PERGAMON

Neural Networks 15 (2002) 1041–1058

Neural Networks

www.elsevier.com/locate/neunet

2002 Special Issue

A self-organising network that grows when required

Stephen Marsland[a,*], Jonathan Shapiro[b], Ulrich Nehmzow[c]

[a]*Division of Imaging Science and Biomedical Engineering, Stopford Building, University of Manchester, Oxford Road, Manchester M13 9PT, UK*
[b]*Department of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, UK*
[c]*Department of Computer Science, University of Essex, Colchester CO4 3SQ, UK*

Abstract

The ability to grow extra nodes is a potentially useful facility for a self-organising neural network. A network that can add nodes into its map space can approximate the input space more accurately, and often more parsimoniously, than a network with predefined structure and size, such as the Self-Organising Map. In addition, a growing network can deal with dynamic input distributions. Most of the growing networks that have been proposed in the literature add new nodes to support the node that has accumulated the highest error during previous iterations or to support topological structures. This usually means that new nodes are added only when the number of iterations is an integer multiple of some pre-defined constant, λ.

This paper suggests a way in which the learning algorithm can add nodes whenever the network in its current state does not sufficiently match the input. In this way the network grows very quickly when new data is presented, but stops growing once the network has matched the data. This is particularly important when we consider dynamic data sets, where the distribution of inputs can change to a new regime after some time.

We also demonstrate the preservation of neighbourhood relations in the data by the network. The new network is compared to an existing growing network, the Growing Neural Gas (GNG), on a artificial dataset, showing how the network deals with a change in input distribution after some time. Finally, the new network is applied to several novelty detection tasks and is compared with both the GNG and an unsupervised form of the Reduced Coulomb Energy network on a robotic inspection task and with a Support Vector Machine on two benchmark novelty detection tasks. © 2002 Elsevier Science Ltd. All rights reserved.

Keywords: Unsupervised learning; Self-organisation; Growing networks; Topology preservation; Novelty detection; Dimensionality reduction; Mobile robotics; Inspection

1. Introduction

Unsupervised learning techniques generate mappings from input spaces of (usually) high dimension to lower-dimensional map fields, often 2D. An example of such an algorithm is the Self-Organising Map (SOM) of Kohonen (1982, 1993), where a lattice of connected nodes learn a representation of an input distribution. Networks such as the SOM learn by adapting the weight vectors linking input vectors from some input space to nodes in the map field. The weight vector of the node that best matches each input is moved closer to that input, as are nodes that are in the neighbourhood of the winner. In this way, perceptions that are 'close' in the input space are mapped to nodes that are near each other in the map.

The SOM and similar networks have two major limitations—firstly, the network structure and dimensionality must be decided prior to learning. This constrains the resulting mappings and the accuracy of the output. Secondly, the capacity of the network is predefined through the number of nodes in the network and the learning parameters. This makes the networks unsuitable for continuous learning or the learning of non-stationary datasets.

Growing networks are one way to work around these limitations of static networks. Many of the growing networks described in the literature (see Section 2) add either single nodes or whole layers of nodes into the network structure at the position where the accumulated error is highest, or to support topological structures. Nodes are only added when the number of learning iterations performed is an integer multiple of some pre-defined constant, λ, as the other iterations are needed to accumulate the error at each node. Once a node has been added, several more iterations of the learning algorithm are performed before another node is introduced. Thus, the network grows at the same rate no matter how the input distribution is changing. The network continues to grow until the algorithm is terminated, often by

* Corresponding author.
E-mail addresses: smarsland@cs.man.ac.uk (S. Marsland), jls@cs.man.ac.uk (J. Shapiro), udfn@essex.ac.uk (U. Nehmzow).

a pre-determined stopping criterion, such as a minimum error, being reached.

The network described in this paper uses a different criterion for when to add a new node, and how to initialise that node. Rather than adding a node to support the node with the highest error, instead, nodes are added whenever the current input is not matched by any of the current nodes to some (arbitrary) accuracy, with the new node being initialised to recognise the current input. This has the benefit that once the input space is matched within some defined error bound by the network nodes, the network will stop growing, but will start to grow again if the input distribution changes. We call the new network the *Grow When Required* (GWR) network because of the property that new nodes are added whenever the input is not sufficiently matched by the current network.

The GWR algorithm decides when to add nodes by evaluating the activity of the node that best matches the current input, with good matches being signified by high activity. It also uses information about how often that node has fired previously. While the network is learning it may be the case that the node that is the best match is still not actually a good match (so that the activity is low). If this node has not fired often it may be that the node is untrained, and just needs further training. However, if the node has often won, that is, been the node that best matches the input, then the activity should be high, as the training will ensure that the node matches the input well. If the activity of the node is low, then a new node is needed to match the current input, and so one is added.

In this paper we compare our new algorithm to one of the standard algorithms, the Growing Neural Gas (GNG) of Fritzke (1995). This network is described in more detail in Section 2. In particular, we compare the learning abilities of the networks for dynamic data sets. We first look at several artificial datasets where the input distribution changes over time, before applying the networks to a real problem, that of novelty detection; detecting novel stimuli from robot sonar readings. The results show that the GWR network performs well on all of these problems, but that the GNG is not suitable for novelty detection tasks because of the way in which nodes are added. For this reason we use another network that grows in use, but does not have neighbourhood connections, a modified form of the Reduced Coulomb Energy (RCE) network of Reilly, Cooper, and Erlbaum (1982). This network is also described in Section 2. Finally, we apply the GWR network to two benchmark novelty detection tasks and report comparison results with a support vector machine performing novelty detection.

It has been demonstrated that other growing neural networks are perfectly topology-preserving (in the sense of Bruske and Sommer (1995)), that is, the ordering of the nodes in the network reflects the ordering of the nodes in the input space. In general, this can only occur if the dimensionality of the map space is the same as that of the input space. We demonstrate that the GWR network is also perfectly topology-preserving.

This work is motivated by previous work on novelty detection (Marsland, Nehmzow, & Shapiro, 2000), where a SOM was used on-line to cluster sonar readings taken while a mobile robot explored. Habituation synapses connected each of the nodes of the map field to an output node. The synapses weakened when the node they were attached to was the winner, so that stimuli that were seen more frequently gave weaker signals to the output node. As the habituation dynamics only had a fixed point where the synapse was fully depleted, and the number of nodes in the network is predefined, the network was able to saturate, so that novel features were classified as previously seen. The network described in this paper is designed to rectify this fault.

2. Related work

The Cascade-Correlation Learning Architecture (CCLA) (Fahlman & Lebiere, 1990) is a supervised learning network that aims to substantially speed up learning. Like most of the growing networks described in this section, CCLA starts with a minimal network and adds units into the network architecture. However, unlike most of the networks described here, CCLA is a supervised network and nodes are added into the hidden layers of the network. The new units are intended to act as feature detectors and are added when no error reduction has occurred over several training iterations. The candidate unit is created and its input weights are trained to maximise the correlation between the output of the node and the residual output error before the node is added to the network. Once the node is added, these weights are frozen.

One of the first unsupervised growing neural networks was the Growing Cell Structure (GCS) network of Fritzke (1994). In the GCS, which is based on the SOM (Kohonen, 1982), the network is made out of k-dimensional simplices. The value of k is pre-defined, and is usually $k = 2$, so that the simplices are triangles. A new node is inserted every λ iterations, where λ is a constant, with the node positioned to support the node that has accumulated the highest error during previous steps. The network continues to adapt and grow until some stopping criterion is met. This can take the form of a prescribed network size, or some acceptable minimum for the accumulated error in the network. The GCS is one of a number of networks that are perfectly topology-preserving (Bruske & Sommer, 1995), in that if k is the dimension of the manifold on which the data lies, the positions of the map nodes conform to the topology of the input space.

A number of authors have proposed variations on the GCS. For instance, Burzevski and Mohan (1996) suggest a way of dealing with the changes in the network structure that deleting nodes can have on the GCSs network. The constraint of the network structure always being a collection of k-dimensional simplices can mean that deleting a node can cause massive upheaval in the network. This problem can be removed by using a hierarchy of GCSs arranged in a tree. The algorithm is extended to allow nodes to split, so that further branches can be formed. For particular data sets this is shown to produce superior results to the basic algorithm, but at the cost of many more nodes and a more complicated structure. Another network, the Dynamic Cell Structures, was proposed by Bruske and Sommer (1994, 1995). In this model, the addition of nodes into the network is aimed at maintaining the topology preserving structure of the network, rather than equalising the expected value of the error at each node, as in the GCS.

A different approach is the Probabilistic GCS of Vlassis (1997). This resets the GCS into a probabilistic framework, the principal advantage of which is to allow an improvement in the way in that the algorithm handles inputs which are correlated. Again, the work of Cheng and Zell (1999, 2000), has considered improvements to the basic structure of the GCSs algorithm. Their work allows for more than one node to be added at each insertion step. One, two or more neurons can be added at each insertion step, depending upon the topological structure of the network around the position of the winning unit. This aims to improve the rate of convergence of the network, although at the cost of using more nodes.

In addition to the GCS, Fritzke (1995) has also proposed the Growing Neural Gas (GNG). This network has a lot in common with the Neural Gas model of Martinetz, Berkovich, and Schulten (1993). As in the GCS, new nodes are added every λ iterations, to support the node with the highest accumulated error, but in the GNG the structure of the network is not constrained, with links being created between the two nodes with the highest activity for each input. For each data sample presented to the network, the two best-matching nodes are selected, that is the two nodes whose weights are closest to the input in the Euclidean sense. A neighbourhood connection is made between the two nodes if it does not already exist, and the positions of these nodes—together with the neighbours of the winning node—are moved so that their weights better match the input. Edges that are not used increase in age, while edges that are used have their age reset to zero. Once the age of an edge exceeds a threshold, that edge is deleted. After λ iterations, the node that has accumulated the highest error during the previous steps is calculated, and a new node is added to support it. The new node is positioned between the node with the highest error and whichever of its neighbours has the next highest error. The algorithm continues until some stopping criterion is reached. A number of works have compared the performance of the GCS and GNG algorithms for particular data sets. Examples are Fritzke (1996) and Kunze and Steffens (1995).

Fritzke (1997) has also investigated modifying the network so that it can be applied to non-stationary datasets, as we do in this paper. This is done by adding the concept of the 'utility' of a node, the amount that the global error would increase if that node were removed. Nodes with low utility are removed by the algorithm, which is known as the Growing Neural Gas with Utility (GNGU).

Other networks that have been proposed include that of Blackmore and Miikkulainen (1993), who proposed an incremental growing grid, where nodes are added to the perimeter of a grid that grows to cover the input space. Connections between adjacent nodes are created and destroyed according to the distance between their weight vectors. This ensures that the network is always a regular 2D structure, making it easy to view. Similar considerations drove the work of Seiffert and Michaelis (1997), except that they use 3D map fields.

An alternative approach is taken by Bauer and Villmann in the Growing Self-Organising Map (Bauer & Pawelzik, 1992; Bauer & Villmann, 1995; Villmann & Bauer, 1998). Their algorithm uses the learning rule of Kohonen's Self-Organising Map (Kohonen, 1982), but after each learning stage, again every λ iterations, neurons are added into the map space. Rather than adding one node at a time, and allowing connections between neurons to develop in the learning phase, entire rows of neurons are created simultaneously so that the rectilinear structure of the SOM map space is maintained. The algorithm chooses whether to add a new row or column, or to extend the map field into a new dimension, according to the appearance and dynamics of the Voronoi cells of the current network. The ability of the network to preserve neighbourhoods in the data, so that there is some topological consistency between the data in both the input and map spaces—a noted property of the SOM—is considered.

Another form of growing network grows new nodes without generating neighbourhood relations between the nodes, to perform categorisation. The simplest of these is the unsupervised RCE network (Kurz, 1996; Reilly et al., 1982), which uses prototype vectors to describe particular classes. If none of the current prototype vectors are sufficiently close to the current input, a new class is generated and the input used as the prototype for that cluster. There are no neighbourhood connections between clusters, nor can prototypes move once they have been placed. A more complex example of this type of network is the Adaptive Resonance Theory network (ART) (Carpenter & Grossberg, 1988). The ART network also adds new categories when a mismatch is found between the current input and the current set of categories, with the degree of mismatch allowed being controlled by a parameter known as the vigilance parameter.

An alternative approach is the Contextual Layered Associative Memory (CLAM) of Thacker and Mayhew (1990), which uses a multi-layered network that has feedback between the layers and resonance within a layer. Patterns are classified over a group of nodes rather than using a winner-takes-all approach. Nodes are added when the probability measure indicates that the region of input space that the current input comes from has low density.

Finally, an approach that does not consider self-organisation, but that has other similarities to the work considered here is that of Roberts and Tarassenko (1994), which is aimed at novelty detection, an application that is considered in Section 6. A Gaussian mixture model, a method of performing semi-parametric estimation of the probability density function is used to learn a model of 'normal' data from a training set. The number of mixtures is not defined in advance, with new mixtures being added if the mixture that best represents the data is further from the input than some threshold. In testing, any input that would require a new mixture to be generated is considered to be novel.

3. The Grow When Required network

This section describes the GWR network. Details of the algorithm are given in Section 3.1. The network has two important components—the nodes, with their associated weight vectors, and the edges that link the nodes to form neighbourhoods of nodes that represent similar perceptions. Both the nodes and edges can be created and destroyed during the learning process.

The technique used for creating and destroying network edges is the competitive Hebbian learning method used by Fritzke (1995) and Martinetz and Schulten (1991). For each input an edge connection is generated between the node that best matched the unit and the second-best matching unit. These edge connections have an associated 'age'. This is originally set to zero, and is incremented at each time step for each edge that is connected to the winning node. The only exception is the edge that links the best-matching and second best units, whose age is reset to zero. Edges whose age exceeds some constant $a_{\max}$ are removed. Any node that has no neighbours, i.e. that has no edge connections, is removed, as it is a dead node.

The new part of the algorithm is the way that the growing process is carried out. Rather than adding a new node after every λ inputs, as in the GNG network, new nodes can be added at any time. For example, several may be added one after another and then no more added for the next hundred iterations. The new nodes are positioned dependent on the input and the current winning node, rather than adding them where the accumulated error is highest, as in Fritzke's GNG algorithm.

A new node is added when the activity of the best-matching node (which is a function of the distance between the weights of the node and the input, see Eq. (6)) is not sufficiently high. The activity of nodes is calculated using the Euclidean distance between the weights for the node and the input. To allow for the fact that recently created nodes may not yet have been trained to match their intended output correctly, which would mean that the node should be trained more rather than a new node created, each node is equipped with a way of measuring how often the node has fired. This could be done in a variety of ways, the simplest of which is to use a simple counter for each node, which is incremented whenever that node is the best match.

An alternative to using the simple counter to record how often each node has fired is to have a variable that decreases exponentially from 1 to 0, so that new nodes have a value of 1 and nodes that have fired frequently are close to 0. This is equivalent to a counter with an upper limit, but has a few benefits. The fact that neighbours of the winning node are also trained can be acknowledged, as their variables can also decrease, although to a lesser extent. Also, the number of times that a node has fired can be very easily taken into account in the learning rate (see Eq. (11)), so that nodes that have fired frequently are trained less. This removes a problem that networks that learn continuously often suffer from, the weights of well-trained nodes continue to move slightly, so that the network does not converge. As with most self-organising networks, the setting of the learning rates is usually based on prior experimentation. Finally, it means that the GWR network can be used as a novelty filter without any modification, if the node that fires has not fired before, or fired very infrequently, then the input is novel (see Section 6). In animals, this decreasing response to a stimulus is known as habituation (Kohonen, 1993).

So, when an input is presented to the network, the activity of each node in the map space is calculated and a winner picked. If this best-matching node represents the input well then the activity of that node will be close to 1 (calculated using Eq. (6)). In that case the best-matching node is trained a little, as are its neighbours. However, if the activity of the network is below the insertion threshold a_T, then either the node has only recently been added to the map and is still being trained, or there is a mismatch between the node and the input. If the node is a new one then the firing counter for the node will be high, and so the node is trained a little and the counter decreases. Otherwise, a new node is needed to represent the input better. This node is added between the (badly matched) winning node, which caused the problem, and the input, with the weights of the new node being initialised to be the mean average of the weights for the best-matching node and the input. This method of node generation, and in particular the insertion threshold a_T, can be thought of as tunable generalisation; the amount to which the network generalises between similar perceptions is controlled by the amount of discrepancy between perceptions that triggers a new node.

In addition to the insertion threshold described previously, a threshold is also required to decide at what level of firing an input is considered to be sufficiently trained, so that low activity signifies a mismatch. In practise, the value of this threshold does not seem to affect the behaviour of the network significantly. Using the exponentially decreasing function suggested in Eq. (14), the threshold was set so that if a node had fired five times then it was considered to be trained.

The value of the insertion threshold a_T does make a large difference, however. If the value is set very close to 1 then more nodes are produced and the input is represented very well. For lower values of a_T fewer nodes are added. The effects of changing the parameter are investigated in Section 5.

3.1. The GWR algorithm

We now detail the precise steps of the algorithm. In order to facilitate comparisons, the new algorithm is described using the same notation as used in Fritzke (1995) to describe the GNG.

Let A be the set of map nodes, and $C \subset A \times A$ be the set of connections between nodes in the map field. Let the input distribution be $p(\boldsymbol{\xi})$, for inputs $\boldsymbol{\xi}$. Define $\mathbf{w}_n$ as the weight vector of node n.

Initialisation. Create two nodes for the set A

$$A = \{n_1, n_2\}, \tag{1}$$

with n_1, n_2 initialised randomly from $p(\boldsymbol{\xi})$. Define C, the connection set, to be the empty set

$$C = \emptyset. \tag{2}$$

Then, each iteration of the algorithm looks like this:

1. Generate a data sample $\boldsymbol{\xi}$ for input to the network.
2. For each node i in the network, calculate the distance from the input $\|\boldsymbol{\xi} - \mathbf{w_i}\|$.
3. Select the best matching node, and the second best, that is the nodes $s, t \in A$ such that

$$s = \arg\min_{n \in A} \|\boldsymbol{\xi} - \mathbf{w}_n\|. \tag{3}$$

and

$$t = \arg\min_{n \in A/\{s\}} \|\boldsymbol{\xi} - \mathbf{w}_n\|, \tag{4}$$

where $\mathbf{w}_n$ is the weight vector of node n.

4. If there is not a connection between s and t, create it

$$C = C \cup \{(s,t)\}, \tag{5}$$

otherwise, set the age of the connection to 0.

5. Calculate the activity of the best matching unit

$$a = \exp(-\|\boldsymbol{\xi} - \mathbf{w}_s\|). \tag{6}$$

6. If the activity $a <$ activity threshold a_T and firing counter $<$ firing threshold h_T then a new node should be added between the two best matching matching nodes (s and t)
   - Add the new node, r

$$A = A \cup \{r\}. \tag{7}$$

   - Create the new weight vector, setting the weights to be the average of the weights for the best matching node and the input vector

$$\mathbf{w}_r = (\mathbf{w}_s + \boldsymbol{\xi})/2. \tag{8}$$

   - Insert edges between r and s and between r and t

$$C = C \cup \{(r,s),(r,t)\}. \tag{9}$$

   - Remove the link between s and t

$$C = C/\{(s,t)\}. \tag{10}$$

7. If a new node is not added, adapt the positions of the winning node and its neighbours, i, that is the nodes to which it is connected

$$\Delta\mathbf{w}_s = \epsilon_b \times h_s \times (\boldsymbol{\xi} - \mathbf{w}_s) \tag{11}$$

$$\Delta\mathbf{w}_i = \epsilon_n \times h_i \times (\boldsymbol{\xi} - \mathbf{w}_i), \tag{12}$$

where $0 < \epsilon_n < \epsilon_b < 1$ and h_s is the value of the firing counter for node s.

8. Age edges with an end at s

$$\mathrm{age}_{(s,i)} = \mathrm{age}_{(s,i)} + 1. \tag{13}$$

9. Reduce the counter of how frequently the winning node s has fired according to

$$h_s(t) = h_0 - \frac{S(t)}{\alpha_b}\left(1 - \mathrm{e}^{(-\alpha_b t/\tau_b)}\right) \tag{14}$$

and the counters of its neighbours (i)

$$h_i(t) = h_0 - \frac{S(t)}{\alpha_n}\left(1 - \mathrm{e}^{(-\alpha_n t/\tau_n)}\right), \tag{15}$$

where $h_i(t)$ is the size of the firing variable for node i, h_0 the initial strength, and $S(t)$ is the stimulus strength, usually 1. α_n, α_b and τ_n, τ_b are constants controlling the behaviour of the curve. The firing counter of the winner reduces faster than those of its neighbours. The values used in the experiments were: $h_0 = 1$, $\alpha_b = 1.05$, $\alpha_n = 1.05$ and $\tau_b = 3.33, 14.3$. Eq. (14) is the solution to the differential equation

$$\tau_b \frac{\mathrm{d}h_s(t)}{\mathrm{d}t} = \alpha_b[h_0 - h_s(t)] - S(t), \tag{16}$$

which is the model of how the efficacy of an habituating synapse reduces over time proposed by Stanley (1976). This has been used in previous work on novelty detection (Marsland et al., 2000).

10. Check if there are any nodes or edges to delete, i.e. if there are any nodes that no longer have any neighbours, or edges that are older than the greatest allowed age, in which case, delete them.
11. If further inputs are available, return to step (1) unless some stopping criterion has been reached.

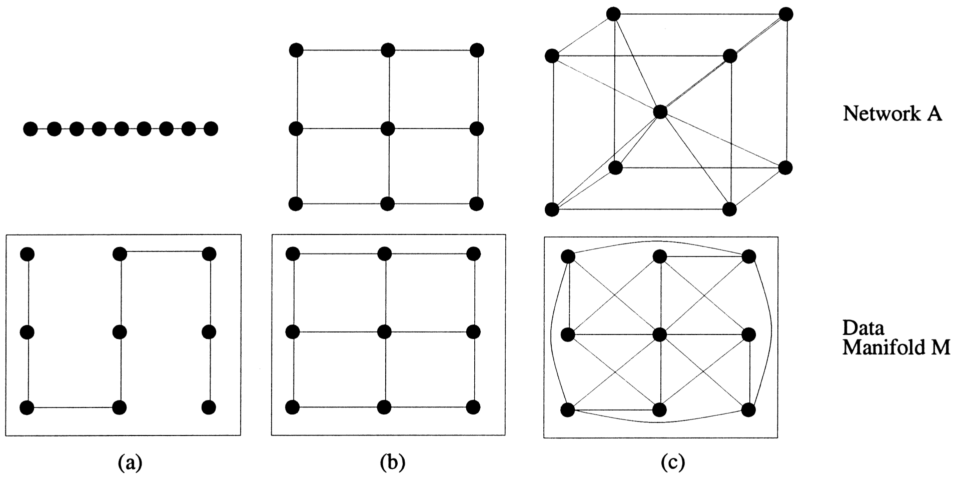

Fig. 1. Networks A of three different dimensionalities receive mappings from a square-shaped manifold M. Only in (b) where the dimensionality of the manifold M and map A are the same is a neighbourhood preserving map generated. Adapted from Martinetz and Schulten (1994).

4. Measures of network performance

This section discusses the properties of the self-organisation and learning processes taking place as the GWR network learns. Section 4.1 introduces the notation that is used in the discussion, neighbourhood preservation is discussed in Section 4.2, and three different measures that have been proposed in the literature to measure it are described. However, neighbourhood preservation is not the only way to evaluate the performance of the mapping learnt by a network. The accuracy with which the network models the data and the length of the neighbourhood connections are also possible measurements. Measures of these properties are described in Section 4.3. These different measures are evaluated when the GWR network learns representations of a number of different datasets in Section 5.

4.1. Notation

This section introduces the notation that is generally used in the literature on self-organisation and topology preservation.

A network A comprises N nodes and receives inputs sampled from a data manifold $M \subset \mathbb{R}^d$. Every node i in A has a synaptic weight vector $w_i \in \mathbb{R}^d$. The representation of M formed in A is defined by the mappings $M_A = (\Psi_{A\to M}, \Psi_{M\to A})$, the mapping from M to A and its inverse, which are defined by:

$$M_A = \begin{cases} \Psi_{M\to A} : M \to A; v \in M \mapsto i^*(v) \in A \\ \Psi_{A\to M} : A \to M; i \in A \mapsto w_i \in M \end{cases}. \quad (17)$$

where $i^*(v)$ is the map unit with weight vector $w_{i^*(v)}$ closest to v. A connection matrix C is defined on the network A by putting non-zero matrix entries between nodes that are connected in the network, i.e.

$$C_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are connected} \\ 0 & \text{otherwise} \end{cases}. \quad (18)$$

4.2. Measuring neighbourhood preservation

The preservation of neighbourhood relations (also known as topology preservation) is a very useful property of self-organising networks and has attracted a great deal of interest. Particularly useful papers are Goodhill and Sejnowski (1997), Martinetz and Schulten (1994) and Villmann, Der, Herrmann, and Martinetz (1997).

Loosely speaking, a mapping preserves neighbourhood relations if nearby points in input space remain close in the map space. This has been formalised by Martinetz (1993), through the definition of the perfectly topology-preserving map. A mapping between input manifold and network is perfectly topology preserving if and only if connected nodes i,j that are adjacent in A have weight vectors w_i, w_j adjacent in M.

In general, a network can only perform a perfectly topology-preserving mapping if the dimensionality of the map space reflects the dimensionality (or at least, the *intrinsic dimensionality*) of the input space. This is demonstrated in Fig. 1. At the bottom of the figure, three different square-shaped manifolds M are shown. Only in Fig. 1(b), where the dimensionality of the map space and input space are the same, can a perfectly topology-preserving map be generated between M and A. For this reason, for a network to be perfectly topology preserving it is necessary for the network to evolve to reflect the dimensionality of the dataset, or have this dimensionality preset.

The question of how topology preservation can be measured has received a lot of attention. Several authors have described ways of quantifying neighbourhood preservation. These can be split into two categories—measures of similarity and measures of similarity ordering. In the first class are those measures that evaluate the similarity of pairs of points before and after the neighbourhood mapping, and require that the two similarity measures are at least correlated, while in the second class it is only required that the relative ordering of the similarities is preserved. A useful review is given by Goodhill and Sejnowski (1997). Two interesting measures are the C measure (Goodhill & Sejnowski, 1997), which requires that symmetric similarity measures are defined both ways between the input space M and the map space A, usually as the Euclidean distances, and the topographic product P (Bauer & Pawelzik, 1992), which evaluates the neighbourhood preservation by computing the distance between neighbours in both the map space and the input space. However, the most useful measure that has been proposed, because it can deal with non-linear data manifolds, is the topographic function, which is described next. This measure is used in Section 5, as are some measures that are proposed for the new network, which are described in Section 4.3.

4.2.1. The topographic function Φ_A^M (Villmann et al., 1997)

The topographic product is limited to linear data manifolds, as the neighbourhood relations are measured using the Euclidean metric within the embedding space of the weight vectors. A way around this problem is proposed by the topographic function, which evaluates the topology preservation of the SOM mapping taking the structure of the data manifold into account using the Delaunay triangulation induced onto it by the mapping. The neighbourhood preservation of the mappings $\Psi_{M\to A}$ and $\Psi_{A\to M}$ are denoted by $f_j(k)$ and $f_j(-k)$, respectively, with j being the index of the node in the map and $k = 1, \dots, N-1$. The topographic function Φ_A^m of map M_A is then defined by

$$\Phi_A^M(k) = \begin{cases} \dfrac{1}{N}\displaystyle\sum_{j\in A} f_j(k) & k < 0 \\ \Phi_A^M(1) + \Phi_A^M(-1) & k = 0 \\ \dfrac{1}{N}\displaystyle\sum_{j\in A} f_j(k) & k > 0 \end{cases} \tag{19}$$

$\Phi_A^M(0) = 0$ if and only if the map is perfectly topology-preserving.

The question is then how to compute the neighbourhood preservation functions $f_j(k)$ and $f_j(-k)$. The basic approach is to use the induced Delaunay triangulation, that is, the graph connecting points with adjacent Voronoi polyhedra. In the form given by Martinetz and Schulten (1994) the topographic function is only specified for rectangular lattices. This means that they are not applicable to the GWR network and other networks where the network structure is not of this form. For the topographic function the problem is in the measurements of $f_i(k)$ and $f_i(-k)$. A description of how these measurements can be made more general is given by Villmann et al. (1997), and is developed further here.

The structure of A is defined by the connectivity graph C that is generated by the competitive Hebbian rule (see Eq. (18)). A discrete topology (Berge, 1997) can be induced on this space using the graph metric in $C(i)$, where $C(i)$ is C with node i taken as the root. A second discrete topology can be induced on A by considering the graph metric of the Delaunay graph, $D(i)$, again with node i taken as the root. These two topologies are referred to as $T_A^-(i)$ and $T_A^+(i)$, respectively.

A neighbourhood topology also needs to be induced on the data manifold M, or at least that subset of it $M^A = \{w_i \in \mathbb{R}^d | i \in A\}$. By generating the Voronoi diagram of M using M^A and constructing the dual Delaunay graph of this D, the required topology (labelled T_{M^A}) is induced.

Thus, three discrete topological spaces have been created, two on $A - (A, T_A^-(i))$ and $(A, T_A^+(i))$, and one on M^A, $(M^A, T_{M^A}(i))$. The map $M_A = (\Psi_{A\to M}, \Psi_{M\to A})$ can then be defined as topology preserving if the maps $\Psi_{M\to A}$ and $\Psi_{A\to M}$ are both continuous mappings for all nodes $i \in A$, on their respective topological spaces

$$\Psi_{M\to A} : \left(M^A, T_{M^A}(i)\right) \to (A, T_A^-(i)) \tag{20}$$

$$\Psi_{A\to M} : (A, T_A^+(i)) \to \left(M^A, T_{M^A}(i)\right). \tag{21}$$

Using these relations, general forms for $f_i(k)$ (which measures the continuity and hence the neighbourhood preservation of $\Psi_{M\to A}$) and $f_i(-k)$ (which does the same for $\Psi_{A\to M}$) can be derived, and are given below:

$$f_i(k) = \#\left\{ j | d_{T_A^-(i)}(i,j) > k; d_{T_{M^A}(i)}(i,j) = 1 \right\} \tag{22}$$

$$f_i(-k) = \#\left\{ j | d_{T_A^+(i)}(i,j) = 1; d_{T_{M^A}(i)}(i,j) > K \right\}, \tag{23}$$

where $\#\{\cdot\}$ is the cardinality of the set, $k = 1, \dots, N-1$ and $d_{T(i)}(i,j)$ is the distance metric based on each of the topologies. These measurements can then be used in Eq. (19) and the topology preservation of the mappings learnt by the GWR network measured.

Evaluating the topographic function is a computationally expensive task, as the Delaunay triangulation has to be computed and then the graph of connections between datapoints has to be created and searched. The topological function has been evaluated for some of the datasets that are described in Section 5, and the results are given there.

4.3. Further performance measurements

The measurements described in the previous section aimed to evaluate the topology preservation of self-organising networks. This is only one of the properties that a network mapping should display. In this section two complementary cost measures are described that aim to evaluate the mapping between input space and map space generated by the algorithm. Any number of cost measures that evaluate the desired properties can be generated, these particular ones were chosen for their simplicity.

The two measures represent a trade-off between a complex network with many nodes that represents all possible inputs very accurately and an efficient network that generalises well.

The network should be as parsimonious as possible, meaning that the length of edges should be short and the number of nodes small, but equally the network of nodes much accurately model the data, so that the distance between an input and the node that best represents it should be small. The cost measures evaluate each of these aims separately. The first measure, E_1, given in Eq. (24), penalises the network for neighbourhood connections between nodes that are placed far apart on the graph:

$$E_1 = \sum_i \sum_{j<i} C_{ij} G(i,j), \tag{24}$$

where the sum is over all the nodes in the network, and C is the connection matrix defined in Eq. (18). For the implementation used in Section 5, $G(i,j) = \|w_i - w_j\|^2$, the Euclidean distance. The second measure, E_2, given by Eq. (25), shows how the network aims to minimise the distance between each data point d_μ and the node that best represents it, w:

$$E_2 = \sum_\mu \sum_i G(d,w) \cdot f_i(d_\mu, \{w\}), \tag{25}$$

where the first sum is over each element of the dataset, and

$$f_i(d, \{w\}) = \frac{e^{-\nu G(d_\mu, w_i)}}{\sum_j e^{-\nu G(d, w_j)}}. \tag{26}$$

In the limit as $\nu \to \infty$, this reduces to winner-takes-all, which is the implementation that is used in the results reported in the next section. With this winner-takes-all implementation, the measure would be optimised by k-means clustering (Luttrell, 1990).

Measure E_1 would be minimised if the network had no connections at all, while E_2 would be minimised if there was a node for every input pattern. It is in the simultaneous minimisation of the two measures that a good mapping is produced. As the number of nodes in the network is variable and the connectivity pattern is unconstrained, it is difficult, if not impossible, to find criterion functions that cannot be trivially optimised (that is, a network with a node on every input and no connections between nodes) for growing networks. In the absence of a probability model some heuristic is required to avoid this trivial solution.

5. Experimental results on simple data

This section demonstrates the performance of the network on a number of demonstration problems. The first is a very simple 2D artificial dataset designed to show how the algorithm learns. The next is an example used by Fritzke (1997) to demonstrate the GNGU. The GNGU network uses the idea of the utility of a node, being the amount that the error in the network would rise if that node were deleted, to decide whether a node was superfluous. This allows the network to track non-stationary datasets. We compare the GWR algorithm with the GNGU. For these two problems we evaluate the two cost functions described in Section 4, and also the topographic function. Then we demonstrate the topology preservation using the well-known two-spirals problem, and demonstrate the ability of the network to represent the dimensionality of the input distribution using a dataset where the input manifold varies in dimensionality.

In Section 6 we apply the GWR network to some real problems based on the task of novelty detection, first on a robotic inspection task and then on medical diagnosis and machine fault detection.

5.1. A very simple dataset

The first, simple, dataset is made by a data distribution where samples are drawn at random from the unit square, with inputs coming with non-zero probability from four squares within the input space and a straight line joining two of the squares. This can be seen in the top left picture of Fig. 2, which shows the behaviour of the network at intervals of 40 samples drawn from the input distribution. The positions of the weights vectors of the nodes are plotted, together with lines linking those nodes that have neighbourhood connections.

The figure shows that the network creates new nodes very rapidly initially—during the 40 data presentations 38 additional nodes are created, with three more made during the second 40 presentations, and only one more being created after this. Instead, the weights of the nodes are adjusted to form a better representation of the data, and extra edges produced by the growing process are pruned. After 240 samples the network has learned the dataset accurately, and the network does not change at all after that.

Fig. 2 also shows that the GWR network is topology preserving. In the squares, where the data is 2D, the network forms a 2D representation, whereas on the line that joins two of the squares, which is 1D, the neighbourhood structure of the network reflects this. This is shown again in Fig. 3, which shows the values of the cost measures described in Section 4 as the number of presentations of data to the network increases.

The first cost function measures the length of the connections between nodes. It can be seen that for both the GWR and GNG networks this value increases as nodes are added into the map field, but that it then decreases to an approximately constant value when the network stabilises.

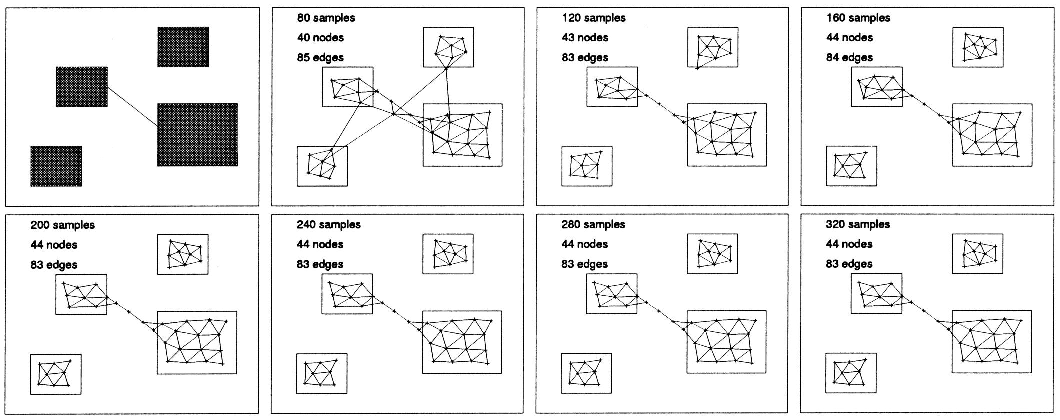

Fig. 2. The GWR Network learns a dataset consisting of four squares and one line. The top left diagram shows the layout of the dataset, the grey squares being the areas from which data can be sampled with non-zero probability. The other diagrams show the positioning of nodes and edges while the network learns. $a_T = 0.99$. Initially the network consists of two nodes places at random within the space. It can be seen that where the input is 2D, the structure of the network is 2D, and where the input is 1D, the network reflects this.

The second cost measure, which evaluates the distance between each datapoint and the corresponding node, shows similar results—the mapping decreases very rapidly to a constant level. This level is obviously controlled by the insertion threshold a_T. For the GWR network the second error function is fixed after 80 data presentations because no new nodes are added to the map and the nodes do not move. For the GNG network the value continues to decrease, because a new node is added at regular intervals. At the point where the GNG and GWR networks have the same number of nodes the cost function is lower for the GWR network. The topographic function for the GWR network is $\Phi(0) = 0.0023$.

The network shown in Fig. 2 was computed used a value of the insertion threshold $a_T = 0.99$. Table 1 shows the effect of this parameter on the number of nodes that the network produces, and the number of samples that were presented before the network stopped adding nodes. It can be seen that only for very high values of the parameter does the network take a large number of samples to stabilise.

5.2. A non-stationary dataset

Fritzke (1997) demonstrated the GNG with Utility using a dataset where the distribution changed rapidly while the network was learning. He showed that the GNG without utility failed to track the change in the dataset, leaving dead nodes behind, but that the GNGU successfully tracked the change in distribution.

Initially the inputs come from two squares, located in the top-left and bottom-right corners of the unit square, as can be seen on the left of Fig. 4. At some later time the distribution changes so that inputs are now sampled from the top-right and bottom-left corners of the square, as can be seen in the third and fourth pictures of Fig. 4.

Figs. 4 and 5 show how the GNGU network and GWR network learn a representation of this dataset. Each epoch, or iteration through the dataset, consists of 200 inputs sampled from the data distribution. At the end of each epoch the GNGU network added a new node in the usual way. The distribution was changed after 40 epochs (8000 samples), as can be seen in the figures. As Fritzke did, we constrained the

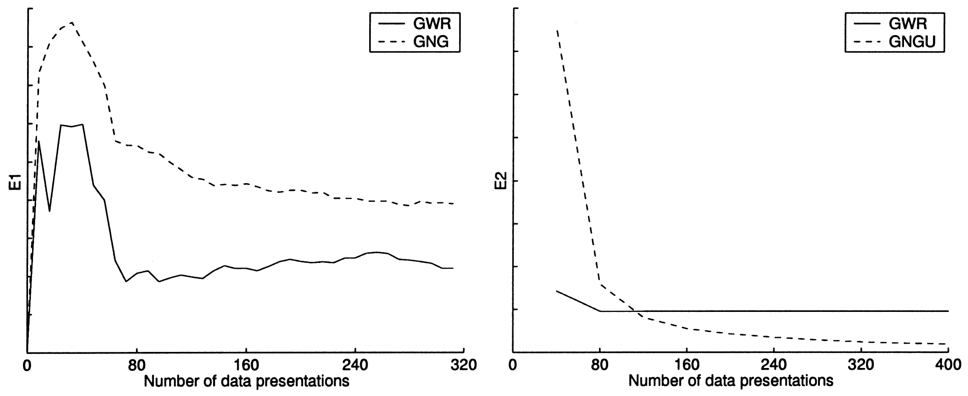

Fig. 3. Evaluation of the two cost measures described in Section 4.3 against the number of data samples for the GNG and GWR networks while the networks learn the four squares dataset. The GWR network performs better on the E_1 measure, concerned with the length of neighbourhood connections, but appears to perform less well on E_2, which measures the accuracy of the mapping. This is because the GNG network continues to add nodes. At the point where the two networks have the same number of nodes the GWR network has a lower value of E_2.

Table 1
The number of nodes produced for the squares problem, together with the number of data presentations required for the network to stop growing for different values of the activity threshold a_T

a_T	Number of nodes created	Number of data presentations to stop growing
0.8	6	2
0.85	6	2
0.9	6	5
0.95	10	19
0.99	42	55
0.999	204	563

size of the networks to be no bigger than 30 nodes, although for the value of the insertion threshold used here ($a_T = 0.95$) the GWR network did not reach this size. The values of the different parameters were set to be the same for both networks using the values suggested in Fritzke (1997). In particular, the learning rates were $\epsilon_b = 0.05$ for the winning node and $\epsilon_n = 0.0006$ for its neighbours.

Comparing Figs. 4 and 5 shows that the GWR network grows a simple solution very rapidly, while the GNGU is still changing the position of the nodes to produce a sensible ordering. When the distribution changes, which occurred immediately after the second picture in each of the figures, the GWR network very rapidly repositions the nodes as well as growing some additional nodes. This means that after another 4000 data samples the network already represents the data fairly well. The GNGU on the other hand, requires further training to delete a few superfluous nodes. Fig. 6 shows the evaluation of the two cost measures on the performance of both networks. Again, the two networks both perform well. In both cost measures it can be seen that the cost increases when the change in the network distribution is made. This is to be expected, as at that point the network is completely wrong about the input distribution. However, both networks recover fairly quickly and return to something like the same level as for the original data distribution. For the GWR network $\Phi(0) = 0.001$.

5.3. The two-spirals dataset

The two-spirals dataset is a benchmark dataset available in the CMU repository. Originally intended for supervised learning, the dataset comprises two interleaved spirals, as is shown in Fig. 7. Bruske and Sommer (1995) applied their Dynamic Cell Structures algorithm (DCS) to the dataset as an unsupervised problem, in order to demonstrate the topology-preserving nature of their algorithm, described in Section 2. They report that after 196 runs through the dataset (which contains 962 examples from the two spirals), their algorithm has learnt a perfectly topology-preserving map of the dataset, with the two spirals clearly separated. At this point the network has 198 nodes, since a new node is added at the end of every run through the training set. We applied the GWR network to the same problem, the results of which can be seen in Fig. 8 ($a_T = 0.8$, $\epsilon_b = 0.1$, $\epsilon_n = 0.01$). The topological function value at the end of training was $\Phi(0) = 0.0041$.

The DCS algorithm, like the GNG, grows slowly. Bruske and Sommer show that after 80 epochs the DCS has only just got the shape of the spirals. By way of contrast, after 80 epochs the GWR network has generated the perfectly topology-preserving map. As usual, the GWR network grows very quickly when trained on this dataset—all the nodes were generated by the sixth epoch, and the remainder of the time was spent removing neighbourhood connections that crossed between the spirals. It is interesting to note that the number of nodes generated by the GWR network is of the order of the number required by the DCS network.

5.4. A multi-dimensional dataset

The dataset shown in Fig. 9 was used by Martinetz and Schulten (1991) to demonstrate the Neural Gas network. It

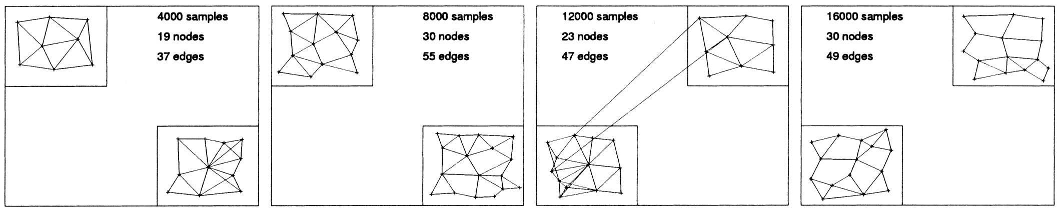

Fig. 4. The GNGU learns a dataset that changes suddenly after 8000 data samples (i.e. immediately after the second picture).

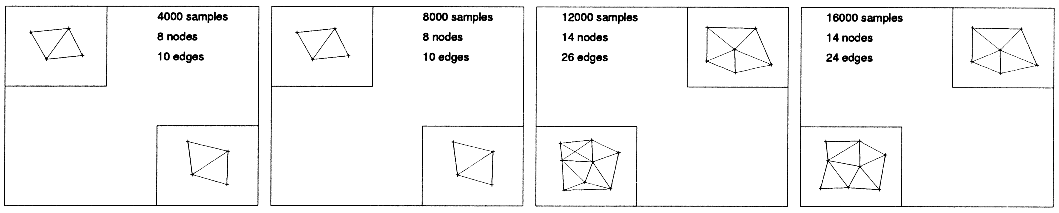

Fig. 5. The Network that Grows When Required learns a dataset that changes suddenly after 8000 data samples (i.e. immediately after the second picture).

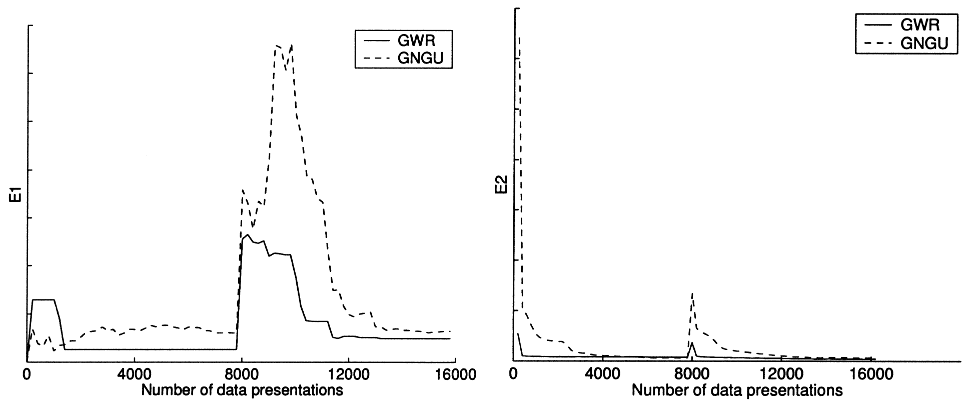

Fig. 6. Evaluation of the two cost measures described in Section 4.3 against the number of data samples for the GNGU and GWR networks while the networks learn the dataset that changes after 8000 data samples. For both measures the GWR network has lower costs.

consists of three parts, with three different dimensionalities, a parallelepiped, a rectangle and a circle with connecting line. It can be seen that the connections between the network nodes shadow the topological structure and dimensionality of the manifold on which the data lies. Datapoints sampled from this distribution were presented to the GWR network, with the parameters used being the same as were used in Fritzke (1995), $\epsilon_b = 0.2$, $\epsilon_n = 0.006$, $\alpha = 0.5$, $a_{\max} = 50$ and the insertion threshold, $a_T = 0.95$.

It can be seen that the network correctly selects the correct dimensionality for each part of the space, and rapidly learns a good representation. The network does not change substantially at any time after the third figure, when 15,000 data points have been presented.

6. Experimental results—novelty detection

The previous sections have demonstrated that the GWR network can learn to represent a simple dataset at least as well as the GNG, and that it can track a dataset that changes over time, responding faster than the GNGU network. In this section we apply the network to a real application, that of novelty detection, and provide comparisons with other networks.

The problem is one of novelty detection, that is, recognising that particular inputs to a network do not fit into the model that describes most of the inputs. Novelty detection is a very important task. For instance, it can be used to detect damage to machinery, or potential diseases from medical data. In addition, it can be used to focus the attention of a learning agent on potentially interesting stimuli, while ignoring others. Examples of all three applications are given in this section.

The GWR network performs novelty detection without any amendments, since the firing counter describes how often a node has fired before, and therefore how novel a stimulus that causes that node to fire is. Neither the GNG nor RCE networks do this.

In order to use GNG and RCE for novelty detection, a similar feature was added to them. A counter obeying Eq. (14) was added to each node of the networks. This counter did not affect the learning of the weights or structure; the standard algorithms were used for that. However, the novelty of a stimulus was recognised when the value of the counter for the firing node exceeded a threshold.

There are other possible ways of using these networks for novelty detection. For the GNG, one could use the error of the winning node to determine novelty; stimuli generating sufficiently large errors would be considered novel. Such an approach is obviously not possible for either the RCE or GWR networks, since in those networks new nodes are introduced to represent patterns with high error, so a novel pattern would immediately become highly non-novel. The GNG network, however, introduces nodes to support other nodes with high error rather than to represent patterns generating high error, so this approach seems feasible. Unfortunately, error is not a good measure of novelty in networks that grow, because the size of the error will naturally reduce as the network grows. The error threshold would have to change over time to make this work. Thus, even in the GNG network, use of error as a measure of

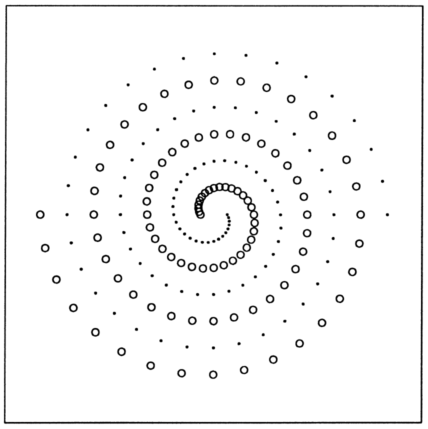

Fig. 7. The two-spirals dataset, which consists of data from two interleaved spirals.

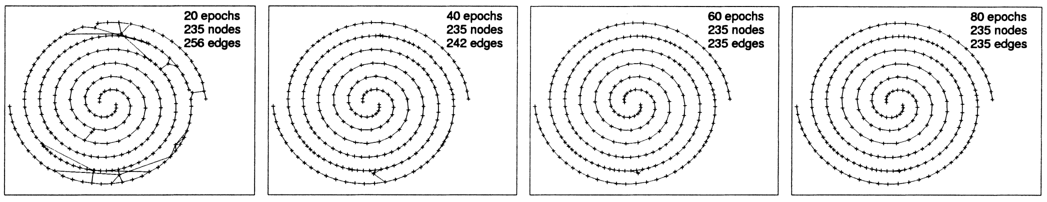

Fig. 8. The GWR network learns the two spirals problem. Note that the learning is unsupervised.

novelty is confounded with the measure that grows the network, albeit in a subtle way. For this reason, we used the same approach in the GNG network to recognise novelty as in the other networks.

6.1. Novelty detection in robot sonar scans

One application of a novelty filter is to recognise changes in a robot's perceptions of its environment. Possible applications of such a system include inspection—training the novelty filter to recognise all features that are known to be normal in a particular environment and then detecting deviations from the acquired model—and selecting places for the robot to explore. It is generally important that this can be performed in real time, since the detection of the novel feature could affect the behaviour of the robot. Further details are given in Marsland (2001). In the experiments described here a robot used its sonar sensors to travel through a short section of corridor sampling the environment with its sonar sensors and using these as inputs to the GWR network.

6.1.1. Experimental procedure

The original experiments were performed using a novelty filter based on the SOM. The results are presented and discussed in Marsland (2001) and Marsland et al. (2000), which also describes the data collection process in detail. The data consists of a number of sonar scans collected by a mobile robot as it travels along a corridor. Each input vector is the average of sonar scans taken over 10 cm of travel by the robot, with the readings of 16 sonar sensors making up each scan. Data was collected in three different environments, each of them being 10 m of corridor in the Computer Science building at the University of Manchester. Environments one and three are shown in Fig. 10. The second environment is identical to the first except that a door on the right hand side of the robot (door D in Fig. 10) was opened, which changed the perceptions of the robot.

As for previous experiments, the same parameters were

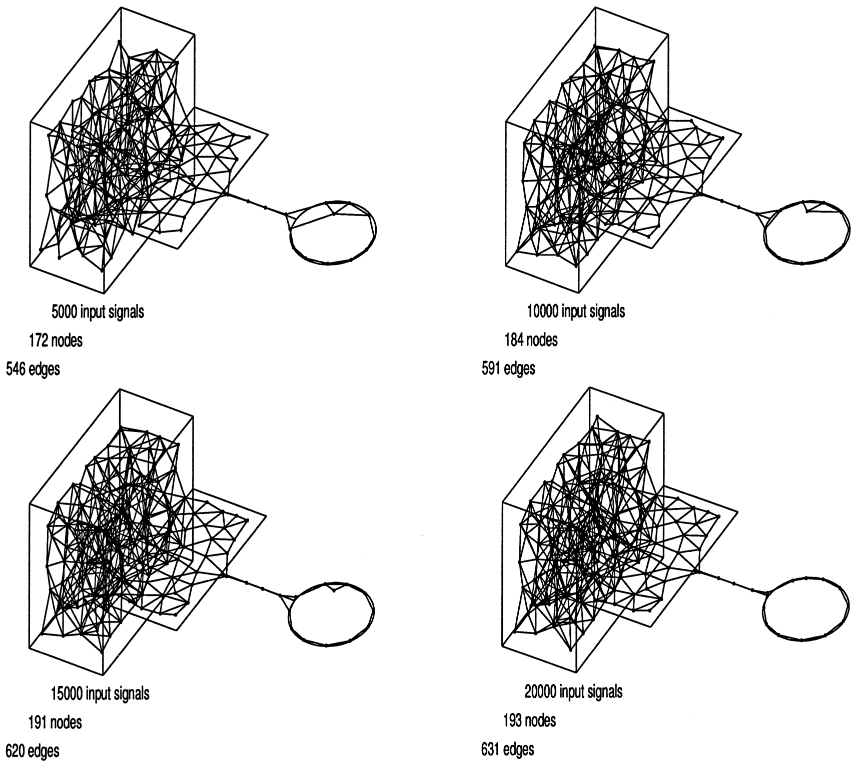

Fig. 9. The GWR network learns a representation of a signal distribution that has three different dimensionalities. It can be seen that the network has learnt a good representation after 15,000 data presentations, and does not change much after that.

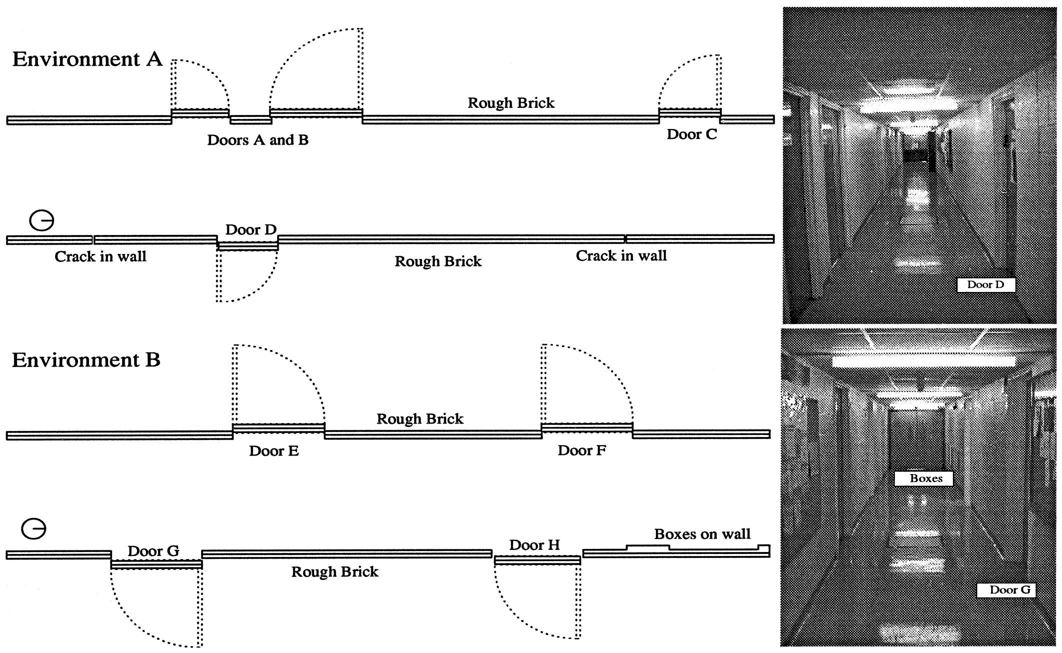

Fig. 10. The two environments used for data collection for the novelty detection problem. Data was also collected in environment A when the door to the right of the robot (door D) was open. This is called Environment A* in the discussion. The robot travelled along the corridor, storing a sonar scan of the environment every 10 cm.

used for both the GNG and GWR networks. These were: $\epsilon_b = 0.1$ for the winning node and $\epsilon_n = 0.001$ for the neighbours. The activation threshold for the GWR was $a_T = 0.9$, while for the GNG network a new node was added after every 50 data presentations, which meant that two nodes were added on each run in an environment. Tests were made to support the setting of this important parameter. For novelty detection, where the network should respond as soon as a novel input is seen, it seems reasonable to add a new node every iteration (i.e. $\lambda = 1$). This means that a new node is added every iteration, so that the network would represent the data with zero quantisation error, but would not discover any topological structure, would not recognise cluster, nor do novelty detection.

The insertion threshold for the unsupervised RCE network, which defines the radius of the hypersphere around each prototype within which a vector is classified as belonging to that class, was set to be 0.05. The RCE network does not have a learning rate as the prototype vectors do not move. If the current input is not matched by any of the classes then the input is used as a prototype vector for the new class that is created. The RCE network has no concept of neighbours, just a set of prototypes, with each prototype representing a class.

The experiments demonstrate that the network performing novelty detection can learn a representation of one environment and can then detect deviations from that representation while exploring in another environment. The same experiments were repeated three times, first using the novelty filter based on the GNG network, then using the novelty filter with a GWR network, and finally the novelty filter based on the RCE network was used. In the first two cases, the network was initialised with two randomly positioned nodes. The RCE network starts off without any nodes in the map. Then, three runs were made along the 10 m of environment A (shown at the top of Fig. 10), with the network generated at the end of one run saved and reloaded at the beginning of the next run. For each of the three algorithms this generated a network that could be considered to have learned about environment A. The evaluation of the novelty at each point in the environments for each of the three algorithms is shown in Fig. 11. The higher the spike, the more novel the perception at that point is considered to be.

This trained network was then used while the robot explored the same environment, but with the door to the right of the robot (door D in Fig. 10) opened, so that the sonar perceptions at that point were very different. The results of this are shown in Fig. 12. Finally, the networks trained in environment A were applied in a second environment, labelled environment B in Fig. 10. This is a very similar environment in the same part of the building, and the results can be seen in Fig. 13.

6.1.2. Discussion of the results

The results, which are given in Figs. 11–13 show how each network evaluated the novelty of the current perception with respect to previous perceptions of the robot. In the graphs, a high spike means that the perception was very novel, and short spikes means that the perception has been seen often.

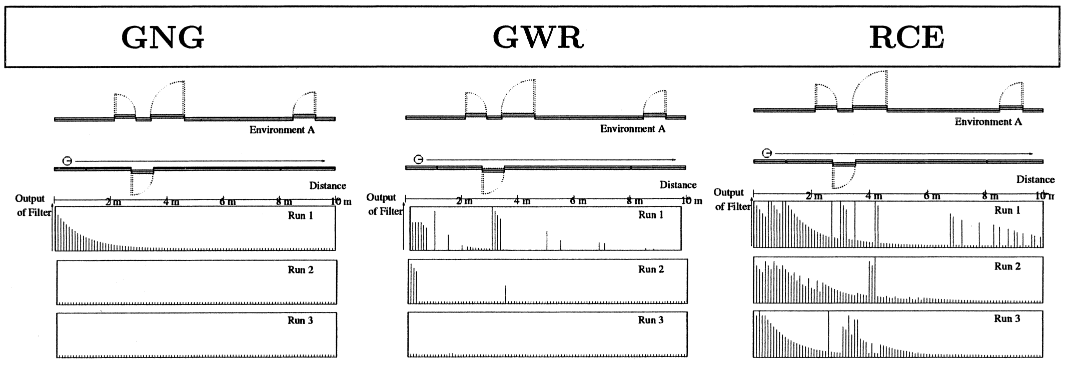

Fig. 11. The GNG, GWR and RCE networks (left to right) learning about environment A. The networks had no initial training. The positions of the spikes are lined up against the figure of the environment.

A comparison between the GNG, GWR and RCE networks for initial training is shown in Fig. 11. Here the networks were randomly initialised, so it would be expected that the perceptions will be novel initially and then learnt fairly fast, since much of the environment is wall. The perceptions that are different, such as the doorway, should produce some novelty at these points. This is what happens for the GWR network, shown in the centre of the figure. However, the GNG network cannot do this as there are only two neurons in the network until after the doorway is perceived, so the network cannot show this novelty. The RCE network (on the right of the figure) finds many things novel. Indeed, this network take a very long time to train. The insertion threshold for the RCE network is difficult to set because it controls the behaviour of the network very strongly. The value of 0.05 used here was chosen so that the network did find the perceptions of the doorway to be novel. However, for this sensitivity the network does not appear to be robust in the face of noise, especially not at the start of each run where the robot may be positioned slightly closer to or further away from the wall than in the previous run.

Fig. 12 compares the results for the networks when the inputs are the perceptions of the robot in environment A, but with the door to the right of the robot open after training with the door closed. For the network output to be considered useful, the network would have to highlight the area of the door as novel, but not any other part of the corridor, which is otherwise identical. This is the case for the GWR network, but the constraint of only adding neurons at predefined intervals limits the performance of the GNG network, which instead highlights novelty when a new node is added, which is after the novel perceptions are actually seen. The burst of novelty seen in the second run by the GWR network is the result of a very thin crack in the wall in environment A. Mostly the sonar sensors do not detect this crack, but it appears that they did in this case. The RCE network also detects the open doorway as novel. However, it also finds the early perceptions of the robot to be novel even though they are identical to those seen before. This is probably because the network is very sensitive to noise and the distance of the robot from the wall varies between runs.

When the network trained in the first environment is moved into the new, but similar, environment B, it could be expected that no novelty should be seen as the networks have already seen all possible perceptions. However, the experiment reported in Marsland et al. (2000) for the SOM-based novelty filter showed that some parts of this environment were considered novel, because the doorways are more deeply inset than in environment A. The further burst of novelty at the end of environment B is caused by the boxes that project from the wall at that point.

The middle of Fig. 13, which shows the results for the GWR network, provides evidence that this is what happens, and the network finds the doorways novel, but

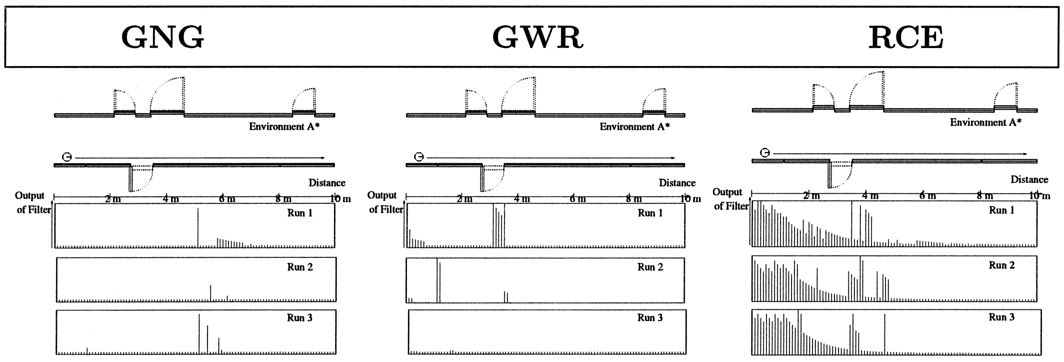

Fig. 12. The GNG, GWR and RCE networks (left to right) learning about environment A after the door has been opened, and after initial training in environment A with the door closed.

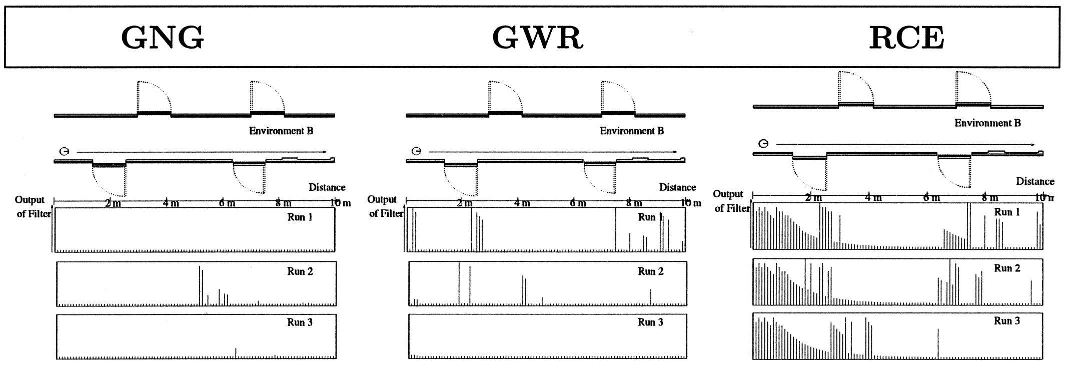

Fig. 13. The GNG, GWR and RCE networks (left to right) learning about environment B after initial training in environment A.

not the other parts of the environment. The RCE network also finds these perceptions novel, although again it has difficulty with the perceptions of the wall early on in each run. However, the GNG network (on the left of Fig. 13) again fails to detect novelty in those places where the perceptions are actually novel, because there are no spare neurons for this task.

Fig. 14 shows how the average output of the GWR novelty filter decreases as the robot makes several passes through an environment. It can be seen that after learning in environment A, environment A* (environment A, but with door D open) does not have very much novelty, environment B has more, while a completely different environment (environment C on the right of the figure, which is a wider corridor in a different part of the building) has as much as environment A did initially. However, in all cases the filter learns quickly and after five runs no novelty is found in any of the environments.

Of the three networks tested, only the GWR network has a steady progression between a stimulus being novel and not novel over several runs in an environment. This is largely a function of the neighbourhood connections, which means that nodes that recognise similar features are neighbours, so that the firing of one node means that the firing counter of the other nodes also reduces a little. The RCE network does not have these neighbourhood connections, which, coupled with the fact that the prototype vectors do not move after they have been placed, makes the network susceptible to noise.

6.2. Two other datasets

In this section the GWR network is tested on two datasets that are publicly available via the internet. The datasets were used by Campbell and Bennett (2000) to test their novelty detection technique that is based on modelling the support of a data distribution using a Support Vector Machine (SVM). This network uses a soft margin classifier with a Gaussian kernel (equivalent to an RBF network)

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathrm{e}^{-|\mathbf{x}_i - \mathbf{x}_j|/2\sigma^2} \tag{27}$$

to generate a hypersphere in feature space that contains most of the data. The novelty detection is based on the idea of Tax and Duin (1999).

The datasets are described below. They demonstrate the two areas where novelty detection is typically used—medical diagnosis and fault detection. These application areas are suitable because in both cases there are typically many examples of healthy data, patients whose tests do not highlight illnesses and machines working normally, and relatively few of problems. In addition, it is not known for definite that all possible manifestations of unhealthy data are known and the main purpose of the data is to avoid false negatives—not detecting possible problems. These are the circumstances under which novelty detection is the method of choice.

The following sections describe the datasets and the approach to training and testing that is used. The results achieved by the GWR network are given, as are those of the SVM novelty filter.

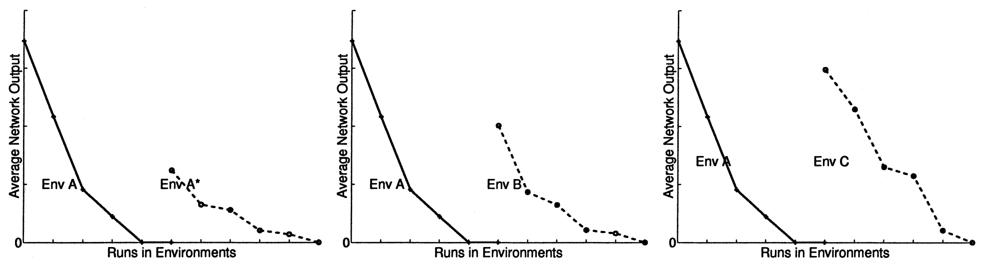

Fig. 14. Graphs showing how the total amount of novelty in an environment decreases as the GWR-based novelty filter learns over three runs in each of a pair of environments. Environment A* is the same as environment A, but with a change made, environment B is similar, and environment C is very different.

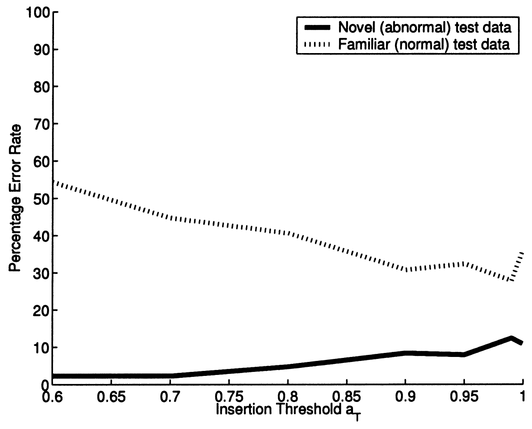

Fig. 15. The error rate against insertion threshold for normal ball-bearing test data (solid line) again abnormal test data (dotted line) with the GWR.

6.2.1. Dataset one: medical diagnosis

The first dataset is the *biomed* dataset available at http://lib.stat.cmu.edu/datasets (Cox, Johnson, & Kafadar, 1982). In total, 209 observations are given, of which 15 have one or more of the four attributes (measurements on blood samples) missing, and were therefore discarded. Of the remaining 194, 127 were normal, i.e., healthy data, and the remaining 67 contained one or more abnormalities, signifying the presence of a genetic disease.

Following the approach of Campbell and Bennett (2000), 100 randomly chosen normal observations were used as a training set and the remaining 27 normal observations together with the 67 inputs that should be detected as abnormal were used as a test set.

The results reported for the kernel classifier were very good- their best result was that 57 of the 67 abnormal inputs were correctly labelled and only two of the normal data were mislabelled as abnormal. The GWR network performed similarly. For a value of the insertion threshold of $a_T = 0.92$, 56 of the 67 abnormal inputs were highlighted correctly, and only two of the normal data were misclassified. This is very similar to that of the kernel machine. Unfortunately Campbell and Bennett (2000) do not report which datapoints were misclassified, and therefore it is not possible to see if the same points were found by both filters.

Campbell and Bennett (2000) show how varying the variance σ in the RBF kernel that was used (Eq. (27)) affects learning. For small σ, all the test data are labelled as abnormal, because a small Gaussian is centred on each training point, with each point requiring its own separate Gaussian cluster to represent it. This is what happens in the GWR network when the insertion threshold is high. As the insertion threshold decreases, or equivalently σ increases, generalisation gets better, but then abnormal inputs that are close to clusters can be missed. Therefore there is a trade-off between false positives and false negatives. As was discussed previously, false positives (that is, points incorrectly labelled as abnormal) are less of a problem, providing there are not too many of them, in which case the classifier is useless. However, the classifier should highlight all the abnormal inputs. Neither novelty filter manages to detect *all* the abnormal inputs on this particular dataset, although they both get fairly close.

6.2.2. Dataset two: fault detection

The second dataset is available at http://www.brunel.ac.uk/research/cnca/sida/html/data.htm as part of the EPSRC Structural Integrity and Damage Assessment (SIDA) network. The data consists of several sets of tests on ball-bearing cages. These are vital components of many machines, and therefore it is important to detect faults in them before use.

Every instance of data consisted of 2048 acceleration samples from a Bruel and Kjaer vibration analyser. A discrete Fast Fourier Transform was used to preprocess the data, resulting in 32 attributes for each input vector. The data was split into five types:

Type	Description	Number of instances
Normal	New ball-bearings	Two sets of 913
Fault 1	External ring completely broken	Two sets of 747
Fault 2	Basket damaged, one degree of freedom	Two sets of 996
Fault 3	Half of basket elements destroyed, four degrees of freedom	One set of 996
Fault 4	No visible damage, but runs loosely	One set of 996

The experiments described in Campbell and Bennett (2000) used the first set of data from each of the first three categories (normal, faults 1 and 2) to train the classifier, and then used the second set of data in each of these categories for testing. Finally the faults 3 and 4 sets were used to test the novelty detection performance. This tests more than just the novelty detection ability of the network, since it also tests the classification performance of data from the three categories that were used in training.

The results reported by Campbell and Bennett (2000) were more concerned with the correct classification of the test data from the first three categories, rather than the novelty detection performance. They report 1.3% error on the data from the normal data, 0% error on fault 1 and 46.7% error on fault 2, but only found 28.3% of the fault 3 data and 25.5% of the fault 4 data to be abnormal.

The results of using the GWR network are the opposite to those of the SVM approach. Rather than learning a good representation of the trained classes, but misclassifying many of the abnormal inputs as familiar, the GWR network performs very well at recognising abnormal inputs, but finds many of the test elements of the familiar classes to be abnormal too. Fig. 15 shows how the performance of the GWR network changes for familiar and abnormal test data as the insertion threshold a_T varies. It can be seen that as the insertion threshold gets closer to 1, so the categorisation performance of the familiar data gets better, but at the cost of missing some of the abnormal inputs.

When the insertion threshold is $a_T = 0.8$ the GWR network finds 95.4 and 95.1% of faults 3 and 4 data to be abnormal, but misclassifies as abnormal 37.8% of the good data, 40.3% of fault 1 and 43.8% of fault 2. Only for the fault 2 data are the SVM data similar. However, for novelty detection it is more important to find the abnormal entries—although the number of familiar perceptions misclassified as abnormal is high, this is less important.

7. Conclusions

This paper has presented an algorithm that describes a new type of self-organising growing neural network. The new network has similarities to growing networks such as those developed by Fritzke, such as the GNG, in that it maintains a set of neighbourhood connections between nodes that match similar perceptions. However, unlike those networks, the new network adds neurons whenever the current input is not matched sufficiently well by any of the current nodes. For this reason we call the algorithm the ‘Grow When Required (GWR)’ network.

This change in the way that nodes are added means that the behaviour of the network is very different to that of the GNG. In particular, the network responds very quickly to changes in the input distribution, adding lots of new nodes, and then stops adding them when each new input is already matched to the required accuracy. This means that the network is particularly suitable for learning about dynamic distributions, since each time the data distribution changes a new set of neurons will be grown to match the new data. A very simple example of this has been demonstrated in this paper, see Section 5.

We have also demonstrated that the GWR network is perfectly topology-preserving in the sense of Bruske and Sommer (1995). That is, the network preserves neighbourhood relations in the data so that inputs that are neighbours in the input space are mapped to neighbouring nodes in the map field. We described two cost measures that evaluate the performance of the network and used them to show how the network performed on a number of datasets.

The learning ability of the new network has been compared to two other growing networks, the GNG, which also maintains neighbourhood connections, but only adds nodes into the map space every time the number of iterations is an integer multiple of some predefined constant, and the unsupervised RCE network, which does not have any concept of neighbourhoods, but adds a new class whenever none of the classes previously defined match the current input to some specified accuracy. The GWR network also learns a representation of the input faster than the other networks.

The three networks were applied to a novelty detection task, where the network had to learn a representation of an environment, evaluating each perception online with respect to the representation as it was learned. The results show that the GNG network is not suitable for this type of task, as there are no spare nodes in the network at the time when the novel input is perceived. It was also found that the RCE network was very susceptible to noise, and that the parameter that controlled how similar inputs had to be to a prototype vector was difficult to set. The new GWR network performed better at this task than the other two, detecting those features that were novel without being too susceptible to noise. We then applied the network to two other novelty detection tasks, the first in medical diagnosis and the second in machine fault detection. The comparison in both of these cases was a Support Vector Machine-based novelty detector. The results showed that for these examples the GWR network was comparable for the first task and performed better at novelty detection (although worse at classification) than the Support Vector Machine for the second task.

Future work will examine the topology preservation capabilities of growing networks in more detail and consider the performance of the GWR network when used to learn about more complex dynamically changing datasets. The use of the GWR network for clustering will also be investigated.

Acknowledgments

The work described in the paper was performed as part of the first author’s PhD thesis in the Department of Computer Science at the University of Manchester. The work was supported by an EPSRC studentship.

References

Bauer, H.-U., & Pawelzik, K. (1992). Quantifying the neighbourhood preservation of self-organising maps. *IEEE Transactions on Neural Networks*, *3*(4), 570–579.

Bauer, H.-U., & Villmann, Th. (1995). *Growing a hypercubical output space in a self-organising feature map*. Technical Report TR-95-030, ICSI, Berkeley, July.

Berge, C. (1997). *Topological spaces: Including a treatment of multi-valued functions, vector spaces and convexity*. New York: Dover.

Blackmore, J., & Miikkulainen, R. (1993). Incremental grid growing: envoding high-dimensional structure into a two-dimensional feature map. *Proceedings of the IEEE International Conference on Neural Networks (ICNN’93)* (pp. 450–455).

Bruske, J., & Sommer, G. (1994). *Dynamic cell structures*. Advances in Neural Information Processing Systems (NIPS’94) (pp. 497–504).

Bruske, J., & Sommer, G. (1995). Dynamic cell structure learns perfectly topology preserving maps. *Neural Computation*, *7*, 845–865.

Burzevski, V., & Mohan, C. K. (1996). *Hierarchical growing cell structures*. Proceedings of the IEEE International Conference on Neural Networks (ICNN’96) (Vol. 3, pp. 1658–1663).

Campbell, C., & Bennett, K. P. (2000). A linear programming approach to novelty detection. In T. K. Leen, T. G. Diettrich, & V. Tresp (Eds.), *Proceedings of Advances in Neural Information Processing Systems 13 (NIPS’00)*, Cambridge, MA: MIT Press.

Carpenter, G. A., & Grossberg, S. (1988). The ART of adaptive pattern recognition by a self-organising neural network. *IEEE Computer*, *21*, 77–88.

Cheng, G., & Zell, A. (1999). Multiple growing cell structures. *Neural Network World*, *5*, 425–452.

Cheng, G., & Zell, A. (2000). Externally growing cell structures for pattern classification. *Proceedings of the Second International Symposium on Neural Computation (NC'2000)* (pp. 233–239).

Cox, L. H., Johnson, M. M., & Kafadar, K. (1982). *Exposition of statistical graphics technology*. ASA Proceedings of the Statistical Computation Section (pp. 55–56).

Fahlman, S. E., & Lebiere, C. (1990). The cascade-correlation learning architecture. In D. S. Touretzky (Ed.), (pp. 524–532). *Advances in Neural Information Processing Systems 2 (NIPS'90)*, San Mateo: Morgan Kaufmann.

Fritzke, B. (1994). Growing cell structures—A self-organizing network for unsupervised and supervised learning. *Neural Networks*, *7*(9), 1441–1460.

Fritzke, B. (1995). A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, & T. K. Leen (Eds.), (pp. 625–632). *Advances in Neural Information Processing Systems 7 (NIPS'94)*, Cambridge: MIT Press.

Fritzke, B. (1996). Growing self-organizing networks—Why? *ESANN'96: European Symposium on Artificial Neural Networks* (pp. 61–72).

Fritzke, B. (1997). *A self-organizing network that can follow non-stationary distributions*. Proceedings of the International Conference on Artificial Neural Networks (ICANN'97) Berlin: Springer (pp. 613–618).

Goodhill, G. J., & Sejnowski, T. J. (1997). A unifying objective function for topographic mappings. *Neural Computation*, *9*, 1291–1304.

Kohonen, T. (1982). Self-organised formation of topologically correct feature maps. *Biological Cybernetics*, *43*, 59–69.

Kohonen, T. (1993). *Self-organization and associative memory* (3rd ed). Berlin: Springer.

Kunze, M., & Steffens, J. (1995). Growing cell structure and neural gas: Incremental neural networks. *Proceedings of the Fourth AIHEP Workshop*.

Kurz, A. (1996). Constructing maps for mobile robot navigation based on ultrasonic range data. *IEEE Transactions on Systems, Man and Cybernetics—Part B: Cybernetics*, *26*(2), 233–242.

Luttrell, S. P. (1990). Derivation of a class of training algorithms. *IEEE Transactions on Neural Networks*, *1*(2), 229–232.

Marsland, S. (2001). *On-line novelty detection through self-organisation, with application to inspection robotics*. PhD thesis, Department of Computer Science, University of Manchester.

Marsland, S., Nehmzow, U., & Shapiro, J. (2000). Novelty detection on a mobile robot using habituation. *From Animals to Animats: Proceedings of the Sixth International Conference on Simulation of Adaptive Behaviour (SAB'00)* (pp. 189–198). MIT Press. URL ftp://ftp.cs.man.ac.uk/pub/robotics/sab2000habit.ps.Z.

Martinetz, T. (1993). Competitive Hebbian learning rule forms perfectly topology preserving maps. *Proceedings of the International Conference on Artificial Neural Networks (ICANN'93)* (pp. 426–438).

Martinetz, T. M., Berkovich, S. G., & Schulten, K. J. (1993). Neural-gas network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, *4*(4), 558–569.

Martinetz, T. M., & Schulten, K. J. (1991). A Neural-gas network learns topologies. In T. Kohonen, K. Mäkisara, O. Simula, & J. Kangas (Eds.), *Artificial neural networks* (pp. 397–402). Elsevier: Amsterdam.

Martinetz, T. M., & Schulten, K. J. (1994). Topology representing networks. *Neural Networks*, *7*(3), 505–522.

Reilly, D. L., Cooper, L. N., & Erlbaum, C. (1982). A neural model for category learning. *Biological Cybernetics*, *45*, 35–41.

Roberts, S., & Tarassenko, L. (1994). A probabilistic resource allocating network for novelty detection. *Neural Computation*, *6*, 270–284.

Seiffert, U., & Michaelis, B. (1997). Growing 3D-SOMs with 2D-input layer as a classification tool in a motion detection system. *International Journal of Neural Systems*, *8*(1), 81–89.

Stanley, J. C. (1976). Computer simulation of a model of habituation. *Nature*, *261*, 146–148.

Tax, D., & Duin, R. (1999). Data domain description by support vectors. In M. Verleysen (Ed.), (pp. 251–256). *Proceedings of European Symposium on Artificial Neural Networks (ESANN'99)*.

Thacker, N. A., & Mayhew, J. E. W. (1990). Designing a layered network for context sensitive pattern classification. *Neural Networks*, *3*(3), 291–299.

Villmann, Th., & Bauer, H.-U. (1998). Applications of the growing self-organising map. *Neurocomputing*, *21*, 91–100.

Villmann, T., Der, R., Herrmann, M., & Martinetz, T. M. (1997). Topology preservation in self-organising feature maps: Exact definition and measurement. *IEEE Transactions on Neural Networks*, *8*(2), 256–266.

Vlassis, N. A., Dimopoulos, A., & Papakonstantinou, G. (1997). The probabilistic growing cell structures algorithm. *Proceedings of the Seventh International Conference on Artificial Neural Networks (ICANN'97)* (645–654).