ICANN '93, Amsterdam
September 13-16 , 1993

SPIN
Learning and Forgetting Surface Classifications with Dynamic Neural Networks

Herman Keuchel, Ewald von Puttkamer & Uwe R. Zimmer

University of Kaiserslautern - Computer Science Department - Research Group Prof. E. v. Puttkamer
P.O. Box 3049 - W6750 Kaiserslautern - Germany
Phone: ...49 631 205 2624 - Fax: ...49 631 205 2803 - Telex: 4 5627 unikl d
e-mail: uzimmer@informatik.uni-kl.de

This paper refers to the problem of adaptability over an infinite period of time, regarding dynamic networks. A never ending flow of examples have to be clustered, based on a distance-measure. The developed model is based on the self-organizing feature maps of Kohonen [6], [7] and some adaptations by Fritzke [3]. The problem of dynamic surface classification is embedded in the SPIN project, where sub-symbolic abstractions, based on a 3-d scanned environment is being done.

1. Survey

First the framing project and the concrete problem context is discussed in short (chapter 2). Then in chapter 3 the network structure and associated aspects and problems are shown in detail, supported by simulation results (chapter 4).

2. SPIN-Project

At the actual state of research the project SPIN (from **S**patial **P**erception to **I**dentification with **N**eural networks) is based on the data of a 3-d scanning device and designed to reach a stage of abstraction where convex clusters of surfaces are generalized, completed and classified.

2-1. Main strategies

The system-design is based on some main principles, which have in common that none of them is in contradiction to a biological system. It is not intended to find the best fitting model for the lower levels of the mammal object-recognition-system, but obviously implausible features should be avoided.

a. Learning instead of preprogrammed models
The internal world model should be build up from a flow of examples scanned from the outer world. The "pre-programmed" knowledge is reduced to elementary features that should be searched for (here: edges).

b. Hierarchy
The main structure is pipeline-oriented instead of being controlled by a central instance (see the Neocognitron by Fukushima [5] for a good example of this strategy).

c. Symmetry
The general purpose processes (like classification or completion) should be quite similar at the different hierarchical levels.

d. Extensive use of feedback
A strict hierarchy is not as useful as it could be, if the different layers are not connected in both directions.

d-1 Error feedback
The back-propagated error messages correct decisions on lower processing stages, so it is not necessary to find always the best answer immediately. Lower stages may take the most likely way, knowing that there is another instance, which will give a negative feedback, if this decision was wrong.

d-2 Focus-of-interest feedback
At the start-up time of the whole system, lower components are triggered by local "instincts", whereas later on the activity of the lower components is more and more initiated by a focus-of-interest, generated by higher stages.

e. Parallelism
The above described strategies lead straight forward to forms of parallelism of rough granularity and by the use of neural networks at several stages to parallelism at a finer granularity.

2-2. Focus on surface-classification

The concrete problem area that will be discussed in this article, is defined now.

Surface representation

Given a laser-range scanner and several preprocessing steps, both beyond the focus of this paper, which produce a surface representation, built upon the curvatures at the borders of the surfaces. The curvatures are calculated by projecting the border (at each border point) on two orthogonal two-dimensional planes and then regarding the 2-d-curvatures on these planes (called the surface- and the border-curvature). The orientations of these planes are defined at each border point by the orientation of the local border tangent and the local surface normal: Both planes have to include the border tangent and one of them (the plane to determine the surface-curvature) has to include the surface normal (this has to be approximated or to be solved analytically). Each surface is then described by the concatenation of two vectors; one consisting of the surface-curvatures at m_1 equidistant points along a whole cycle along the border and another consisting of the border-curvatures at m_2 equidistant points along the same distance.

This may be regarded as just another surface vector-representation, but with the main aspect that the form of the surface is described by the curvatures at the border only, i.e. the characteristics e.g. at the middle of the surface are not detected at all. The idea beyond this representation is the assumption that surfaces in indoor environments may be sufficiently captured by the characteristics at their borders.

The task

Based on the representation described above, the continuous flow of scanned and pre-processed surfaces is to be clustered (or classified) based on the euclidian distance of the surface-vectors.

3. Network Model

The part of SPIN discussed in this paper requests a network model, with features listed below:

- Unsupervised clustering
- Dynamical number of clusters
- Forgetting by time or frequency of access
- Flexibility over an infinite period of time

There is only a small number of well known networks that might be used for such purposes (e.g. ART-models by Carpenter & Grossberg [2], Self-organizing feature-maps by Kohonen [6], [7], GAL by Alpaydin [1]). But all show limited abilities in at least one of the mentioned points.

3-1. Dynamic Network Model

The base of the following network-model is the self-organizing feature-map model by Kohonen [6]. This well known structure is extended by the possibility of adding and removing new cells. This work was being done by Fritzke in 1991 [3], [4]. Although this is already published, we will show the main aspects of this extension in short form before we discuss our adaptations.

Generalization & Learning

The representation of the surfaces as shown above is a vector in a m-dimensional real-valued vector-space. These vectors are mapped to an array S of cells c, each attached to an m-dimensional position vector $pos(c)$. The cells in S are connected in a triangular structure (the original Kohonen-model uses a rectangular structure). An input vector x is then mapped onto the cell with the smallest distance to it (in the Euclidian norm). This cell is called *bmu* (best matching unit) and this part is the classification.

$$\forall c \in S : \quad \| pos(bmu) - x \| \leq \| pos(c) - x \| \qquad (1)$$

For learning purposes the cell bmu and it´s topological neighbours in the triangular structure

are moved towards the input vector (by a fraction e_{bmu} and $e_{neighbour}$ of the distance). This training step is repeated n_d-times.

The described procedure results (if well defined) in a *topology-preserving* (i.e. adjacent input vectors are mapped on adjacent cells) and *distribution-preserving* map (i.e. the relative density of the cells approximates the probability density P(X) of the input vectors).

Growing cell-structures

Up to here, the number of cells in *S* is constant, i.e. the choice of the number of cells that should represent adequately the probability-distribution of the input vectors has to be known in advance (Additionally the initial position of the cells is critical regarding the ability and speed of learning). To overcome this restriction a mechanism of expanding the structure is introduced. The initial structure is a single triangle, with arbitrary values of the three cells at the corners. In each learning step the distance between the input vector and the *bmu* is added to an error-variable associated with the found *bmu*. After n_d learning steps the cell with the maximal error-variable is detected. This cell is called *bs* (black sheep). Then the farthest direct neighbour in the structure is detected and called *f*. The new cell is inserted in the middle between them:

$$pos(c_{new}) = \frac{1}{2} \cdot (pos(bs) + pos(f)) \qquad (2)$$

This new cell must be connected with surrounding cells in order to keep the triangular structure. The error-variable is initialized as a mean value of the error-variables of all *d* direct connected neighbours:

$$err(c_{new}) = \frac{1}{d+1} \sum_{i=1}^{d} err(neighbor_i) \qquad (3)$$

The error-variables of the neighbours are reduced according to that amount:

$$\forall i = 1, \ldots, d:$$

$$err(neighbor_i) = \frac{d}{d+1} err(neighbor_i) \qquad (4)$$

Shrinking cell structures

Starting from a single triangle and expanding the structure as described above will produce a *connected* structure trying to approximate the probability density P(X). This might be the inadequate model because P(X) can consist of several distinct regions with P(X)=0 between them. So there is a need for a procedure to disconnect the structure if necessary.

The basic idea is to find cells which are positioned in areas with P(X)=0 and to remove them. As the indicator of this constellation, the number of classifications without being *bmu* is recorded for each cell. If this number *k* for a certain cell exceeds the value in (5) this cell is removed and so are those of it´s neighbours necessary to return to a structure of triangles (p_s means the probability of keeping needed cells and *n* is the actual number of cells)

$$k > n \cdot n_d \cdot (1 - (1 - p_s^{1/n})^{1/(n_d+1)}) = k_r \qquad (5)$$

For a more detailed description of the algorithms so far see e.g. [3], [4].

3-2. Extensions & Adaptations

A number of problems with the above network-structure have being found regarding the concrete restrictions of our surface-clustering and classification. The central aspect here is the fact that our learning set is not fixed but consists of a continual flow of examples. So the task is not to model the best approximation of a limited set of input vectors, but to find a representation of the probability distribution and a good clustering of the *most recently* presented surfaces. Additionally the classification-function has to be accessible *all the times* (after a certain amount of learned surfaces), i.e. the structure should change smoothly from one state to the next while learning.

Buffering the flow of examples

The number of produced examples per time-interval is smaller than the number of surfaces that might be learned during this interval. So the received examples are buffered in a FIFO-list and each example is being learned several times.

Limited growing and adaptation

In the original model the learning phase simply stops when the required accuracy is reached. The SPIN-project does not know of an end of

learning, so there must be another definition of stability.

Assuming a tolerated error of $d_{accuracy}$ and an input vector x. If the *bmu* resulting from the classification of the vector x fulfils the equation (6)

$$d_{bmu} = \|x - pos(bmu)\| \leq d_{accuracy} \qquad (6)$$

then the learning for this cell is switched to a modified learning scheme: The fraction of *bmu* correction e_{bmu} is reduced by the factor Δe_{bmu} and the *bmu* is now moved towards x by this reduced fraction of $\Delta e_{bmu} \cdot e_{bmu}$. The direct neighbours are not corrected at all. Additionally this classification does not increment n_d, i.e. the generation of new cells is delayed.

If these classifications continue over some period of time, the moving of cells will become slower and no new cells are created. The modified learning scheme is reset to normal learning when one classification does not fulfil (6), i.e. e_{bmu} is set to the original value, etc.

Summarizing this modification, two main effects are important:

a. The growing of the network is stopped (or at least slowed down, depending on the classification error), although the flow of examples may still continue.

b. The network structure (i.e. the positions of the cells) is stabilized, when the examples fit into the clustering built up.

Cautious removal

The removal of cells in the original algorithms causes the removal of neighbouring cells, in order to return to a structure of triangles. This procedure does not care about the importance of these neighbouring cells, so often used cells might disappear. If the relative density of cells is large, then the remaining cells might fulfil most of the further oncoming classifications. But in a sparse clustering, i.e. each cell represents a whole isolated class (e.g. with P(X)=0 at it´s borders), the functionality of these cells can not be adequately approximated by the remaining ones. And even worse, these cells will be generated again in the further process, because the classification-errors in this area will rise significantly. On the way to this rebuilt structure, the remaining cells are corrected in large steps because of large classification-errors.

In some constellations we found a cyclic behaviour, i.e. even in the moment when the structure was build up again, one of the cells in this area is being removed, and the procedure starts again. This unstable and discontinuous behaviour can not be tolerated in our system.

The solution we have chosen is based on the idea of accepting "over-classification" under some circumstances, but not deleting cells, which are used. This means concrete that each time a cell is detected as not being used for a certain period of time, it is only removed when all the cells which would be removed in order to keep the structure of triangles intact are also detected as unused. As a result of this manipulation, the network will consist of more cells than necessary, and so one might think of effects like swapping between two cells in situations where one cell would be sufficient or other instability problems. But in our simulations we have never observed such situations.

Speed of forgetting

In the original model the value of k_r might be approximated as (see equation (5)):

$$k_r \leq \lceil n \cdot n_d \rceil ; (0 \leq p_s \leq 1) \qquad (7)$$

A value of p_s near 1 implies a good approximation of the vectors in the current learning set. But as our learning set is dynamic, another aspect arises: What happen to a learned cell, when the generating examples are deleted from the learning set. In our simulations this cell is removed or even massively corrected in a couple of minutes, i.e. there is absolutely no long-term memory.

In order to create a possibility to determine slower forgetting then implied by p_s=1, we extended the definition of k_r for values of p_s>1 (Notice that p_s is not a probability in this case):

$$k_r = \begin{cases} \left\lceil n \cdot n_d \cdot \left(1 - (1 - p_s^n)^{\frac{1}{n_d+1}}\right) \right\rceil & ; p_s \leq 1 \\ \lceil n \cdot n_d \cdot p_s \rceil & ; p_s > 1 \end{cases} \qquad (8)$$

So arbitrary long storage-times of presented surfaces can be implemented. One might think of additional information stored for every cell in S, like frequency of access or time since last access, etc. pp., in order to find a better choice of cells to delete, but this ideas are not tested here.

3-3. Parameters

This section is intended to give an idea of the meaning of some parameters as well as showing up the ranges which appears to be useful to us.

Network size

The size of the network structure is not determined directly by an upper or lower bound, but is implied by the required accuracy of the classification. Therefore this parameter needs not to be tuned at all (because it is not there).

Accuracy of classification

The accuracy of classification is the euclidian distance between the example-vector and the found *bmu* (see equation (6)). So the first stage of learning is reached when all the vectors of the learning set are in between of at least one of the hyper-spheres with radius $d_{accuracy}$ around the cells of the network. Then the modified learning-phase is entered, i.e. no new cells are created and the speed of moving and the number of cells moved in each step are reduced (until the classification errors rises again). The network is called "stable", when the value of e_{bmu} remains below a certain limit e_{stable}.

Choosing $d_{accuracy}$ too small results in a one-to-one mapping of the actual learning set and the cells in the structure, i.e. one cell is created for each vector in the learning set. On the other hand a large value of $d_{accuracy}$ means a small number of created cells and a large tolerated classification-error. Finding the "optimal" value depends widely on the purpose of the system and the used vector-representation of the examples.

Moving fractions e_{bmu}, $e_{neighbour}$ & Δe_{bmu}

Simulations have shown good results with values in range of (9) for e_{bmu}.

$$\frac{1}{20} \le e_{bmu} \le \frac{1}{5} \tag{9}$$

A "working range" for $e_{neighbour}$ is shown in (10), but there are some conditions, which one has to keep in mind.

$$\frac{1}{50} e_{bmu} \le e_{neighbor} \le \frac{1}{20} e_{bmu} \tag{10}$$

A small value for $e_{neighbour}$ might result in moving of cells not preserving topology in the network-structure, because one cell, which is marked as the actual *bmu* could be moved over a long distance, without moving the surrounding cells in an adequate manner. Large values for $e_{neighbour}$ are dangerous too, because if a neighbouring cell of a critical-cell is the *bmu*, the correction distance of this critical-cell could be larger than in the case that the critical-cell is the *bmu* itself.

Δe_{bmu} is limited by the following idea. When the value of e_{bmu} remains below the limit e_{stable}, the network is said to be stable. But this should imply that every vector of the learning set is classified *at least once* with a classification error smaller than $d_{accuracy}$ (in (11) $|Learning\ set|$ is the number of example vectors in the learning set).

$$e_{bmu} \cdot \Delta e_{bmu}^{|\text{Learning set}|} > e_{stable} \tag{11}$$

or

$$\left(\frac{e_{stable}}{e_{bmu}}\right)^{1/|\text{Learning set}|} < \Delta e_{bmu} < 1 \tag{12}$$

But these are only the trivial limits. Both bounds would not result in a reasonable learning behaviour, because a value of Δe_{bmu} too near to 1 produces a very slow convergence of the network without reaching a better accuracy and a value too near to the lower bound makes the system rest too early i.e. in sub-optimal positions of the cells. The simulations have shown good results for values in the range of:

$$\Delta e_{bmu} = \left(\frac{e_{stable}}{e_{bmu}}\right)^{\frac{1}{c|\text{Learning set}|}} ; c \in [3,6] \tag{13}$$

Removal of cells

Our simulations have shown an uncritical behaviour (with the new learning scheme) with values of p_s larger than 0.5. Each increase of this value depends only on the amount of time a learned example vector (or class of vectors) should remain in the network, without the need to "refresh" the corresponding cell with this example vector(s).

Creation of new cells

The factor n_d, which determines the number of learning steps before a new cell is inserted, should be chosen in a way that there can be "just

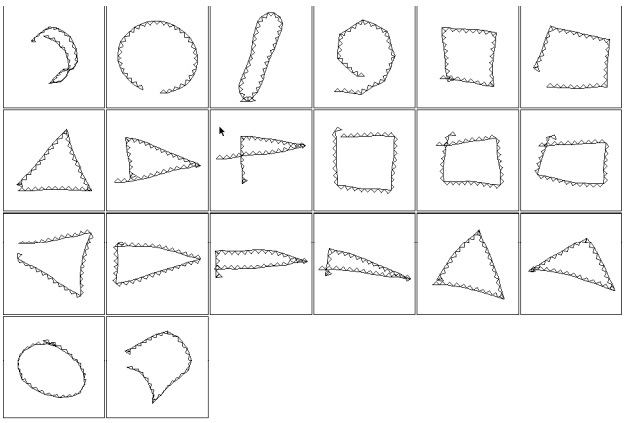

figure 1 : Some of the input surfaces

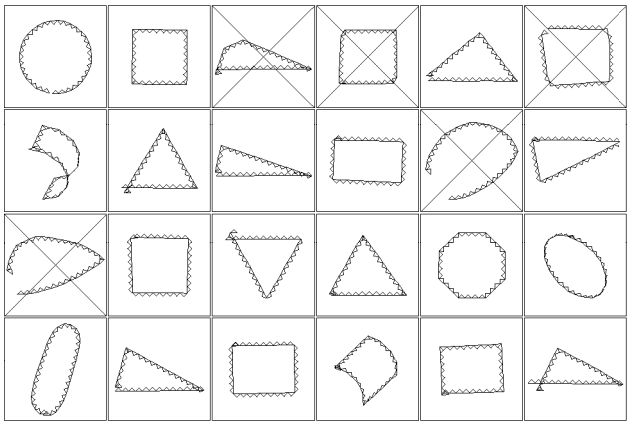

figure 2 : Generated classes

enough" probability-density-information accumulated before a position for a new cell is determined. Twice the number of expected classes works as a rough approximation, but obviously this "just enough" depends widely on the distribution of the example vectors.

4. Simulations

A flow of examples consisting of 20 different types of surfaces, which are all quantisized in 32 values for the segment curvature and in 16 entries for the surface curvature was simulated. The number of generated different examples is 2000. Noise is added in the form that every orientation of the bordering surface pieces is manipulated in the range of ±6° for the segment orientation and ±9° for the surface orientation. A part of the generated examples is shown in figure 1. As a result of adding noise, some surfaces are not closed, but this does not influence the generality of the test-set. After 6500 learning steps (1´ 50" on a MC68030 processor at 40 MHz clock) 19 different classes are generated (n_d=40, e_{bmu}=0.1, $e_{neighbour}$=0.005, Δe_{bmu}=0.998). In figure 2 the surface-classes are shown, with unused cells crossed out. The shown example is representative, i.e. all of the well defined test-sets have caused such a network behaviour.

5. Conclusion

An open problem in our structure is that most of the surfaces are learned relatively fast, and only a small number (one or two) of not still learned surfaces keeps the learning-phase running for quite a long time. A learning focused on these "problem-vectors" might be a solution, but this has not yet been tested in detail.

As a quite general technique for dynamic clustering problems (and based on the encouraging simulations), we are about to use this network on problems like topologic environment mapping, based on sparsely preprocessed sensor-informations, i.e we are trying to get additional refinements through further real applications.

References

[1] Alpaydin Ethem - 1.5.1991
GAL: Networks that grow when they learn and shrink when they forget
Technical Report 91-032

[2] Carpenter Gail A., Grossberg Stephen - 1987
A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine
Computer Vision, Graphics, and Image Processing 37, 54-115 (1987)

[3] Fritzke Bernd - 2.7.1991
Unsupervised clustering with Growing Cell Structures
Proc. of the IJCNN-91 Seattle (IEEE)

[4] Fritzke Bernd - 1991
Let It Grow - Self-Organizing Feature Maps with Problem Dependent Cell Structure
Proc. of the ICANN-91 Helsinki

[5] Fukushima Kunihiko - 1.3.1988
A Neural Network for Visual Pattern Recognition
Computer, March '88, pp. 65-75

[6] Kohonen Teuvo - 1.6.1990
Statistical Pattern Recognition Revisited
Advanced Neural Computers / R. Eckmiller (ed.)

[7] Kohonen Teuvo - 1984
Self-Organization and Associative Memory
Springer - Berlin, Heidelberg, New York, Tokyo