1994. *Advances in Genetic Programming*. Cambridge, Mass.: MIT Press, 75–97.

[Armstrong and Thomas, 1996] William W. Armstrong and Monroe M. Thomas. Adaptive Logic Networks. In *The Handbook of Neural Computation*, Fieseler, E. and Beale E., eds., Institute of Physics Publishing and Oxford University Press USA.

[Blake and Merz, 1998] Catherine L. Blake and C. J. Merz. UCI Repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html` Irvine, CA: University of California, Department of Information and Computer Science.

[Gallant, 1993] Stephen I. Gallant. Extracting Rules from Neural Networks. In *Neural Network Learning and Expert Systems*, chapter 17, 315–330.

[Gaudet, 1996] Vincent C. Gaudet. Genetic Programming of Logic-Based Neural Networks. In *Genetic Algorithms for Pattern Recognition*, Pal, S.K. and Wang, P.P., eds., Boca Raton: CRC Press, 213–226.

[Golubski and Feuring, 1999] Wolfgang Golubski and Thomas Feuring. Evolving Neural Network Structures by Means of Genetic Programming. In *Genetic Progamming: Proceedings of the Second European Workshop, EuroGP'99*, Ricardo Poli, et al, eds., Springer-Verlag Lecture Notes in Computer Science, Vol. 1598, 211–220.

[Keith and Martin, 1994] Mike J. Keith and Martin C. Martin. Genetic Programming in C++: Implementation Issues, in Kinnear, K.E. ed., 1994. *Advances in Genetic Programming*. Cambridge, Mass.: MIT Press, 285–310.

[Koza, 1992] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, 1992.

[Levenick, 1991] James R Levenick. Inserting introns improves genetic algorithm success rate: Taking a cue from biology. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, Belew, R.K; and Booker, L.B., eds., San Mateo, CA: Morgan Kaufmann Publishers Inc., 123–127.

[Liu and Motoda, 1998] Huan Liu and Hiroshi Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. The Kluwer International Series in Engineering and Computer Science, Vol. 454, 161–168, Kluwer Academic Publishers, Boston.

[Michie, 1988] Donald Michie. The Fifth Generation's Unbridged Gap. In *The Universal Turing Machine: A Half-Century Survey*, Rolf Herken (ed.), 466–489, Oxford University Press.

[Niwa and Iba, 1996] Tatsuya Niwa and Hitoshi Iba. Distributed Genetic Programming: Empirical Study and Analysis. In *Genetic Programming : Proceedings of the First Annual Conference 1996*, edited by John, R. Koza *et al.*, 339–344. Cambridge, Mass. : MIT Press.

[Setiono and Liu, 1996] Rudy Setiono and Huan Liu. Symbolic Representation of Neural Networks, *Computer*, 29(3), 71–77.

[Tan, 1997] Ah Hwee Tan. Cascade ARTMAP: Integrating Neural Computation and Symbolic Knowledge Processing, IEEE Transactions on Neural Networks, 8(2): 237–250.

[Tan *et al.*, 1996] Chew Lim Tan, Tong Seng Quah, and Hoon Heng Teh. An Artificial Neural Network that Models Human Decision Making, *Computer*, 29(3), 64–70.

[Teh, 1995] Hoon Heng Teh. *Neural Logic Network, A New Class of Neural Networks Called Neural Logic Networks*, Singapore: World Scientific.

[Thrun *et al.*, 1991] Sebastian Thrun, *et al*. The MONK'S Problems: A Performance Comparison of Different Learning Algorithms. Technical Report CMU–CS–91–197, Carnegie Mellon University, Pittsburgh, PA.

[Towell and Shavlik, 1993] Geoffrey G. Towell and Jude W. Shavlik. Extracting Refined Rules from Knowledge-Based Neural Networks, *Machine Learning*, 13(1), 71–101.

Sensitivity Analysis of Multilayer Perceptron*

Daniel S. Yeung, Xuequan Sun, and Xiaoqin Zeng
Department of Computing, The Hong Kong Polytechnic University

Abstract

Sensitivity analysis of neural network is useful for network design. Piché used a stochastic model to describe the Multilayer Perceptron (MLP), but it doesn't match the true MLP closely, and too severe limitations are imposed on both input and weight perturbations. This paper attempts to generalize Piché's stochastic model of MLP, and derive an universal expression of MLP's sensitivity for all sigmoidal activation functions, without any restriction on input and output perturbations. The effects of network design parameters such as the number of layer, the number of neuron per layer and the chosen activation function are analyzed, and they provide useful information for network design decision-making. Furthermore, we use our sensitivity expression to design MLP for a given application. It can help to design the network structure, as well as the training of MLP.

1 Introduction

The sensitivity analysis of neural network has been investigated for over 30 years. In 1962, Hoff used n-dimensional geometry approach to analyze the sensitivity of Adaline [Hoff, 1962]. Glanz [1965] simplified Hoff's results. After two decades, Winter [1989] derived an analytical expression for the probability of error in Madaline caused by weight perturbation. Stevenson continued Winter's work, and established the sensitivity of Madaline to weight error in 1990 [Stevension, 1990a; Stevension *et al.*, 1990b]. Cheng and Yeung [1999] use this geometrical arguments to analyze the sensitivity of Neocognitrons. However, the their results are only fit for neurons with threshold activation function and binary inputs, and they are not applicable to other continuous activation functions. Yeung and Wang [1999] try to apply the method proposed in [Cheng and Yeung, 1999] to the sensitivity analysis of the MLP. Earlier in 1992, Choi and Choi [1992] present a thorough idea of sensitivity analysis of the MLP with differentiable activation functions. Although their analysis is systematic, the result is only applicable when the network has been trained and the inputs are known, so it cannot be used for network design and construction.

Piché [1992;1995] uses statistical approach to relate the output error and the change of weights in Madaline with several activation functions such as linear, sigmoid and threshold. But Piché's stochastic model is a simplification of the true MLP, and his result holds true only when the input and weight perturbations are small enough.

This paper proposes an improved stochastic model which is better suited for the true MLP. We also present a function approximation method to derive an universal analytical expression of sensitivity for MLP with sigmoidal activation function. We assume no restriction on the amplitude of the input and weight perturbations. Using this function approximation method, one can use three parameters to characterize the activation function. So not only the effects of input and weight perturbations, the number of neuron and the number of layer, but also that of the activation function on the sensitivity of MLP can be analyzed by the universal analytical expression.

2 The Improved Stochastic Model of MLP

According to Piché's stochastic model, all neurons have the same activation function. All network inputs, weights, input perturbations and weight perturbations are respectively independent identically distributed random (iidr) variables with zero mean; inputs and input errors, inputs and weight errors, weights and inputs, weights and weight errors are statistically independent from each other. This model has the following weaknesses:

1. The bias input is omitted. This is a basic component of MLP;
2. The expectations of all network inputs are zeros. Many applications don't satisfy this condition;
3. All network inputs have the same variance. It is also difficult for many applications to satisfy this requirement;
4. The correlations between the inputs and their corresponding perturbations are not considered. Even if the inputs and their associative perturbations in the first layer are independent, there still exist correlations between the inputs and their corresponding perturbations in the successive layers of MLP.

* This research work is supported by a Hong Kong Polytechnic University research grant number G-S546 and a Hong Kong CERG number 5114/97E

Our improved stochastic model of MLP overcomes the above shortcomings in the following ways:

1. All network inputs are independent random variables with their own expectations and variances. The expectation of the bias input is 1, and its variance is 0;
2. All input perturbations are independent random variables with zero mean and their own variances. The variance of the bias input perturbation is 0. The correlations between network inputs and their corresponding perturbations are expressed by their correlation coefficients;
3. All weights and their perturbations are independent random variables with zero mean and their own variances. The correlations between weights and weight perturbations are also expressed by their correlation coefficients;
4. Except the correlations between inputs and input perturbations, weights and weight perturbations, all variables are independent from each other.

Our model is a better description of the true MLP.

3 Definitions

First, some notations are introduced. The layers are indexed from *0* to *L*. The *0* layer is the input layer, and the *Lth* layer is the output layer. The number of nodes in layer *l* is denoted by N^l. The function $g(\cdot)$ denotes the activation function. Neuron is the basic element of MLP, so one can first investigate the sensitivity of a single neuron, and then the whole MLP.

Suppose X^l, ΔX^l are the input and input perturbation vectors of neuron *i* in layer *l*, W_i^l, ΔW_i^l are its weight and weight perturbation vectors. D(.) denotes the variance.

Definition 1: The absolute sensitivity of the neuron to input and weight perturbations is given by

$$S_i^l \overset{def}{=} \sqrt{D(g((W_i^l+\Delta W_i^l)\cdot(X^l+\Delta X^l))-g(W_i^l\cdot X^l))}.$$

Definition 2: The relative sensitivity of the neuron to input and weight perturbations is

$$\mathrm{Re}\,l.S_i^l \overset{def}{=} \sqrt{\frac{D(g((W_i^l+\Delta W_i^l)^T\cdot(X^l+\Delta X^l))-g({W_i^l}^T\cdot X^l))}{D(g({W_i^l}^T\cdot X^l))}}.$$

Since the outputs of MLP are the outputs of the neurons in the output layer, we can use the output neurons to represent the outputs of the MLP.

Definition 3: The absolute sensitivity of MLP to input and weight perturbations is

$$S \overset{def}{=} (S_1^L, S_2^L, \cdots, S_{N_L}^L)^T.$$

Definition 4: The relative sensitivity of MLP to input and weight perturbations is

$$\mathrm{Re}\,l.S \overset{def}{=} (\mathrm{Re}\,l.S_1^L, \mathrm{Re}\,l.S_2^L, \cdots, \mathrm{Re}\,l.S_{N_L}^L)^T.$$

4 Function Approximation Approach

If the sensitivity is directly calculated by using the above definitions, it will be very difficult to obtain an analytical expression for the sensitivity unless we restrict the amplitudes of the input and weight perturbations. But this would severely limit the range of application of our sensitivity analysis. So a function approximation approach is proposed to overcome this problem. This is based on the idea that a given class of activation functions can be characterized by a general function expression using a set of common characteristics, i.e., parameters of this expression, which should satisfy the followings:

1) They can be used to approximate the activation function concerned, and
2) They should have the form convenient for sensitivity computation.

The threshold and sigmoidal activation functions are commonly used for the MLPs. It can be characterized by three factors: its maximum value, minimum value and obliquity, which are denoted by *B*, *C* and *A* respectively, Different values of *B*, *C* and *A* represent different activation functions. The main idea of the function approximation approach is to use another function $g_{A,B,C}(x)$ to approximate $g(x)$. The function $g_{A,B,C}(x)$ has the following form

$$g_{A,B,C}(x) = \frac{B+C}{2} + f_{A,B,C}(x), \tag{1}$$

where

$$f_{A,B,C}(x) = \begin{cases} \dfrac{B-C}{2}(1-e^{-Ax^2}), & x \ge 0 \\ -\dfrac{B-C}{2}(1-e^{-Ax^2}), & x < 0 \end{cases}, \tag{2}$$

The objective of function approximation is to determine the three factors *A*, *B* and *C* of $g(x)$, so that the distance between the two functions $g(x)$ and $g_{A,B,C}(x)$ is minimized.

The factors *B* and *C* are determined by

$$B = \lim_{x\to+\infty} g(x), \tag{3}$$

$$C = \lim_{x\to-\infty} g(x). \tag{4}$$

The factor *A* can be found by solving the following problem

$$\min_A \sqrt{\int_{-\infty}^{+\infty} (g(x) - g_{A,B,C}(x))^2 dx}. \tag{5}$$

Because $g(x)$ and $g_{A,B,C}(x)$ are antisymmetric about a certain point on the y-axis, problem (5) is equivalent to

$$\min_A \sqrt{\int_{0}^{+\infty} (g(x) - g_{A,B,C}(x))^2 dx}. \tag{6}$$

The commonly used sigmoidal activation function has the form $g(x) = \dfrac{B+Ce^{-\lambda x}}{1+e^{-\lambda x}}$, we have $A = \dfrac{\sqrt{\pi}}{4}\lambda^2$.

5 Sensitivity of a Single Neuron

According to the definitions of absolute and relative sensitivity, we need first derive the variances of the output and the output error of a neuron. The output of the unit *j* in

layer l is

$$y_i^l = g(W_i^{l^T} \cdot X^l). \quad (7)$$

Its output error is

$$\Delta y_i^l = g((W_i^l + \Delta W_i^l)^T \cdot (X^l + \Delta X^l)) - g(W_i^{l^T} \cdot X^l). \quad (8)$$

The covariances between inputs and input perturbations, weights and weight perturbations are $\sigma^l_{x_i \Delta x_i} = \rho^l_{x_i \Delta x_i} \sigma^l_{x_i} \sigma^l_{\Delta x_i}$ and $\sigma^l_{w_{ij} \Delta w_{ij}} = \rho^l_{w_{ij} \Delta w_{ij}} \sigma^l_{w_{ij}} \sigma^l_{\Delta w_{ij}}$, where ρ denotes the correlation coefficient.

Let σ^2 and μ represent variance and expectation. We have

$$E(W_i^{l^T} \cdot X^l) = E(\Delta W_i^{l^T} \cdot X^l + W_i^{l^T} \cdot \Delta X^l + \Delta W_i^{l^T} \cdot \Delta X^l) = 0, \quad (9)$$

$$D(W_i^{l^T} \cdot X^l) = \sum_{j=1}^{N^{l-1}} (\sigma_{x_j}^{l\ 2} + \mu_{x_j}^{l\ 2}) \sigma_{w_{ij}}^{l\ 2} + \sigma_{w_{i0}}^{l\ 2}, \quad (10)$$

$$D(\Delta W_i^{l^T} \cdot X^l + W_i^{l^T} \cdot \Delta X^l + \Delta W_i^{l^T} \cdot \Delta X^l), \quad (11)$$

$$= \sum_{j=1}^{N^{l-1}} ((\sigma_{x_j}^{l\ 2} + \mu_{x_j}^{l\ 2}) \sigma_{\Delta w_{ij}}^{l\ 2} + \sigma_{\Delta x_j}^{l\ 2} \sigma_{w_{ij}}^{l\ 2} + \sigma_{\Delta x_j}^{l\ 2} \sigma_{\Delta w_{ij}}^{l\ 2}) \quad (12)$$

$$+ \sum_{j=1}^{N^{l-1}} (2\sigma^l_{x_j, \Delta x_j} \sigma^l_{w_{ij}, \Delta w_{ij}} + 2\sigma^l_{x_j, \Delta x_j} \sigma_{\Delta w_{ij}}^{l\ 2} + 2\sigma^l_{\Delta x_j}{}^2 \sigma^l_{w_{ij}, \Delta w_{ij}}) + \sigma_{\Delta w_{i0}}^{l\ 2}$$

Let

$$\sigma_R = \sqrt{D(W_i^{l^T} \cdot X^l)}, \quad (13)$$

$$\sigma_P = \sqrt{D(\Delta W_i^{l^T} \cdot X^l + W_i^{l^T} \cdot \Delta X^l + \Delta W_i^{l^T} \cdot \Delta X^l)}, \quad (14)$$

$$U = \frac{W_i^{l^T} \cdot X^l}{\sigma_R}, \quad (15)$$

$$V' = \frac{\Delta W_i^{l^T} \cdot X^l + W_i^{l^T} \cdot \Delta X^l + \Delta W_i^{l^T} \cdot \Delta X^l}{\sigma_P}. \quad (16)$$

According to the Lindeberg Central Limit Theorem, when N^{l-1} is large enough, u and v' are normally distributed with mean 0 and variance 1, i.e., $u \sim N(0,1)$ and $v \sim N(0,1)$. The correlation coefficient between u and v is

$$\rho_{U,V'} = \frac{\sum_{j=1}^{N^{l-1}} (\sigma^l_{w_{ij}, \Delta w_{ij}} (\sigma_{x_j}^{l\ 2} + \mu_{x_j}^{l\ 2}) + \sigma^l_{x_j, \Delta x_j} \sigma_{w_{ij}}^{l\ 2} + \sigma^l_{x_j, \Delta x_j} \sigma^l_{w_{ij}, \Delta w_{ij}}) + \sigma^l_{w_{i0}, \Delta w_{i0}}}{\sigma_R \sigma_P}. \quad (17)$$

Let

$$V = \frac{-\rho_{U,V'}}{\sqrt{1-\rho^2_{U,V'}}} U + \frac{1}{\sqrt{1-\rho^2_{U,V'}}} V', \quad (18)$$

we get $V \sim N(0,1)$, and $\rho_{U,V} = 0$.

We introduce two notations

$$\alpha = \sigma_R + \sigma_P \rho_{U,V'}, \quad (19)$$

$$\beta = \sigma_P \sqrt{1-\rho^2_{U,V'}}. \quad (20)$$

Then

$$W_i^{l^T} \cdot X^l = \sigma_R U, \quad (21)$$

$$(W_i^l + \Delta W_i^l)^T \cdot (X^l + \Delta X^l) = \alpha U + \beta V. \quad (22)$$

The distribution function of u, and the joint distribution function of u and v may be expressed as

$$f_U(u) = \frac{1}{2\pi} e^{-\frac{1}{2}u^2}, \quad (23)$$

$$f(u,v) = \frac{1}{2\pi} e^{-\frac{1}{2}(u^2+v^2)}. \quad (24)$$

Now, the expectations of output and output error are

$$E(y_j^l) = \int_{-\infty}^{+\infty} g(\alpha u) \cdot \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} du, \quad (25)$$

$$E(\Delta y_j^l) = \int_{-\infty-\infty}^{+\infty+\infty} (g(\alpha u + + \alpha\beta v) - g(\alpha u)) \cdot \frac{1}{2\pi} e^{-\frac{1}{2}(u^2+v^2)} du dv \quad (26)$$

The variances of output and output error are

$$D(y_j^l) = \int_{-\infty}^{+\infty} g^2(\alpha u) \cdot \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} du, \quad (27)$$

$$D(\Delta y_j^1) = \int_{-\infty-\infty}^{+\infty+\infty} (g(\alpha u + \alpha\beta v) - g(\alpha u))^2 \frac{1}{2\pi} e^{-\frac{1}{2}(u^2+v^2)} du dv. \quad (28)$$

Using our function approximation approach and the following polar coordinate transformation

$$\begin{cases} u = r\cos\theta \\ v = r\sin\theta \end{cases}, r \geq 0, \theta \in [0, 2\pi], \quad (29)$$

Formulae (25), (26), (27) and (28) become

$$Ey_i^l == \frac{B+C}{2}, \quad (30)$$

$$E\Delta y_i^l = 0, \quad (31)$$

$$Dy_i^l = \frac{(B-C)^2}{4} \left(1 - \frac{2}{\sqrt{2A\sigma_R^2 + 1}} + \frac{1}{\sqrt{4A\sigma_R^2 + 1}}\right), \quad (32)$$

$$D\Delta y_i^l = \frac{(B-C)^2}{4} P_i^l. \quad (33)$$

According to Definitions 1 and 2,

$$S_i^l = \frac{B-C}{2} \sqrt{P_i^l}, \quad (34)$$

$$\mathrm{Re}\, l.S_i^l = \sqrt{\frac{P_i^l}{(1 - \frac{2}{\sqrt{2A\sigma_R^2 + 1}} + \frac{1}{\sqrt{4A\sigma_R^2 + 1}})}}, \quad (35)$$

where, let T=

$$(arctg \frac{[A(\alpha^2+\beta^2+\sigma_R^2)+1]tg(\frac{1}{2}arctg(\frac{\alpha^2+\sigma_R^2-\beta^2}{2\alpha\beta})+arctg(\frac{\beta}{\alpha})) - A\sqrt{(\alpha^2+\sigma_R^2-\beta^2)^2+4\alpha^2\beta^2}}{\sqrt{4A^2\sigma_R^2\beta^2+2A\alpha^2+2A\beta^2+2A\sigma_R^2+1}}$$

$$- arctg \frac{[A(\alpha^2+\beta^2+\sigma_R^2)+1]tg(\frac{1}{2}arctg(\frac{\alpha^2+\sigma_R^2-\beta^2}{2\alpha\beta})) - A\sqrt{(\alpha^2+\sigma_R^2-\beta^2)^2+4\alpha^2\beta^2}}{\sqrt{4A^2\sigma_R^2\beta^2+2A\alpha^2+2A\beta^2+2A\sigma_R^2+1}}),$$

$$P_i^l = \frac{4}{\pi} arctg(\frac{\beta}{\alpha}) + \frac{1}{\sqrt{4A\sigma_R^2+1}} + \frac{1}{\sqrt{4A(\alpha^2+\beta^2)+1}}$$

$$- \frac{2}{\sqrt{4A^2\sigma_R^2\beta^2+2A\alpha^2+2A\beta^2+2A\sigma_R^2+1}}$$

$$- \frac{4}{\pi\sqrt{2A\sigma_R^2+1}} arctg(\frac{\beta\sqrt{2A\sigma_R^2+1}}{\alpha})$$

$$- \frac{4}{\pi\sqrt{2A\alpha^2+2A\beta^2+1}} arctg(\frac{\beta\sqrt{2A\alpha^2+2A\beta^2+1}}{\alpha})$$

$$+\frac{4T}{\pi\sqrt{4A^2\sigma_R^2\beta^2+2A\alpha^2+2A\beta^2+2A\sigma_R^2+1}} \tag{36}$$

For each neuron in a given layer, its inputs and the associated errors are just the outputs and their associated errors of the former layer. So $y_j^{l-1}=x_j^l$, $\Delta y_j^{l-1}=\Delta x_j^l$. Now the covariance between y_j^{l-1} and Δy_j^{l-1} can be calculated by

$$\sigma_{x_i\Delta x_i}^{l+1}=\sigma_{y_i\Delta y_i}^{l}=\frac{(B-C)^2}{4}Q_i^l, \tag{37}$$

where

$$\begin{aligned}Q_i^l=&-\frac{2}{\pi}arctg(\frac{\beta}{\alpha})+\frac{1}{\sqrt{2A\sigma_R^2+1}}-\frac{1}{\sqrt{4A\sigma_R^2+1}}-\frac{1}{\sqrt{2A(\alpha^2+\beta^2)+1}}\\&+\frac{1}{\sqrt{4A^2\sigma_R^2\beta^2+2A\alpha^2+2A\beta^2+2A\sigma_R^2+1}}\\&+\frac{2}{\pi\sqrt{2A\sigma_R^2+1}}arctg(\frac{\beta\sqrt{2A\sigma_R^2+1}}{\alpha})\\&+\frac{2}{\pi\sqrt{2A\alpha^2+2A\beta^2+1}}arctg(\frac{\beta\sqrt{2A\alpha^2+2A\beta^2+1}}{\alpha})\\&-\frac{2T}{\pi\sqrt{4A^2\sigma_R^2\beta^2+2A\alpha^2+2A\beta^2+2A\sigma_R^2+1}}.\end{aligned} \tag{38}$$

All formulae are derived exactly except the approximation of the activation function. So there is no need for us to restrict the input and weight perturbations to be very small. The network sensitivity expressions are expressed in terms of *A*, *B* and *C*, i.e., the activation function itself. This enables us to investigate the effect of a given activation function on the network sensitivity.

6 Sensitivity of MLP

Since the sensitivity of a MLP is defined to be the sensitivity vector of its output neurons, it will be necessary to calculate the sensitivity of neurons from the input layer to the output layer successively.

The algorithm to calculate the sensitivity of the network is given by

1) Based on the application requirement, determine the expectations and variances of network inputs $\mu_{x_i}^0,\sigma_{x_i}^{0\,2}$, input perturbation ratios $\frac{\sigma_{\Delta x_i}^{0\,2}}{\sigma_{x_i}^{0\,2}}$, the correlation coefficients between inputs and input perturbations $\rho_{x_i\Delta x_i}^0$, the variances of weights $\sigma_{w_{ij}}^{l\,2}$, weight perturbation ratios $\frac{\sigma_{\Delta w_{ij}}^{l\,2}}{\sigma_{w_{ij}}^{l\,2}}$ and the correlation coefficients between weights and weight perturbations $\rho_{w_{ij}\Delta w_{ij}}^l$;
2) Using function approximation approach to determine *A*, *B* and *C* of the activation function;
3) Let l=1, $\sigma_x^{l\,2}=\sigma_x^2$, $\sigma_{\Delta x}^{l\,2}=\sigma_{\Delta x}^2$;
4) For all neurons in layer *l*, calculate $\sigma_{y_i}^{l\,2}$ and $\sigma_{\Delta y_i}^{l\,2}$ by applying formulae (32) and (33), and then $\sigma_{y_i\Delta y_i}^l$, by Formula (39);
5) If $l<L$, let $l=l+1$, $\sigma_{x_i}^{l\,2}=\sigma_{y_i}^{l-1\,2}$, $\sigma_{\Delta x_i}^{l\,2}=\sigma_{\Delta y_i}^{l-1\,2}$, $\sigma_{x_i\Delta x_i}^l=\sigma_{y_i\Delta y_i}^{l-1}$, then goto step (4); else
6) Calculate S_i^L and $\mathrm{Re}\,l.S_i^L$ by formulae (34) and (35), then the absolute and relative sensitivities of MLP is $S=(S_1^L,S_2^L,\cdots,S_{N_l}^L)^T$, $\mathrm{Re}\,l.S=(\mathrm{Re}\,l.S_1^L,\mathrm{Re}\,l.S_2^L,\cdots,\mathrm{Re}\,l.S_{N_l}^L)^T$, and the program is terminated.

7 Effects of Parameters on Sensitivity

Many factors affect the sensitivity of the MLP. These factors include the network inputs and input perturbations, weights and weight perturbations, and network architecture parameters such as the number of layer, the number of neurons in each layer and the activation function chosen. How these factors affect the sensitivity of MLP is an important criteria for neural network design in support of a particular application.

The experimental results are shown in figures 1 to 13. The main results are:

1) The input and weight perturbation ratios are the two most important factors affecting the sensitivity of the MLP. The absolute and relative sensitivities increase when these two ratios do.
2) The network sensitivities increase with the absolute value of the input expectation and the input variance. The normalization of network input can help us decrease the network sensitivity;
3) The increase of weight variance causes the network sensitivity to increase. So we should limit the amplitude of the weight in training process;
4) When the number of neurons increases, the absolute and relative sensitivities increase, but when it is significantly large, the two sensitivities will nearly remain constant. The neuron number of later layers has more influence upon the absolute sensitivity than that of the previous layer.
5) The absolute and relative sensitivities increase with the number of layer. When the number of layer is small, these two sensitivities increase rapidly. Otherwise they become flat. When the weight perturbation ratio increases, the absolute and relative sensitivities increase as well, i.e., if the sensitivity is supposed to be less than a certain value, the number of layer could be small when the weight perturbation ratio is large.
6) The larger the obliquity *A* of the activation function is, the more sensitive the MLP is. The sensitivities increase with *B*-*C*. So the threshold MLP is the most sensitive one, and The activation function $\frac{1}{1+e^{-x}}$ is more suitable than *tanh(x)*.

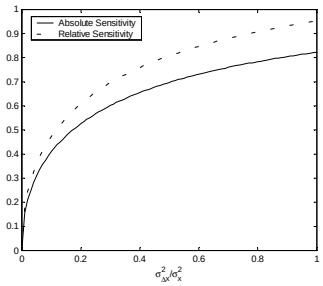


Figure 1: The absolute and relative Sensitivities of MLP versus input perturbation ratio.

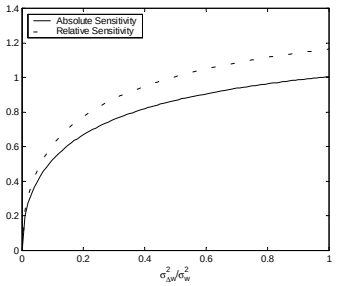


Figure 2: The absolute and relative Sensitivities of MLP versus the weight perturbation ratio.

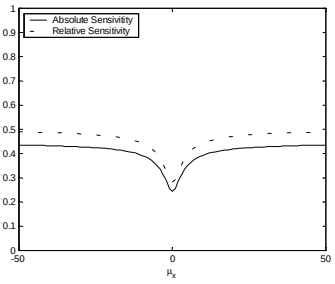


Figure 3: The absolute and relative Sensitivities of MLP versus the expectation of network input.

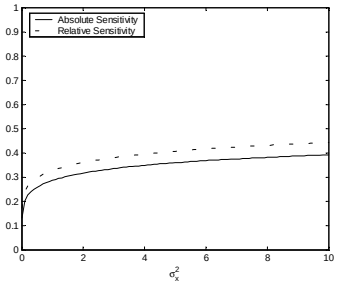


Figure 4: The absolute and relative Sensitivities of MLP versus the variance of network input.

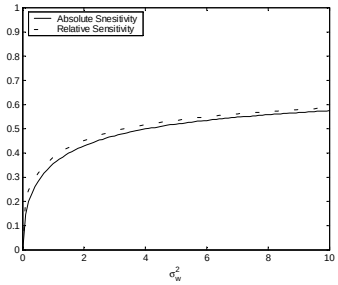


Figure 5: The absolute and relative Sensitivities of MLP versus the weight variance.

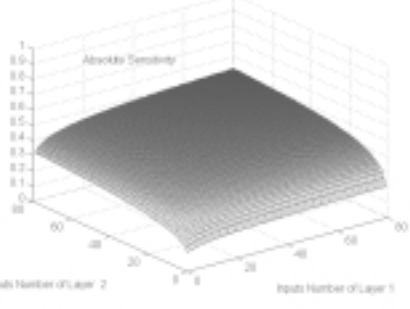


Figure 6: The absolute sensitivity of MLP versus the number of neurons in each layer.

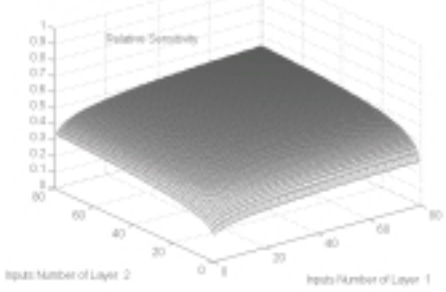


Figure 7: The relative sensitivity of MLP versus the number of neurons in each layer.

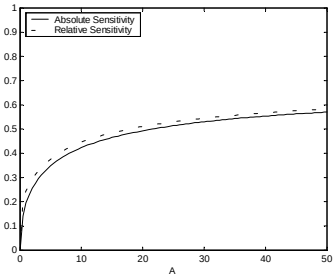


Figure 8: The absolute and relative sensitivities of a neuron versus obliquity A of the activation function.

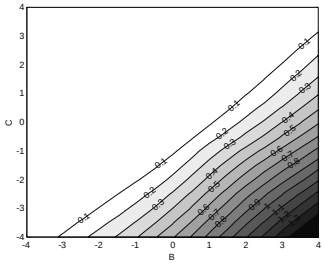


Figure 9: The absolute sensitivity of a neuron versus the max and min value of the activation function.

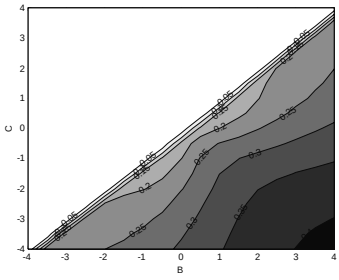


Figure 10: The relative sensitivity of a neuron versus the max and min value of the activation function.

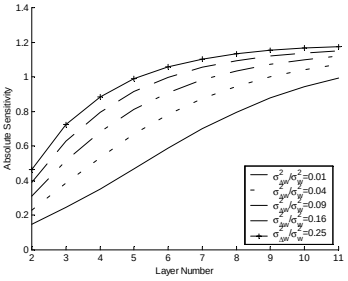


Figure 11: The absolute sensitivity of MLP versus the number of layer.

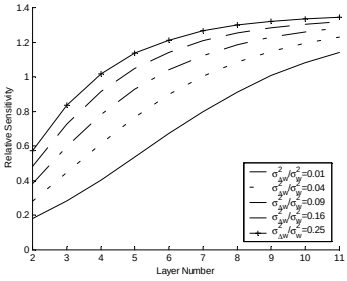


Figure 12: The relative sensitivity of MLP versus the number of layer.

8 The Design of MLP

In order to simplify the network design, we suppose that all weights have the same variances, and all weight perturbation ratios are identical. So all neurons in the same layer have the same sensitivity. We can use a output neuron's sensitivity to represent the network sensitivity vector.

For a given application, suppose parameters *L*, *B* and *C* are determined by the network designer, from the sample set

and the application environment, we can estimate the following parameters: N_0, N_L, $\mu_{x_i}^0$, $\sigma_{x_i}^{0\,2}$, $\rho_{x_i,\Delta x_i}^0$, $\frac{\sigma_{\Delta x_i}^{0\,2}}{\sigma_{x_i}^{0\,2}}$, $\rho_{w,\Delta w}$, $\frac{\sigma_{\Delta w}^2}{\sigma_w^2}$. Our goal is to find the unknown parameters N_1, …, N_{L-1}, A and σ_w^2, such that $S_R < S_R^*$, where S_R^* is a given, acceptable relative sensitivity.

Our solution is: first select a possible range for each of the unknown parameters; then using our sensitivity calculation algorithm to compute the mapping from $\{N_1,\cdots,N_{L-1},A,\sigma_w^2\}$ to $\{S_R\}$; finally, according to the mapping, determine the region $\{N_1,\cdots,N_{L-1},A,\sigma_w^2\}$ that satisfies $S_R < S_R^*$.

We present the following example to show how to design a three-layer MLP.

Example: The input and output numbers are 10, and all inputs are independent random variables that are uniformly distributed over [-1,1]. Suppose the correlation coefficient between weights and weight perturbations are 0.01. Construct a three-layer MLP with input and weight perturbation ratios less than *1%*, and the relative sensitivity of this MLP should be less than *10%*.

For this example, the known parameters are L=2, $N_0 = N_2 = 10$, $\mu_{x_i}^0 = 0$, $\sigma_{x_i}^{0\,2} = 4/12$, $\rho_{x_i,\Delta x_i}^0 = 0$, $\rho_{w,\Delta w} = 0.01$, $\frac{\sigma_{\Delta x_i}^{0\,2}}{\sigma_{x_i}^{0\,2}} = \frac{\sigma_{\Delta w}^2}{\sigma_w^2} = 0.01, S_R^* = 0.1$. Suppose B=1, C=0, and the possible ranges for parameters A, N_1 and σ_w^2 are [0, 1.7725], [1, 100] and [0, 25] respectively. Using our sensitivity algorithm to calculate the mapping over the above ranges. Here, we plot two contour maps of S_R. When the obliquity A is large and small, the contour lines of S_R with respect to N_1 and σ_w are shown in figures 13 and 14 respectively. The region below the contour line at $S_R = 0.1$ is our design result.

Our sensitivity analysis can be used to determine the network structure numerically. Furthermore, it can estimate the permitted weight range for network training. We find that the larger N_1 is, the smaller the maximum permitted weight variance is. That is to say, if we select larger number of neurons in the hidden layer, we must restrict the weight variance to be less in the training of MLP; when the obliquity of activation function increases, the solution region reduces. This means that if we select activation function with smaller obliquity, we can permit larger number of neurons in the hidden layer and larger weight variance in the network training.

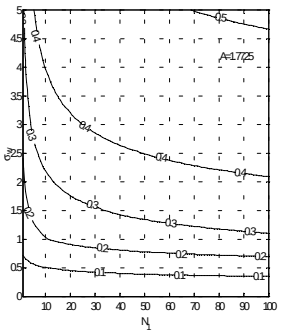


Figure 13: The contour plot of relative sensitivity of three-layer MLP with respect to N_1 and σ_w. A=1.7725.

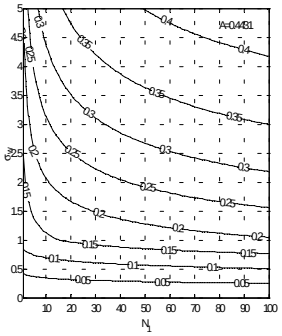


Figure 14: The contour plot of relative sensitivity of three-layer MLP with respect to N_1 and σ_w. A=0.4431.

9 Conclusion

In this paper an improved stochastic model of MLP is proposed to better describe the, and the function approximation method is used to obtain an universal expression of the network sensitivity for sigmoidal activation functions. We use the expression of sensitivity to investigate the effects of network parameters on the network sensitivity. It is also used to design the network architecture and provide useful guideline for the network training.

References

[Hoff, 1962] M. E. Hoff, Learning phenomena in network of adaptive switching circuits. Stanford Electron. Labs., Stanford, CA, Tech. Rept. 1554-1, July 1962.

[Glanz, 1965] F. H. Glanz. Statistical extrapolation in certain adaptive pattern recognition systems. Stanford Electron. Labs., Stanford, CA, Tech. Rept. 6767-1, May 1965.

[Winter, 1989] R. G. Winter. Madaline rule II: A new method for training network s of Adalines. Stanford University, Stanford, CA, Ph.D. dissertation, Jan. 1989.

[Stevenson, 1990a] M. Stevenson. Sensitivity of Madalines to weight errors. Stanford University, Stanford CA, Ph.D. dissertation, Dec. 1990.

[Stevenson *et al*., 1990b] M. Stevenson, R. Winter, and B. Widrow. Sensitivity of feedforward neural networks to weight errors. *IEEE Trans. Neural Networks*, 1(1):71-80, 1990.

[Cheng and Yeung, 1999] Antony Y. Cheng, D.S. Yeung/ Sensitivity Analysis of Neocognitron. *IEEE Trans. Syst., Man, And Cyber.*, 29(2):238-248, May 1999.

[Yeung and Wang, 1999] D.S. Yeung, X.Z. Wang, Initial analysis on sensitivity of multilayer perceptron. *IEEE SMC '99 Conference Proceedings*, 3:407 –411, 1999.

[Choi and Choi, 1992] J. Y. Choi and C. -H. Choi. Sensitivity analysis of multilayer perceptron with differentiable activation functions. *IEEE Trans. Neural Networks*, 3.101-107, Jan. 1992.

[Piché, 1992] S. Piché. Selection of weight accuracies for neural networks. Stanford University, Stanford, CA, Ph.D., dissertation, May 1992.

[Piché, 1995] Stephen W. Piché. The selection of weight accuracies for Madalines. IEEE Trans. Neural Networks, 6(2):432-445,1995