Tutorial

Using artificial neural networks for solving chemical problems

Part II. Kohonen self-organising feature maps and Hopfield networks

W.J. Melssen, J.R.M. Smits, L.M.C. Buydens, G. Kateman

*Laboratory for Analytical Chemistry, Faculty of Science, Catholic University of Nijmegen, Toernooiveld 1,
6525 ED Nijmegen (Netherlands)*

(Received 25 January 1993; accepted 27 April 1993)

Abstract

This second part of a Tutorial on neural networks focuses on the Kohonen self-organising feature map and the Hopfield network. First a theoretical description of each type is given. The practical issues concerning applications of the networks are then discussed. For each network, a description is given of the types of problems which can be tackled by the specific neural network, followed by a protocol for the development of the neural network system. It is seen that different neural networks are suited for different kinds of problems. Guidelines to avoid common difficulties in using neural networks are also given.

0. Contents

1. Kohonen self-organising feature map	268
1.1. Introduction	268
1.2. Theory	269
1.2.1. Network architecture	269
1.2.2. Data representation	269
1.2.3. Training the network	269
1.2.4. Example	270

Correspondence to: W.J. Melssen, Laboratory for Analytical Chemistry, Faculty of Science, Catholic University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen (Netherlands).

1.3.	Aspects of use	271
1.3.1.	Types of problems	271
1.3.1.1.	Topology	271
1.3.1.2.	Data reduction	271
1.3.1.3.	Clustering	271
1.3.2.	Protocol	272
1.3.2.1.	Data acquisition and selection	272
1.3.2.2.	Data preprocessing	272
1.3.2.3.	Network design	273
1.3.2.4.	Stop criteria	276
1.3.2.5.	Output analysis	277
2.	Hopfield networks	280
2.1.	Introduction	280
2.2.	General theory of Hopfield neural networks	280
2.3.	Discrete Hopfield network	282
2.3.1.	Processing	282
2.3.2.	Example of a discrete Hopfield network application	282
2.3.3.	Aspects of use of the discrete Hopfield network	283
2.4.	Continuous Hopfield network	284
2.4.1.	Processing	284
2.4.2.	Example of a continuous Hopfield network application	285
2.4.3.	Aspects of use of the continuous Hopfield network	287
3.	Short overview of neural networks	288
3.1.	Perceptron	288
3.2.	Adaptive linear element (ADALINE)	288
3.3.	Brain state in a box (BSB)	288
3.4.	Hopfield network	289
3.5.	Kohonen network	289
3.6.	Boltzmann machine	289
3.7.	Multi-layer feed-forward neural networks (MLF)	289
3.8.	Bidirectional associative memory (BAM)	289
3.9.	Learning vector quantisation (LVQ)	289
3.10.	Counter-propagation network (CPN)	289
3.11.	Adaptive resonance theory (ART)	290
	Bibliography	290
	References	290

1. Kohonen self-organising feature map

1.1. Introduction

The Kohonen network [1] belongs to the class of self-organising feature maps (SOFMs) and differs substantially from the multi-layer feed-forward (MLF) networks described in Part I [2]. Usually, a Kohonen neural network consists of a one- or two-dimensional framework of units,

hereafter referred to as the Kohonen map. Training of the network is performed in an unsupervised way, i.e., during the learning process the input patterns are presented without specifying what the output of the network should be. When training has been completed, the weight vectors of the units tend to approximate the distribution of the patterns in the object space. Due to the Kohonen learning rule, the individual weight vectors are arranged in the map and oriented in such

a way that the structure of the object space, i.e., the topology of that space, is represented as best as possible in the resulting map. Therefore, the Kohonen neural network is a topology-preserving mapping technique. The Kohonen mapping technique is primarily used for the examination of data sets for which no or only a little a priori knowledge concerning the internal structure is available. Once the network has been trained, each unit in the Kohonen map might be associated with an object class and then the map may be used for classification purposes.

1.2. Theory

1.2.1. Network architecture

Formally, a Kohonen network (Fig. 1) consists of two layers of units: an input layer and an output layer. The array of input units operates simply as a flow-through layer for the input vectors and has no further significance. Often this layer is left out, as is depicted in Fig. 1. The units in the output layer are ordered in a low-dimensional framework of units, e.g., a one-dimensional array or a two-dimensional matrix. Usually, one- or two-dimensional networks are used, and

henceforth, only these two types of Kohonen networks will be considered. Each unit in the network is fed by the input layer and is equipped with a single weight vector. The dimension of the weight vectors is equal to the number of components of the input vectors. No signals are transferred between units in the output layer.

1.2.2. Data representation

As in multi-layer feed-forward networks, the components of the input vectors or patterns may take binary or continuous values. Of course, the objects have to be represented in a fashion which retains as much information as possible and best suits the actual problem. Often the input vectors are scaled in some way, e.g., vector normalisation, before these are used for training the network. Because training is performed in an unsupervised way, no network output has to be specified.

1.2.3. Training the network

Since the introduction of the Kohonen neural network [3], several training strategies have been proposed (see, e.g., refs. 4–7) which deal with different aspects of use of the Kohonen network. In this section, we will restrict ourselves to the

Fig. 1. A two-dimensional Kohonen self-organising neural network. Shown is a 5×5 two-dimensional Kohonen network. An input vector is fed into the network via the formal input layer (not shown). The dashed box depicts a single unit and its associated weight vector. For a full explanation see text.

neural network which has been proposed by Kohonen [1]. The Kohonen learning algorithm can be subdivided into five clearly distinguishable stages which are presented briefly below. In Section 1.3 each stage will be discussed in more detail.

Before the training process is initiated, the weight vectors of the units in the output layer need some preparation. Therefore, for each unit j in the network, random values are assigned to the elements of its associated weight vector w^j . Then an input vector, x , is drawn randomly from the training set. For each unit in the output layer a predefined similarity or distance measure, $D(w^j, x)$, which operates on the unit's weight vector and the input vector, is determined. The unit in the Kohonen map possessing the most extreme value of $D(w^j, x)$, i.e., its weight vector is most similar (the similarity measure $D(w^j, x)$ is at a maximum) or close (the distance measure $D(w^j, x)$ is at a minimum) to the input vector, is declared as the winner. In addition, all units in the close vicinity of the best matching unit are selected too. Finally, the weight vectors of the winning unit and its neighbours are modified according to

$$w^j(t+1) = w^j(t) + \eta(t)N(t, r)[x - w^j(t)] \quad (1)$$

where t and $\eta(t)$ denote the iteration number and the learning rate, respectively. The $N(t, r)$ is a neighbourhood function (see Section 1.3.2.3) in which r indicates the distance in the Kohonen map between the unit j which has to be updated and the winning unit. Note that both the neighbourhood function and the learning rate explicitly depend on the iteration number. This completes the processing of one input vector. Next, a new input vector is drawn randomly from the training set and the learning process continues. When all patterns of the training set are presented to the network, one full processing cycle or iteration has been completed. Finally, the learning process stops after a predefined number of processing cycles.

As a result, the weight vectors of the winning unit and its neighbours are gradually moved along the straight line which connects w^j and x towards

Fig. 2. Adaptation of weight vectors. Depicted is the update of the weight vector $W(t)$ of the winning unit. A fraction, here indicated by the learning rate η , of the difference between the applied input vector X and $W(t)$ is added to the weight vector in order to construct the new weight $W(t+1)$. The weight vectors of the units in the neighbourhood of the winning unit are updated in the same way.

the applied input vector (Fig. 2). By this mechanism, input vectors which possess similar features will be mapped onto the same region in the Kohonen map. However, due to the unsupervised character of the learning algorithm, one cannot control which unit or cluster of units in the map will be associated with a specific input pattern or class of input patterns.

1.2.4. Example

Suppose that a one-dimensional Kohonen network consisting of four units has been trained with an arbitrary number of seven-dimensional continuous valued input vectors. A possible outcome of this training session is depicted schematically in Fig. 3. Each column (1–4) in this diagram represents the weight vector of a single unit. Each row (a–g) represents for all units in the map the values of the respective weight vector components. As can be seen, similar large values (filled circles) as well as small values (open circles) serve as connecting features between two neighbouring units. So, while going from one unit to another in the Kohonen map, the weight vectors of these units are chained together as in the game dominoes. By the self-organising topology-preserving mechanism, clustering of weight vectors possessing regions of similarity is established.

Fig. 3. Chaining of weight vectors by the Kohonen algorithm. See text for a full explanation.

If large values are set equal to 1 and small values to 0, this example demonstrates also that information concerning distances between vectors in data space might be lost. The real distance, measured in the metrics of the object space, between two weight vectors cannot be deduced from their actual positions in the Kohonen map. For instance, using Euclidean metrics, the weight vector of unit 1 (w^1) is closer in the object space to w^4 ($D(w^1, w^4) = \sqrt{5}$) than to w^3 ($D(w^1, w^3) = \sqrt{6}$), although unit 3 is positioned closer to unit 1 in the map than to unit 4. This phenomenon is caused by the folding over of a high-dimensional space which is forced to fit in a one-dimensional map [1,7].

Summarising this example, when a high-dimensional object space is mapped onto a one- or two-dimensional Kohonen map, the network tries to preserve the topology of the object space as best as possible. In this circumstance, information concerning the real distance between input vectors which are mapped on two non-adjacent units in the Kohonen network might be lost to a great extent.

1.3. Aspects of use

1.3.1. Types of problems

1.3.1.1. Topology. While training and analysing the Kohonen map, attention can be paid to, for instance, the structure or topology of the object space. If the end-points of the input vectors form

a planar circle in a high-dimensional object space, then this topology will be reflected by the ordering of the weights in the resulting Kohonen map. Another interesting application of the Kohonen network is the use of the map to estimate the probability density function of the object space. However, in that case the original Kohonen learning algorithm has to be slightly modified [8,5]. This extension will be discussed below.

1.3.1.2. Data reduction. Each of the input vectors can be associated with the map coordinates of the corresponding winning unit, or conversely, each weight vector in the map may be considered to be representative for one or more patterns of the training set. In the case where a high-dimensional object space is mapped onto a low-dimensional Kohonen map, a very efficient and quick way of data reduction might be achieved. However, the accuracy of this data reduction method depends highly on the variance present in each subset of input vectors which is mapped onto a single unit in the Kohonen map.

1.3.1.3. Clustering. The map facilitates the examination of clustering properties of the input vectors in the training set. If there are clusters present in the data set, the Kohonen map is likely to reflect these clusters provided there are enough units to make a separation between the clusters possible.

If clustering is observed, the Kohonen map may be used as a classifier. For this goal, every unit in the map must be labeled, i.e., associated with some (average) property of those object vectors that are mapped onto that particular unit. When this is done, units in the Kohonen map represent different classes of objects. When a new object is provided to the network, one of the units in the map will be assigned as the winner and the new object is assumed to belong to the same class this winning unit was labeled with. Of course, there has to be a sufficient degree of separability between the clusters. If the clusters touch or overlap in the original multi-dimensional object space, they are also likely to touch or overlap in the Kohonen map. In that case, it is more difficult to define the boundaries between

the clusters. An advantage of the Kohonen technique is that the clusters do not have to be separated by linear discriminant lines or (hyper) planes.

In conclusion, the Kohonen neural network is an elegant and powerful technique for mapping a generally high-dimensional object space onto a low-dimensional framework of units. In this way the map enables and facilitates a thorough investigation of high-dimensional data spaces. Eventually, the map may be used for classification purposes.

1.3.2. Protocol

1.3.2.1. Data acquisition and selection. As opposed to MLF networks, the Kohonen neural network is not very sensitive to unbalanced training sets. For example, if a large fraction of the input vectors is nearly identical, i.e., they form a relatively narrow and dense cluster in the object space, then these vectors are mapped onto a single unit in the Kohonen map. After convergence of the network, the weight vector of that particular unit will approximate the mean vector of this cluster. The remaining units in the map represent the less frequently appearing objects. It is not necessary to pay much attention to balancing of the training set. Moreover, the composition of the training set often is not known beforehand. In fact, usually

the Kohonen network is applied to gain insight into the training set.

1.3.2.2. Data preprocessing. As was previously mentioned, the input vectors are often scaled before use, but this might result to some extent in a loss of information. If vector normalisation is applied to two vectors which point in the same direction but which have different lengths, the normalised vectors are no longer distinct. However, scaling of the input vectors might be necessary, especially if different vector components have different ranges of values. For example, if the Euclidean distance is taken as a distance measure in the Kohonen learning algorithm, one has to pay attention to these different value ranges. If the range of a specific variable is large compared to the other variables, relatively small variations of this variable, i.e., relative to its range, may dominate the variations of the other variables. This may cause (perhaps important) changes in these other variables to be neglected by the Kohonen network. If this range effect is not desired, then range scaling per variable may be a proper strategy.

If different variables ought to have different influences on the similarity measure, each of the components of the input vectors should be multiplied by a suitable weighting factor before the similarity measure is calculated. Of course, these

Fig. 4. Three possible network structures. (a) One-dimensional network. (b) Two-dimensional rectangular network. (c) Two-dimensional hexagonal network. Note that the lines drawn between the units do not represent physical connections, but emphasise the particular arrangement of the units in the network.

remarks concerning the scaling of input vectors are not only valid for Kohonen networks but apply also to other computational methods which are based on vector similarity or distance measures.

1.3.2.3. Network design. After preprocessing of the data, the Kohonen network has to be designed. Decisions have to be made about network dimensionality, size, connectedness of the units in the map, weight vector initialisation, similarity or distance measure, neighbourhood function, and learning rate.

Dimensionality, size, and connectedness of the network. Usually, one- or two-dimensional Kohonen networks are used. Higher-dimensional Kohonen networks may be applied as well, but such networks require, especially when dealing with multi-dimensional input vectors, a tremendous number of computations before convergence of the weight vectors can be achieved. Moreover, the output of such high-dimensional networks is difficult to interpret and visualise in a convenient manner. Three widely used network configurations are the one-dimensional network and the rectangular and hexagonal two-dimensional networks (Fig. 4). The structure of the network determines the degree of connectedness of the units. Except for the units located at the borders of the Kohonen map, each unit in the networks depicted in Fig. 4 has (a) two, (b) four, or (c) six neighbours, respectively.

The choice of the size and dimensionality of the network depends on the problem type. If the network is intended to be used as a classification system, one rule of thumb is to take at least twice as much units as the number of classes expected in the training set. On one hand, if not enough units are chosen, input vectors that differ substantially are forced to map onto the same unit and hence no good separation is achieved. On the other hand, if there are too much units, input vectors which are very similar will eventually become separated in the map. A substantial number of units might even be wasted because none of the input vectors is mapped onto these units. If the number of units is almost equal to the number of input vectors in the training set, the net-

work might operate as a kind of memory and not represent in a reliable way (in a statistical sense) the general features present in the object space. Combining the aforementioned requirements it is advised to choose the number of units, N_{units} , in the map according to

$$2N_{\text{classes}} < N_{\text{units}} \ll N_{\text{patterns}}$$

where N_{classes} and N_{patterns} represent the expected number of classes and the number of patterns in the training set, respectively.

Weight vector initialisation. Before training, one may initialise the weight vector components with small random values. However, usually the input training vectors are not distributed randomly. In such cases many units might never be assigned as the winner and consequently, the corresponding weight vectors will not be used at all. This may lead to a situation in which there are too few weight vectors left to represent properly all different features present in the training set.

Obviously, the optimal initial distribution of the weight vectors is the one which approximates the distribution of the input vectors in the object space. However, in most applications this is precisely the information the Kohonen network should yield. When the network is meant to be used as a classifier, the mean vectors of the known classes form good starting points for the individual weight vectors. Again, this information is, in general, not available beforehand.

One way of dealing with the problem that a lot of weight vectors are 'out of reach' is to add random noise to the input vectors. During training the amplitude of the noise is gradually decreased. In this way, units possessing weight vectors which are very dissimilar to all input vectors will have a higher chance of being assigned as a winner, but still this might not be sufficient. Another solution is to choose all weight vectors close together, preferably nearby the origin, meanwhile adapting the set of input vectors in such a way that these point to the same region in the object space as the weight vectors do. During training, the components of the input vectors are slowly changed back to their original values and the weight vectors are likely to follow these altering input vectors.

Unfortunately, both methods described above are rather time-consuming. The problem of unused or infrequently used weight vectors may be circumvented by using a modified Kohonen learning algorithm described by Desieno [8]. In this adapted version, the Kohonen map will produce a set of so-called equiprobable weight vectors. These are a set of weights such that an input vector randomly chosen in accordance with the probability density function of the object space will have an equal probability of being closest to each of the weights in the Kohonen map [5]. Basically, each unit keeps track of the frequency, f_j , at which it had been assigned as a winner. If $f_j > 1/N$, where N denotes the number of units in the map, then this unit will have a smaller probability that its weight vector will be updated. In contrast, units with a lower assignment frequency, $f_j < 1/N$, will have a higher probability of being updated. Thus, very 'busy' units might temporarily be shut down allowing other less frequently updated units to be assigned as a winner instead. While doing this, the distribution of the weight vectors in the Kohonen map becomes statistically equal to the distribution of the vectors in the training set. The resulting Kohonen map may be used as an estimator for the probability density function of the object space.

The most elegant way to profit from as many weight vectors as possible is to use a wide neighbourhood function. First, the weight vectors are set equal to the mean of the training set and then random noise is added to each weight vector. In the beginning of the training, not only the winning unit is updated, but also units in the neighbourhood of this winning unit. If this neighbourhood of units is chosen large enough, the map will reflect the rough structure of the object space after just a few processing cycles. While processing continues, the number of neighbouring units is decreased. So, in time less and less units are updated allowing the network to tune in on specific features present in the data set. This mechanism will be described in more detail under the topic 'Neighbourhood function' below.

Similarity and distance measures. In the literature, many definitions of vector similarity and distance measures can be found. Each of these,

e.g., scalar product, direction cosine and Minkowski distance, may be used in the Kohonen learning algorithm to determine which unit will be assigned as the winner. The scalar or inner product, $D_S(\mathbf{a}, \mathbf{b})$, obtained from two vectors, $\mathbf{a}$ and $\mathbf{b}$, may be used as a similarity measure and is defined by

$$D_S(\mathbf{a}, \mathbf{b}) = \sum_i a_i b_i \quad (2)$$

where a_i and b_i are the i th elements of $\mathbf{a}$ and $\mathbf{b}$, respectively. If a difference in direction between two vectors $\mathbf{a}$ and $\mathbf{b}$ is considered more important than a difference in magnitude, the direction cosine, $D_D(\mathbf{a}, \mathbf{b})$, is a more appropriate similarity measure:

$$D_D(\mathbf{a}, \mathbf{b}) = \cos(\theta) = \frac{D_S(\mathbf{a}, \mathbf{b})}{\|\mathbf{a}\| \|\mathbf{b}\|} \quad (3)$$

where θ is the angle between $\mathbf{a}$ and $\mathbf{b}$, and $\|\mathbf{a}\|$, $\|\mathbf{b}\|$ represent the norm of $\mathbf{a}$, $\mathbf{b}$, respectively. Obviously, $D_S(\mathbf{a}, \mathbf{b}) = D_D(\mathbf{a}, \mathbf{b})$ when vectors are normalised to unit length. Another suitable measure, in fact a vector distance measure in a metrical system, is the Minkowski distance $D_M(\mathbf{a}, \mathbf{b})$ which is defined by

$$D_M(\mathbf{a}, \mathbf{b}) = \sqrt[n]{\sum_i |a_i - b_i|^n} \quad (4)$$

If $n = 1$, the Minkowski distance measure is called the Manhattan or city block distance and for $n = 2$, it corresponds to the well known and commonly applied Euclidean distance measure. Other measures like the Tanimoto similarity measure and the Mahalanobis distance [1] may serve as well.

Learning rate η . Usually, the learning rate, η , is chosen relatively large in the early stages of the training and is decreased linearly as a function of the number of processing cycles. A good starting value of η might be 0.1–0.5, whereas at the end the learning rate should be in the range 0.0001–0.0005.

Neighbourhood function. In Kohonen networks updating a unit corresponds to changing its weight vector. This change is proportional to the difference between the input vector and the weight

vector that is to be updated (Fig. 2). The fraction that is actually used to calculate the new weight vector depends not only on the learning rate, $\eta(t)$, but also on the neighbourhood function, $N(t, r)$ (Eqn. 1). Both the iteration number, t , and the variable, r , which refers to the distance in the map between the winning unit and the unit to be updated, appear in this function.

Several strategies concerning the choice of the shape and the adaptation of the width of the neighbourhood function, $N(t, r)$, have been proposed in the literature [1,7]. In the original Kohonen algorithm, the width of the neighbouring region is step-wise decreased during the training session, causing a diminishing number of units to be updated together with the winning one. An example of a shrinking square-shaped neighbourhood is depicted schematically in Fig. 5. Initially, the neighbourhood function covers almost the entire Kohonen map and after a few processing cycles the network has copied more or less the rough structure which is manifest in the training

set into the weight vectors. During this phase, the formation of the global object space topology has been established in the map. Next, after a fixed number of processing cycles, the width of the neighbourhood function is decreased and the network will start linking weight vectors bearing similar features in their components together. In the last phase of the training, only the weight vector of the winning unit is updated and the weight vectors in the Kohonen map become more and more specialised to specific, perhaps class-characteristic, features present in the input vectors.

Regardless of the shape of the function, $N(t, r)$, it is recommended that the shrinking of the width of the neighbourhood be scheduled in such a way that the number of iterations at which a certain width is applied is inversely proportional to the square of that width. Considering the 5×5 network illustrated in Fig. 5, an appropriate sequence of neighbourhood widths is given by the linearly decreasing series 5, 3, and 1, implying that maximally 25, 9, and 1 units are

Fig. 5. Adaptation of neighbourhood during training of a Kohonen network. Shown are three stages during the training of a two-dimensional 5×5 sized Kohonen network: first a 5×5 neighbourhood is applied which covers the entire map because the winning unit is located precisely at the center of the map. Next, a shrunk 3×3 neighbourhood of units accompanies the winning unit, and during the final stage the winning unit is updated exclusively. For full details see text.

Fig. 6. Four types of neighbourhood functions. In each of the diagrams the winning unit is located at the center of the abscissa. Horizontal axes indicate the distance r to the winning unit, vertical axes indicate the value of the neighbourhood function $N(r)$. Depicted are (a) a block, (b) a triangular, (c) a Gaussian bell, and (d) a mexican-hat shaped neighbourhood function, respectively. The height of the function, $N(t, r)$, indicates what fraction of the difference $\Delta w^j = x - w^j$ is used to update the weight vector w^j at iteration t . A positive value indicates that the weight vector is slightly moved towards the input vector, whereas a negative value (d) means that the weight vector is pushed away from this input vector.

updated simultaneously, respectively. Exclusively odd numbers appear in this series because only then can a neighbourhood centered symmetrically around the winning unit be obtained. Given the ratio of the reciprocal squared widths, $\frac{1}{25} : \frac{1}{9} : 1$, the recipe above prescribes that the widths — 5, 3, and 1 — should be applied subsequently at $\frac{9}{259}T$, $\frac{25}{259}T$, and $\frac{225}{259}T$ iterations, where T denotes the predefined total number of iterations.

In Fig. 6 four different neighbourhood functions are shown. Using a one-dimensional network, the distance, r , is simply equal to the number of units which separate the winner and the unit to be updated, incremented by one. For higher-dimensional network configurations r may be defined in several ways. If the map is two-dimensional, one defines r to be, e.g., the radius of a circle or the half-width of a square, both centered around the winning unit, respectively.

Before discussing the implications of the choice of a particular neighbourhood function, the following point must be made. In the original Kohonen algorithm [1] the boundaries of the map are considered as 'hard' delimiters. When a winning unit is located near the edge of the map, part of the neighbourhood function vanishes to the exterior of the map. For some applications it might be

more appropriate to let the neighbourhood function scroll from one edge of the map to the opposite one [9]. As a result, if the winning unit is located at the upper-left corner of the map, 'neighbouring' units positioned at the lower-left, upper-right, and lower-right corners of the map will be updated too, provided the width of the neighbourhood function is large enough. If a two-dimensional Kohonen network is trained in this 'modulo' fashion, one can imagine that the map is ideal in representing the topology of the surface of a three-dimensional sphere.

Since the introduction of the Kohonen network various neighbourhood functions have been proposed [1,10,7,9]. Kohonen used a block neighbourhood function (Fig. 6a). Especially for those applications in which low-dimensional object spaces are investigated this function might suffice. Application of a triangular (Fig. 6b) or Gaussian bell-shaped (Fig. 6c) neighbourhood function might lead to a more smooth formation of topology in the map and a faster convergence of the weight vectors [10]. The Mexican-hat function (Fig. 6d), which has its origin in biological neural networks, may be useful when constructing a Kohonen map for classification purposes. This typical shape, a positive peak at the center and a negative valley at surrounding regions, enhances the sharpening of the boundaries between two or more adjacent clusters. The clusters might even become separated in the map by strings of units onto which none of the input vectors is mapped.

1.3.2.4. Stop criteria. Usually, the training of a Kohonen network stops after a predefined number of processing cycles. Kohonen recommends that each input vector of the training set should be presented to the network at least 500 times the number of units in the map. An alternative is to trace during the learning process the variations of each of the weight vector components. One easy way of doing this is to calculate for each unit in the map the sum of the squared differences between the respective components of the current weight vector and the previous one. When each of these sums becomes (negligibly) small after completion of a processing cycle, the net-

work has settled into a stable weight configuration and the training session may be interrupted.

1.3.2.5. Output analysis. Once the training of the Kohonen neural network has been completed, the weight vectors are fixed and the map is ready to be used. The way in which the weights and the map are analysed depends on the type of the problem considered. In this section some possible analysis methods will be discussed.

Output activity map. When training has been completed, for a given input vector, x , only a single unit in the map is assigned as the winner. However, the input vector might be similar or close to one or more of the other weight vectors, w^j , too. During training of the Kohonen network the selection of the winning unit was based on a vector similarity or distance measure, $D(w^j, x)$. Preferably the same similarity or distance measure should be used afterwards to define the output, y^j of unit j in the map according to

$$y^j = D(w^j, x) \quad (5)$$

The resulting output activity map allows the examination of regions of similarity or adjacency of the vector, x , and the weights, w^j . Keep in mind that, in general, every input vector is characterised by a different output activity map. The output activity map may be represented graphically by a grey-encoded raster diagram (Fig. 7a) or an iso-contour plot in order to facilitate the examination of regions of (dis)similarity.

Cluster analysis. Another method of analysis is provided by the so-called counting map [11]. The counting map is obtained by calculating for each unit the number of input vectors of the training set that is mapped onto this specific unit. Despite the unsupervised character of the Kohonen learning algorithm, this map might provide evidence for the existence of two or more clusters in the (high-dimensional) object space. An example of a counting map exposing three well-defined clusters is depicted in Fig. 7b.

If the separation between clusters is less obvious, i.e., there are no strings of units in the map which are never or not often assigned as winner, it might be an error prone task to define such clusters. In this circumstance, examination of the

Fig. 7. Analysis of the Kohonen map. Four different ways of analysing a two-dimensional Kohonen map. (a) Grey-encoded output activity map for a single input vector x . Dark areas in the map indicate that the input vector is similar to the respective weight vectors, whereas light areas correspond to regions of dissimilarity. (b) Graphical representation of the counting map. Dark areas indicate that a large number of input vectors is mapped onto this (cluster) of units. Units onto which none or just a few input vectors are mapped are represented by white or light-grey. Three clearly separable clusters can be observed in the map suggesting that at least three clusters are present in the object space as well. (c) A feature map. If a unit is labeled with +, - or X, one or more input vectors labeled with +, - or both, respectively, are mapped upon this unit. Blank units are never assigned as winner. The Kohonen map appears to separate the two feature values reasonably well. (d) A superimposed feature map. If a unit is labeled with A, B, C or X, one or more input vectors labeled with class A, B, C or both B and C, respectively, are mapped upon this unit. Blank units are never assigned as winner. In the upper-left area of the map, a cluster of A appears. This area is well separated from regions B and C in the map. Note that the cluster of B partially merges with C (denoted by X).

weight vectors may offer a solution. Note that the existence of a boundary between two clusters might be expected when one observes a relatively large difference, as expressed by a similarity or distance measure, between the weight vectors of two neighbouring units. In order to define the boundaries between two or more clusters, first calculate for two neighbouring weight vectors a similarity or distance measure, $D(w^j, w^k)$, which

does not necessarily have to be the same measure as the one embedded in the Kohonen learning algorithm. Then proceed by defining a minimum similarity or maximum distance level, i.e., a threshold. If the value of $D(\mathbf{w}^j, \mathbf{w}^k)$ exceeds this threshold, then a boundary is defined to be present between the units j and k . Then a subdivision of the Kohonen map into multiple clusters may be established. The clustering obtained in this way, of course, is highly determined by the chosen value of the threshold.

Labeling of input vectors. The Kohonen algorithm is unsupervised and therefore it does not make use of output vectors to guide the adaptation of its weight vectors. However, if output vectors are known, the Kohonen map may be used afterwards for studying the relationship between the topology found in the input space and the distribution of a specific variable of the output vectors.

For every variable (feature) of the output vector the following analysis may be performed. Since each input vector is associated with an output vector, one may label each input vector with the value of the specific output vector feature that is being studied. If a labeled input vector is now presented to the network, it is projected onto the map, resulting in the assignment of a winning unit. This winning unit gets the same label as the input vector. After presentation of all input vectors, every unit in the map is labeled with zero, one, or more (perhaps different) labels. For example, a specific feature has the value + or -. The resulting so-called feature map as shown in Fig. 7c mimicks the relation between the topology of the input space and the value of this feature. Such a feature map may be generated for every feature of the output vector. If information on the number of input vectors that map per unit is added, a grey-encoded feature map results [11].

If the output vectors represent different classes, the analysis described above is more straightforward. Again, one could generate a feature map for every feature (in this case a class). However, since an input vector is defined to belong to only one class, one may as well simply label all the input vectors only once with a class symbol (e.g., class A, B or C). Following the same

procedure as described above, a superimposed feature map results (Fig. 7d).

A multiple assignment (denoted by 'X' in Fig. 7c and d) occurs when input vectors with different labels map onto the same unit. When the Kohonen network is intended to be used as a classification system, these results might cause severe trouble. Retraining the network with more units, another network structure or dimensionality, or a different similarity or distance measure then might offer a possible solution to circumvent this problem. However, a multiple assignment might also suggest that one of the predefined features or classes is in fact a sub-class of one of the other features.

When using the Kohonen network as a classifier, the classification abilities of the network can be further optimised by incorporating the learning vector quantisation (LVQ) algorithm. By means of the LVQ method the Kohonen network is (re)trained in a supervised way. For a detailed discussion on this topic the reader is referred to ref. 1. By inspection of the feature maps, one can study whether input vectors representing the same features are separable from input vectors associated with other features, or whether input vectors representing different features tend to cluster together in the Kohonen map.

Weight vector analysis. To reveal more insight in the topology of the data set one is exploring, it would be worthwhile to pay attention to the weight vectors of the Kohonen network. For the moment it is assumed that the training set consists of two-dimensional input vectors and, thus, the weight vectors are also two-dimensional. The components of each weight vector may be regarded as the coordinates of a point in a two-dimensional plane. Each pair of weight vector end-points in this plane is connected by a straight line if, and only if, the corresponding two units in the Kohonen network are direct neighbours (Fig. 4). By using this representation, one can visualise in an elegant way the structure of the space spanned by the weight vectors which, in turn, reflects the topology of the object space.

Moreover, during the training phase, this kind of analysis also can be performed each time one or more processing cycles have been completed.

The resulting Kohonen movie [1] gives an impression of the evolution of the weight vectors. An example thereof is depicted in Fig. 8, in which a two-dimensional object space is mapped onto a one-dimensional Kohonen map. The Kohonen movie provides visual feedback during the training session and may serve as an additional indicator as to whether the training session should be interrupted or not.

Due to the unsupervised self-organising character of the Kohonen learning algorithm, one cannot control which unit in the map will be assigned to a particular region in the object space. Especially when the weight vectors are initialised in a random fashion, the final map might be rotated, e.g., by 90° , 180° , or 270° with respect to the actual orientation of the object space. Considering the olympic ring shaped object space of Fig. 8, a 180° rotation corresponds to the situation that the ring configuration in the map will become turned upside-down. Another problem which might occur is that the map becomes twisted. For example, a rectangular-shaped object space might appear as a butterfly-like structure in the Kohonen map [1].

When dealing with three-dimensional input vectors, the Kohonen movie analysis can be extended by replacing the two-dimensional coordinate system in which the weight vector points are graphically represented by a three-dimensional one. In the case of high-dimensional input vectors, this type of analysis might still be of great value. However, some a priori knowledge would be of great help in selecting the two or three components of the weight vectors which are visualised in the Kohonen movie.

Fig. 8. Kohonen movie. A one-dimensional Kohonen network consisting of 180 units was trained with 1440 two-dimensional vectors of which the end-points formed the symbol of the Olympic games. While training the network, the Euclidean distance measure was used together with the block neighbourhood function $N(t, r)$ depicted in Fig. 6a. The upper diagram shows the weight configuration just after the initialisation procedure. In the second and third diagram the weights begin to reflect roughly the structure of the training set. After convergence of the network (lower diagram) the weight vectors mimic nicely the topology (the five-ring configuration) of the object space.

An alternative approach is to represent each of the weight vectors by a single graph, thereby arranging the graphs according to the ordering of the corresponding units in the Kohonen map. This kind of representation might be of great value if one is examining multi-dimensional object spaces spanned by vectors of which the components represent the values of some ordered series of variables, e.g., spectra or time-series. Such an analysis may also be performed per weight vector component, resulting in a weight vector component map [12]. This map is obtained by drawing an iso-contour plot of a specific weight vector component over the units. Such a map may be generated for each weight vector component, and it may reveal the distribution of the particular input vector variable over the entire map.

2. Hopfield networks

2.1. Introduction

The renewed interest in artificial neural networks was triggered by the introduction of a computational neural network developed by Hopfield [13]. This network functions as an auto-associative memory for storage and recall of objects and may be used for noise reduction. In the following years, this so-called discrete Hopfield network was modified and improved, resulting in the continuous Hopfield network [14–17]. The latter network also operates as an auto-associative memory, but, as will be discussed below, it may also serve as a computational optimisation network. Applied in this way, the Hopfield network is able to solve a large variety of optimisation problems, i.e., those problems for which a possible solution can be associated with a cost function expressing the quality of that particular solution. Applications of Hopfield-like neural networks are given in refs. 18 and 19.

In almost the same time period, Kosko [20,21] proposed the adaptive bidirectional associative memory (ABAM) which essentially is a slight extension of the Hopfield network. In addition to auto-association, the ABAM is able to learn in an adaptive way, as in multi-layer feed-forward net-

works, hetero-association too. Because the underlying theoretical concept is roughly the same for these two types of neural networks, only the discrete and continuous Hopfield networks will be discussed.

2.2. General theory of Hopfield neural networks

First some basic concepts will be outlined which are valid for both the discrete and the continuous Hopfield networks. Both types of networks behave like dynamical systems. The output state of each unit in the network is updated in time (processing) according to some rule until convergence has been achieved. In this way, Hopfield networks differ from MLF networks in which the weights between the units are updated. Both types of Hopfield networks presented in this paper are deterministic models. Given a certain initial state of the network, the dynamic behaviour of the network, i.e., the trajectory describing the development in time of the output states of all units, is completely determined by this initial state. Stochastic Hopfield networks, where the output state of each unit is updated stochastically according to some probability function and, hence, the dynamic behaviour of the network becomes stochastic too, are beyond the scope of this Tutorial.

Fig. 9. Architecture of the Hopfield neural network. This figure shows a linearly arranged Hopfield network consisting of N units (enumerated open circles). The output of each unit is connected to the input of any other unit in the network (thin lines). The black dots at intersecting lines above the units represent the weights W_{ij} . Two of the weights, i.e., W_{23} and W_{N1} , are shown explicitly in the figure. Each unit is driven by an external signal I_i (bold lines drawn above the units).

Fig. 10. Alternative architecture of the Hopfield neural network. An alternative matrix-wise arrangement of the units in the Hopfield network is depicted in Fig. 9. Not all connections between the units are shown.

Hopfield networks operate on a specific class of input patterns, namely N -dimensional binary vectors, $\mathbf{x}$, the components of which take only the values 0 or 1. As a consequence, each pattern is located on a vertex of an N -dimensional Hamming hypercube and, hence, precisely 2^N different patterns can be represented.

The architecture of a network consisting of N units is depicted in Fig. 9. The size of the network, N , is determined by the dimension of the input patterns. Each unit i in the network receives input from all other units j via the weights W_{ij} . Therefore, the weights can be arranged in an $N \times N$ sized matrix $\mathbf{W}$. The weight matrix is symmetric, $W_{ij} = W_{ji}$, and direct feed-back loops are omitted, so $W_{ii} = 0$. In addition, each unit is driven by a constant external signal, I_i . Because the network is fully interconnected, the units may be arranged in any particular way favoured to simplify visualisation or interpretation of the output, e.g., in an array (Fig. 9) or a matrix (Fig. 10).

As opposed to perceptron-like networks, the weight matrix of the Hopfield network is constructed in advance. Suppose we would like to store in the auto-associative Hopfield network the training set, S ,

$$S = \{\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3, \dots, \mathbf{x}^P\} \quad (6)$$

which consists of P N -dimensional input vectors, $\mathbf{x}^p$. Then, considering the Hebbian learning rule (see, e.g., ref. 5), which states “the strength of the connectivity between two units in a (biological) neural network has to be reinforced when these units exhibit correlated activity”, the weight matrix for the Hopfield network can be constructed [4,13] according to:

$$W_{ij} = \begin{cases} \sum_{p=1}^P (2x_i^p - 1)(2x_j^p - 1) & \text{if } i \neq j \\ 0 & \text{else} \end{cases} \quad (7)$$

Note, that a positive correlation between the i th and j th component of the pattern, $\mathbf{x}^p$, (thus, for binary vectors, $x_i^p = x_j^p$) will lead to an increase of the weight, W_{ij} , between the units i and j , whereas a negative correlation ($x_i^p = 1 - x_j^p$) leads to a decrease of that weight.

The total input of unit i , net_i , is given by the equation

$$net_i = \sum_{j=1}^N W_{ij} o_j + I_i \quad (8)$$

where o_j represents the output state of unit j . The output of unit i is given by

$$o_i = g(net_i) \quad (9)$$

where $g(net_i)$ represents some transfer function, which is different for each type of Hopfield network.

For a better understanding of how Hopfield networks work, we have to introduce the concept of the network energy function. Given a certain weight matrix, $\mathbf{W}$, a set of external signals, I_i , and the state of the network (that is, the set of all output states, o_i), then the generic network energy function E [13,15] is defined by

$$E = -\frac{1}{2} \sum_i \sum_j o_i W_{ij} o_j - \sum_i I_i o_i + \sum_i T_i(o_i) \quad (10)$$

where the last term, $T_i(o_i)$, depends on the Hopfield network type: discrete or continuous. It can be shown (for a thorough discussion on this topic see, e.g., ref. 4) that the energy function fulfills the following requirements: (1) Any change in the state of the network results in a decrease of the energy E . (2) The energy function is bounded

below: $E \geq E_{\min}$. (3) When the network energy changes, it must change by a finite amount.

The first two rules show that E is a so-called Lyapunov function. As a consequence, during processing the network energy will be minimised and the state of the network will evolve in time until a minimum, local or global, of the energy function has been encountered. Then, the network stops processing and remains in its final state. The latter state in fact forms a stable state of the dynamical Hopfield system. The third rule guarantees that this stable state, representing the solution of the network to the initially given problem, is reached within a finite time-span.

The Hopfield network typically is a single-layer network. Apart from the external driver channels, I_i , there is no explicit subdivision made in input and output units as is the case in MLF networks. In Hopfield networks a new (unknown) pattern, u , is loaded into the network by making the initial unit output states equal to the components of this pattern:

$$o_i(t_0) = u_i \quad \text{for } i = 1, 2, \dots, N \quad (11)$$

Note that the choice of the constant external driver signals does not depend on the applied pattern, u . Next, the output of each unit in the network is updated resulting in a new output state, $o_i(t+1)$. When all units in the network are updated, one basic processing cycle has been completed. Equipped with these basic concepts and ideas, the major differences between discrete and continuous Hopfield networks now will be discussed.

2.3. Discrete Hopfield network

2.3.1. Processing

For the moment, it is assumed that the weight matrix, W , (Eqn. 7) already has been composed in order to store the pattern set S (Eqn. 6) in the network. The units in the discrete Hopfield network are equipped with a threshold transfer function (Fig. 11a). Given the total input, net_i , at time t , the output is updated according to

$$o_i(t+1) = \begin{cases} 1 & \text{if } net_i(t) > U_i \\ o_i(t) & \text{if } net_i(t) = U_i \\ 0 & \text{if } net_i(t) < U_i \end{cases} \quad (12)$$

Fig. 11. Transfer functions: (a) threshold; (b) sigmoid.

in which U_i represents the threshold of the transfer function. The extra term in the energy function (Eqn. 10) is equal to [14]

$$T_i(o_i) = U_i o_i \quad (13)$$

Without loss of generality, however, the threshold and the external driver signals may be chosen as $U_i = 0$ and $I_i = 0$, respectively.

2.3.2. Example of a discrete Hopfield network application

Suppose that the four patterns depicted in Fig. 12 have to be stored in the discrete Hopfield network. These patterns are represented as $N_R \times N_C$ sized data matrices ($N_R = 7$, $N_C = 6$) which contain the binary numbers '0' (shown as white boxes) and '1' (black boxes). In order to construct the weight matrix, the pattern matrices, X^p , have to be converted to vectors, x^p (see Eqn. 7). This is easily done by the index transformation

$$x_k^p = X_{ij}^p \quad \text{with } k = (i-1)N_C + j \quad (14)$$

This preprocessing yields four N -dimensional vectors ($N = N_R N_C = 42$) and the weight matrix can be computed. Suppose that we have an unknown pattern, U , which is a noisy exemplar of the stored pattern, X^1 (in fact, each 'bit' of the pattern, X^1 , was flipped with a probability of 25%). Next, take the preprocessed pattern, u , as

Fig. 12. Example of pattern set and processing of a noisy pattern. (a) Pattern set for the discrete Hopfield network (see text). The patterns (7×6 sized matrices containing the binary numbers 0 and 1) are graphically represented as black ('1') and white ('0') images. (b) The left diagram represents the initial output state (a 25% corrupted exemplar of the pattern X^1) of the network. Black and white boxes indicate unit output values of '1' and '0', respectively. The other diagrams show three consecutive processing cycles. The right diagram ($t = 3$) shows the final state (i.e., the stored pattern X^1) of the network. (c) Depicted is the complement of the pattern X^1 . (d) Spurious state of the network. This state contains fractions of some of the patterns shown in (a).

the initial output state of the network (Fig. 12b, $t = 0$). To facilitate comparison with the original pattern matrices, the units in the Hopfield network are rearranged in such a way that each unit on position (i, j) represents the corresponding component of the pattern matrix which has to be recalled.

The transfer function thresholds and the external driver signals are set to zero. After calculation of the total input of each unit, i , (Eqn. 8) the new output state $o_i(t+1)$ can be determined. The result is shown in the second middle diagram ($t = 1$) of Fig. 12b. Application of Eqn. 10 reveals that $E_{t=1} < E_{t=0}$. The network continues processing (Fig. 12b, $t = 2$) until after three processing cycles the network reaches a stable state which is identical to the (noise-free) pattern, X^1 (Fig. 12b, $t = 3$). By using Eqn. 10 again one indeed obtains that $E_{t=3}$ is equal to the global minimum of the energy function.

In summary, if an unknown noisy pattern is provided, the network will recall one of the previously stored patterns. During this processing the network energy function is minimised and, in general, the recalled pattern is in a Hamming distance sense * as close as possible to the initially imposed unknown pattern.

2.3.3. Aspects of use of the discrete Hopfield network

The previous example showed that the discrete Hopfield network is straightforward in use, since unlike perceptron-like networks, no optimal set of network parameters has to be found, and is very fast in finding a solution to the presented problem. However, there are some pitfalls. In this section the most common problems are discussed and some remedies are given.

First of all, the maximum number of patterns, $P_{\max}$, that can be stored in the network is determined by the size, N , (i.e. the number of units) of the network. Note that the network size, in turn, is determined by the dimension of the patterns. It can be shown [13] that

$$P_{\max} \leq 0.15N \quad (15)$$

In fact, twice as much patterns are stored in the network because the complements of the patterns in the set, S , are stored as well. This property can be deduced from Eqn. 7. As a consequence, the network might give such a 'complementary' solution (Fig. 12c) because it is also a legitimate stable state of the network. This symmetry can be broken by taking appropriate non-zero transfer function thresholds, U_i , forcing the Hopfield network to favour the stored pattern or its complement. This property can be derived from the network energy function (Eqn. 10). Ignoring the complementary patterns, about $0.15N$ patterns can theoretically be stored simultaneously in the network before errors during the recall process become severe. An overloaded network might not 'remember' all stored patterns properly. For in-

* For two binary vectors, x and y , the Hamming distance is defined by the number of differences between the components of x and y .

stance, if one of the stored patterns is taken as the unknown vector, an overloaded network might yield a solution which resembles a combination of two or more other patterns in the set S (Fig. 12d). In other words, the network recalls a pattern which it was not taught. These so-called spurious states emerge from too many ‘coincidences’ in the weight matrix and cause additional local minima to be present in the energy function. Consequently, spurious states also are stable states of the Hopfield network. Because the number of coincidences grows rapidly as the number of patterns approximates or exceeds the theoretical value of $0.15N$, it is recommended that the number of patterns be kept well below this upper bound.

In conjunction with this, the noise tolerance level at which the network gives an error-free output is also determined by the number of patterns stored in the network. Returning to the example above, it appears that with four patterns at least a 25% noise level is allowed before errors occur during recall. Addition of a few new patterns to the weight matrix, in general, results in a substantial decrease of this noise tolerance level.

Spurious states also might arise when two or more of the stored patterns exhibit a high degree of correlation. The network is easily ‘confused’ if a noisy exemplar of one of these patterns is provided. In that case the noise in the unknown pattern highly determines which of the two patterns will be recalled. In order to prevent this problem, the desired pattern set, S , should be examined by means of a correlation procedure before the weight matrix is constructed. If it appears that two patterns exhibit a high correlation, then one of these patterns should be removed from S .

The Hopfield network operates on binary or, after a simple transformation, on bipolar * vectors. Such vectors are ideal to represent ‘black and white’ patterns. However, after a suitable transformation, many other object types can be stored in the network as well. As an example,

consider a set of range-scaled infrared spectra having absorbance values in the range 0.0–1.0. After choosing suitable wavelength intervals, the absorbances for each wavelength interval are represented by ‘sticks’ of, say, 8-bit binary or Gray encoded numbers [22]. Then, for each spectrum, concatenation of subsequent wavelength sticks results in a suitable binary input vector with dimension $8N_\lambda$, where N_λ denotes the number of wavelengths.

2.4. Continuous Hopfield network

2.4.1. Processing

A unit in the continuous Hopfield network possesses a sigmoidal transfer function (Fig. 11b)

$$o_i = g(\text{net}_i) = \frac{1}{1 + \exp(-2\lambda \text{net}_i)} \quad (16)$$

where λ , the gain factor, determines the steepness of the slope of the sigmoid function. In the high-gain limit, $\lambda \rightarrow \infty$, the transfer function approaches the threshold function given in Fig. 11a.

In contrast with the discrete Hopfield model, the input, net_i , is an implicit function of time and its dynamic behaviour is described by the differential equation

$$\frac{d}{dt} \text{net}_i = -\frac{1}{\tau} \text{net}_i = \sum_{j=1}^N W_{ij} o_j + I_i \quad (17)$$

For an elaborate discussion on this issue see ref. 15. The time constant, τ , can be interpreted as a ‘forgetting’ term, i.e., a small time constant ($\tau \approx 1$) implies that a new input state, $\text{net}_i(t+1)$, does not depend explicitly on the previous state, $\text{net}_i(t)$. The other terms are the same as in Eqn. 8. The additional term in the energy function, which originates from the nonlinear transfer function [14], is equal to

$$T_i(o_i) = \frac{1}{\lambda\tau} \sum_i \int_0^{o_i} g^{-1}(s) ds \quad (18)$$

In the high-gain limit, this term vanishes and the stable states of the Hopfield system lie on the vertices of the N -dimensional Hamming hypercube, as is the case for the discrete Hopfield model. For smaller values of λ , however, the

* A bipolar vector y has components which take the values -1 or 1 .

Fig. 13. The traveling salesperson problem. (a) This diagram shows a possible configuration of a five-city TSP. The five cities are labeled with the identifiers a, b, c, d, and e, respectively. The shortest possible tour is indicated by the thin lines. (b) Representation of the solution to the TSP. (c) The output of the Hopfield network for the solution given in (b).

stable states are located in the interior of the hypercube. In the most extreme case, $\lambda \rightarrow 0$, there exists exactly one stable state. For a finite gain factor, the continuous Hopfield network performs its search to the solution in a more sensitive and smooth fashion. This is not the case with the discrete Hopfield network which considers only the vertices of the hypercube as valid solutions.

2.4.2. Example of a continuous Hopfield network application

Of course, the continuous Hopfield network also behaves as an auto-associative memory. It can be used in a way similar to that which has been outlined for the discrete Hopfield network. The following example, however, focuses on a totally different application of the continuous Hopfield network, namely to solve a class of problems known as optimisation problems. For this purpose, the well known traveling salesperson problem (TSP) [4,15] will be discussed.

Briefly, in the TSP a salesperson has to visit a number of cities (see Fig. 13a), but each city only once, while minimising the total distance traveled. After the last city has been visited, the salesperson returns to the first city. Thus, the problem is to find the right sequence of cities (hereafter referred to as a tour) to visit.

Especially for a large number of cities this is a difficult problem to solve, even on modern computers. Suppose that we have N cities which must be placed in a certain order in a tour. Obviously, there are $N!$ possible solutions to this problem. However, it does not matter at which city the salesperson starts, reducing the number of solutions by a factor N . Furthermore, the direction in a given tour, clockwise or counter-clockwise, is irrelevant for the TSP, which reduces the problem size further by a factor 2. Hence, there exist $(N-1)!/2$ different solutions for an N -city TSP. As a consequence, the number of solutions grows exponentially with the size of the problem.

Now the question arises as to how this optimisation problem can be mapped onto the Hopfield network. Recall that the behaviour of the network is fully determined by its initial state, i.e., the output states, $o_i(t_0)$, and the differential equation (Eqn. 17). In this equation two terms appear which can be used to incorporate the TSP: the weights, W_{ij} , and the driver signals, I_i . To find how the weights and external signals must be chosen, one has to return to the network energy function, E . As has been demonstrated, the network always will settle in a local or global minimum of this function. Therefore, the TSP has to be incorporated in terms of constraints on the energy function. Favourable solutions (short tours) to the TSP have to correspond to deep minima in E , whereas worse solutions (long tours) correspond to more shallow minima.

Before constructing the energy function, an appropriate representation of the TSP has to be chosen. Considering the five-city configuration depicted in Fig. 13a, a possible solution to the TSP can be represented by a 5×5 matrix containing the binary numbers '0' and '1' (Fig. 13b). Each row in this matrix represents a city, e.g., the fourth row represents city 'd'. For each row, the column at which the '1' appears indicates a city position in the tour. In this example, city 'd' is located at the fifth position in the tour. The units in the Hopfield network are reordered accordingly (Fig. 13c). Henceforth, the unit outputs are labeled with two indices (o_{xi}), where the first index, x , refers to a city row and the second index, i , to a city-position column. Then, the

differential equation can be rewritten as

$$\frac{d}{dt} net_{xi} = -\frac{1}{\tau} net_{xi} + \sum_y \sum_j W_{xi,yj} o_{yj} + I_{xi} \quad (19)$$

where $W_{xi,yj}$ denotes the weight between the unit located at (x, i) and the unit at (y, j) . The other terms are self explanatory. It is stressed that only the subscript notation has been changed, not the network itself nor the weight matrix. Note that an N -city TSP problem requires N^2 units and thus an $N^2 \times N^2$ sized weight matrix.

The TSP is characterised by four constraints which need to be incorporated in the energy function. There are three strong constraints on E which concern the validity of the solution: (1) Energy minima must favour states that have each city only once on the tour. (2) Minima must favour states that have each position only once. (3) Minima must favour states that include precisely N cities. The fourth constraint is a weak one, and concerns the quality of the solution: (4) Energy minima must favour states corresponding to the shortest total distances in a tour.

An appropriate form of the network energy function can be found by considering the high-gain limit [14]. Consider the function:

$$\begin{aligned} E = & \frac{1}{2} A \sum_x \sum_i \sum_{j \neq i} o_{xi} o_{xj} \quad (\text{constraint 1}) \\ & + \frac{1}{2} B \sum_i \sum_x \sum_{y \neq x} o_{xi} o_{yi} \quad (\text{constraint 2}) \\ & + \frac{1}{2} C \left(\sum_x \sum_i o_{xi} - N \right)^2 \quad (\text{constraint 3}) \\ & + \frac{1}{2} D \sum_x \sum_{y \neq x} \sum_i d_{xy} o_{xi} (o_{yi+1} + o_{yi-1}) \\ & \quad (\text{constraint 4}) \quad (20) \end{aligned}$$

where A , B , C and D are positive constants. For notational convenience, the column indices are defined modulo N in order to express matrix boundary effects: $o_{xn+1} = o_{x1}$ and $o_{x0} = o_{xn}$. The first triple sum is zero if and only if each city row x contains no more than one '1', whereas the rest of the entries in that row are '0'. In other words, the first energy term is at a minimum when, for each row in the Hopfield network, no more than

one unit is turned 'on'. Similarly, the next energy term is zero if and only if each city-position column i contains no more than one '1'. Consequently, these two energy terms embody the first and the second TSP constraint explicitly. The third term is zero if and only if N units in the network are 'on'. The fourth term, sometimes called the data term, contains the necessary information needed to solve the TSP, i.e., the distances d_{xy} between the cities x and y . This term makes it likely that two cities with a short distance between them will appear next to each other in the tour. Once a valid solution has been found by the network, this term is numerically equal to the total length of the tour. The expression, $o_{xi}(o_{yi+1} + o_{yi-1})$, takes into account the fact that two cities can be incorporated in the tour in a clockwise or counter-clockwise fashion.

It can be proven [15] that any change in an output state, o_i , results in a decrease of the network energy. Moreover, the energy function has a lower bound because $E \geq DL_0$, where L_0 denotes the shortest possible length of a tour. So, indeed we have constructed a Lyapunov function and, hence, convergence to a (local) energy minimum is guaranteed.

By comparing the original Hopfield energy function

$$E = -\frac{1}{2} \sum_{i,j} \sum_{x,y} o_{xi} W_{xi,yj} o_{yj} - \sum_x \sum_i I_{xi} o_{xi} \quad (21)$$

to Eqn. 20 we can obtain expressions for the weights, $W_{xi,yj}$, and the external signals, I_{xi} . Comparison of the quadratic terms, $o_{xi} o_{yj}$, appearing in both equations reveals that

$$\begin{aligned} W_{xi,yj} = & -A \delta_{xy} (1 - \delta_{ij}) \quad (\text{row inhibition}) \\ & - B \delta_{ij} (1 - \delta_{xy}) \quad (\text{column inhibition}) \\ & - C \quad (\text{global inhibition}) \\ & - D d_{xy} (\delta_{ji+1} + \delta_{ji-1}) \quad (\text{data term}) \quad (22) \end{aligned}$$

where the Kronecker δ_{pq} is equal to 1 if $p = q$, and 0 otherwise. The first term concerns the mutual influence of units on one particular network row (δ_{xy}), and ensures that a unit inhibits the activity of all other units on its row but does

not suppress its own activity ($1 - \delta_{ij}$). So, during processing, the units in a single row are involved in a winner-takes-all competition. In the second term the δ functions are interchanged and therefore this term concerns the mutual inhibition in a column. The third term is a constant and each unit receives the same amount of global inhibition. This can be interpreted as a kind of self-regulatory mechanism which prevents too many units in the network from becoming activated simultaneously. The last term contains information about the distances between the cities. For a given column j , i.e., for the j th city-position on the tour, the terms δ_{ji+1} ('counter-clockwise' round trip) and δ_{ji-1} ('clockwise' round trip) are responsible for inhibitory weights between units in adjacent columns. The strength of the mutual inhibitory connection between two units is proportional to the distance between the corresponding two cities. Hence, this term imposes the condition that it is more likely that two adjacent cities appear next to each other in the tour (i.e., the units on the corresponding rows are turned 'on' in neighbouring columns) than two remote cities do.

Comparison of the remaining linear terms, o_{xi} , which appear in Eqns. 20 and 21 leads to

$$I_{xi} = CN \quad (23)$$

indicating that each unit in the network has to be driven by an equal constant external signal. When the constant, C , which appears in the weights, $W_{xi,yj}$, is adapted appropriately, the external signals, I_{xi} , even become superfluous.

This completes the design of the Hopfield neural network in terms of the TSP. The choice of the network parameters and the initial state highly influences the quality of the solution and the speed of convergence. Given a ten-city problem, Hopfield [15] provides a suitable parameter set: $A = B = D = 500$, $C = 200$, $\lambda = 50$, and $\tau = 1$. Concerning the initial state of the network, it would be reasonable to make all the outputs, o_i , at t_0 equal to the reciprocal of the number of cities, i.e., $1/N$. Then the total network output is given by

$$\sum_x \sum_i o_{xi} = N \quad (24)$$

Fig. 14. Network processing of the five-city TSP. Shown are three intermediate processing stages (a, b, c) and the final state (d) of the continuous Hopfield network which solved the TSP depicted in Fig. 13. The size of the black boxes is proportional to the unit output values o_{xi} . As can be seen, city c is the first city in the tour, followed by the cities e, b, and d. City a is the last city in the tour.

which equals the desired number of units that finally should be turned 'on'. Unfortunately, such an initial setting would paralyse the network. This situation is comparable to a ball placed on top of a hill; without intervention the ball would remain there forever. So, making all initial output states equal places the network on an unstable equilibrium point. To circumvent this, a small random value (β_i) must be added to each output state: $o_i = 1/N + \beta_i$. A possible outcome of the five-city TSP with the Hopfield network is depicted graphically in Fig. 14. Note that the network arrives at a sequence of cities (c, e, b, d, a) which differs from the optimal tour as shown in Fig. 13a.

2.4.3. Aspects of use of the continuous Hopfield network

Everything that has been mentioned previously about the aspects of using the discrete Hopfield network as an auto-associative memory applies also to the continuous Hopfield network. Therefore, this section describes the use of continuous Hopfield networks for solving optimisation problems.

The TSP has been discussed in detail to demonstrate both the power, complexity, and ap-

plicability of the continuous Hopfield network. This example showed that, for a given problem, finding an appropriate representation of the data and defining the constraints is the most difficult part in designing the Hopfield network. In general, this applies to all types of artificial neural networks. The choice of the network parameters is not straightforward. For instance, the parameter set provided for the 10-city TSP might be inadequate for solving a 200-city problem. Usually, parameter selection is done by a trial and error process, but fortunately, some rules of thumb are at one's disposal. In the TSP the parameters A and B concern similar constraints and thus, preferably are made equal: $A = B$. Next, while performing some test runs with the network, the parameter C has to be adjusted such that the desired number of units is turned 'on'. Thereafter, the parameter D can be chosen. Eventually, if necessary, the entire parameter optimisation process is repeated once or more.

Fig. 14d shows that the network reached a stable state which does not correspond to the tour with the shortest total distance. Obviously, the network settled in a local minimum of the energy function. This appears to be characteristic of the Hopfield network. Most times a good solution is found in a relatively short time span, whereas seldom the best one is obtained. Recalling that the Hopfield network is a deterministic model, the final state of the network is fully determined by the initial state. Therefore, we advise that at least a few runs be performed with different initial output state settings. In this way, perhaps deeper minima of the energy function will be encountered and, consequently, better solutions obtained.

Another strategy to circumvent shallow energy minima is to add a small amount of noise to the external signals, I_i . Of course, this procedure has to be repeated each time a processing cycle has been completed. In this way, the network might be pushed out of a shallow energy minimum and allowed to continue its search to a deeper one. Note that, due to the addition of the time-varying noise to the input signals, the behaviour of the Hopfield network will become stochastic instead of deterministic.

3. Short overview of neural networks

In the two parts of this Tutorial, three types of neural networks (MLF, Kohonen and Hopfield) were described. A variety of other networks has been developed. Some of these differ substantially with respect to their architecture and functionality from the above-mentioned networks, whereas others are closely related to the types of networks presented in this tutorial series. In this section a number of networks will be discussed briefly, more or less in their chronological order of appearance.

3.1. Perceptron

The perceptron [23,24] is the first precisely specified, computationally oriented neural network. It may be regarded as the predecessor of the current artificial neural networks. With respect to its functionality, the perceptron is only suited for classification problems in which the classes are linearly separable. The network is trained in a supervised manner and the weights are updated if and only if the network makes a wrong decision.

3.2. Adaptive linear element (ADALINE)

ADALINE [25] is also one of the earlier networks. The former name was adaptive linear neuron, but 'neuron' was replaced by 'element' when the neural networks fell into disgrace. ADALINE is in many aspects similar to the perceptron. The network is trained under supervision and uses a principle on which the back-propagation learning rule is based, i.e., the weights in the network are updated by small steps which are proportional to the difference between the desired and the actual solution (the Widrow–Hoff rule). Neither the perceptron nor the ADALINE network are used much at present.

3.3. Brain state in a box (BSB)

BSB [26] is a linear auto-associative network which closely resembles the Hopfield network. However, it is usually not fully interconnected

and the weight matrix is not required to be symmetric. A non-linear post-processing algorithm is invoked to deal with spurious network states [27]. Auto-associative networks like BSB, bidirectional associative memory (BAM) and Hopfield are used to fill in missing information or applied as noise-filters. A possible application is described in ref. 28. This paper describes how the BSB network is used to store diseases and their treatments. Each input vector is a concatenation of a description of a disease, its cause, and the treatment. If an incomplete input vector is presented to the network, e.g., a disease only, the vector is restored and the network comes up with the cause and the treatment. Unfortunately, the BSB network appeared to perform far from perfectly [27].

3.4. Hopfield network

The first paper in which Hopfield describes this network [13] triggered renewed interest in artificial neural networks. Many of the ideas presented in that paper were not new, as Hopfield acknowledged, but the main concepts were combined and analysed mathematically in such a detailed way that this type of network is named after Hopfield. For a detailed description of this network the reader is referred to Section 2.

3.5. Kohonen network

The Kohonen network [1] is a self-organising feature map which is trained in an unsupervised manner. It maps vectors from a multi-dimensional space onto a low-dimensional space, thereby preserving the topology of the data set as well as possible. See Section 1 for a detailed description of the Kohonen network.

3.6. Boltzmann machine

The structure and dynamics of the Boltzmann machine [29] are quite similar to that of the Hopfield network, but it has a probabilistic updating rule which is based on some principles of thermodynamics. The Boltzmann network searches for the minimum of its associated energy function and its performance may be seriously

hampered by getting stuck in local minima. To escape from these, it incorporates a technique known as simulated annealing. Much 'thermal' noise is added in the beginning of the search and this noise is gradually decreased as the search continues. Simulated annealing may also be used for other network structures like MLF networks.

3.7. Multi-layer feed-forward neural networks (MLF)

Multi-layer architectures were described at an early date [29] but they became popular when a learning algorithm was described to train MLF networks [30]. This algorithm, the back-propagation learning rule, is in fact a generalisation of the Widrow–Hoff rule. See Part I for a description of MLF networks and the back-propagation learning rule.

3.8. Bidirectional associative memory (BAM)

The BAM network [20,21] is similar to the Hopfield network but it can be applied as a hetero-associative network as well. There even exists an adaptive version (ABAM) in which the weights are trained, as in MLF networks, instead of being calculated in advance.

3.9. Learning vector quantisation (LVQ)

LVQ is a supervised extension of the Kohonen algorithm [1] and it is usually applied for classification problems. LVQ may also be used to refine a trained Kohonen self-organising feature map to improve its performance when circumstances are changing, e.g., when new objects have to be added to the data base with which the Kohonen network was trained.

3.10. Counter-propagation network (CPN)

In the CPN [31] different types of processing layers are combined, with each layer having its own kind of functionality. It operates as a look-up table capable of generalising. The CPN associates input vectors with output vectors and acts comparably to the MLF network. However, the CPN is

not as general as the MLF. Because of its typical architecture, the CPN requires much less time than MLF networks to achieve convergence of the weights. This network may be applied for rapid prototyping, with perhaps the MLF network as the final choice for the user-implementation, or used for those problems where a quick approximation of the solution is of vital importance.

3.11. Adaptive resonance theory (ART)

Several versions of the ART network have been developed [32,33]: e.g., ART1 which operates on binary input vectors and ART2 which can cope with continuous inputs. Initially, the ART network stores a rather limited number of exemplar patterns which correspond to a few distinct classes. A new pattern is checked against the best matching exemplar pattern of this set. If the resemblance, which is expressed by means of a similarity measure, of the two patterns is below a certain threshold, the pattern is defined to belong to a new class and the network is expanded with this new class. The ART network is inspired by biological neural networks, in which the synapses, c.f. the weights, are being modified permanently. According to this so-called plasticity principle, biological networks adapt their weights in order to deal with changing circumstances or to perform new tasks. Weight adaptations are made without destroying the previously stored information.

Bibliography

For more information on neural networks, the reader is referred to the following books:

J.A. Anderson and E. Rosenfeld (Editors), *Neurocomputing: Foundations of Research*, MIT Press, Cambridge, MA, 1988.

R. Beale and T. Jackson, *Neural Computing, an Introduction*, Adam Hilger, IOP Publishing, Bristol, 1990.

J.A. Freeman and D.M. Skapura, *Neural Networks: Algorithms, Applications, and Programming Techniques*, Addison-Wesley, Reading, MA, 1991.

R. Hecht-Nielsen, *Neurocomputing*, Addison-Wesley, Reading, MA, 1990.

T. Kohonen, *Self-Organization and Associative Memory*, Springer Verlag, Berlin, 1984.

D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*, Vols. 1 and 2, MIT Press, London, 1986.

P.D. Wasserman, *Neural Computing, Theory and Practice*, Van Nostrand-Reinhold, New York, 1989.

References

- [1] T. Kohonen, *Self-Organization and Associative Memory*, Springer Verlag, Berlin, 1984.
- [2] J.R.M. Smits, W.J. Melssen, L.M.C. Buydens and G. Kateman, Using artificial neural networks for solving chemical problems. Part I. Multi-layer feed-forward networks *Chemometrics and Intelligent Laboratory Systems*, 22 (1994) 165–189.
- [3] T. Kohonen, Self-organized formation of topologically correct feature maps, *Biological Cybernetics*, 43 (1982) 59–69.
- [4] J.A. Freeman and D.M. Skapura, *Neural Networks: Algorithms, Applications, and Programming Techniques*, Addison-Wesley, Reading, MA, 1991.
- [5] R. Hecht-Nielsen, *Neurocomputing*, Addison-Wesley, Reading, MA, 1990.
- [6] P.D. Wasserman, *Neural Computing, Theory and Practice*, Van Nostrand-Reinhold, New York, 1989.
- [7] H. Ritter and K. Schulten, Convergence properties of Kohonen's topology conserving maps: fluctuations, stability, and dimensions selection, *Biological Cybernetics*, 60 (1988) 59–71.
- [8] D. Desieno, Adding a conscience to competitive learning, *Proceedings of the International Conference on Neural Networks*, I (1988) 117–124.
- [9] J. Zupan and J. Gasteiger, Neural networks: A new method for solving chemical problems or just a passing phase?, *Analytica Chimica Acta*, 248 (1991) 1–30.
- [10] H. Ritter and K. Schulten, Extending Kohonen's self-organizing feature mapping algorithm to learn ballistic movements, in R. Eckmiller and C. v.d. Malsburg (Editors), *Neural Computers*, Springer-Verlag, Heidelberg, 1987, pp. 393–406.
- [11] W.J. Melssen, J.R.M. Smits, G.H. Rolf and G. Kateman, Two-dimensional mapping of IR spectra using a parallel implemented self-organising feature map, *Chemometrics and Intelligent Laboratory Systems*, 18 (1993) 195–204.
- [12] M. Novic and J. Zupan, 2-D mapping of infrared spectra using Kohonen neural network, *Vest. Slov. Kem. Drus.*, submitted for publication.

- [13] J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, *Proceedings of the National Academy of Science of the USA*, 79 (1982) 2554–2558.
- [14] J.J. Hopfield, Neurons with graded response have collective computational properties like those of two-state neurons, *Proceedings of the National Academy of Science of the USA*, 81 (1984) 3088–3092.
- [15] J.J. Hopfield and D.W. Tank, Neural computation of decisions in optimization problems, *Biological Cybernetics*, 52 (1985) 141–152.
- [16] J.J. Hopfield and D.W. Tank, Computing with neural circuits: a model, *Science*, 233 (1986) 625–633.
- [17] D.W. Tank and J.J. Hopfield, Collective computation in neuronlike circuits, *Scientific American*, 257(6) (1987) 104–114.
- [18] M. Otto and U. Höchner, Application of fuzzy neural network to spectrum identification, in J. Gasteiger (Editor), *Software Development in Chemistry*, Vol. 4, Springer-Verlag, Berlin, 1990.
- [19] Y. Takefuji, C.W. Lin and K.C. Lee, A parallel algorithm for estimating the secondary structure in ribonucleic acids, *Biological Cybernetics*, 63 (1990) 337–340.
- [20] B. Kosko, Constructing an associative memory, *Byte*, 12(10) (1987) 137–144.
- [21] B. Kosko, Bidirectional associative memories, *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1) (1988) 49–60.
- [22] C.B. Lucasius and G. Kateman, Understanding and using genetic algorithms. Part 2. Representation, configuration and hybridization, *Chemometrics and Intelligent Laboratory Systems*, in press.
- [23] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organisation in the brain, *Psychological Review*, 65 (1958) 386–408.
- [24] M.L. Minsky and S.A. Papert, *Perceptrons, an Introduction to Computational Geometry, Expanded Edition*, MIT Press, Cambridge, MA, 1988.
- [25] B. Widrow and M.E. Hoff, *Adaptive Switching Circuits*, in IRE WESCON Convention Record, August 23–26, New York (1960), pp. 96–104.
- [26] J.A. Anderson, Neural models with cognitive implications in D. LaBerge and S.J. Samuels (Editors), *Basic Processes in Reading Perception and Comprehension*, Erlbaum, Hillsdale, NJ, 1977.
- [27] *Manual NeuralWorks Professional IItm*, NeuralWare Inc, Sewickley, PA, USA.
- [28] J.A. Anderson, K. Murphy and L. Gregory, Psychological concepts in a parallel system, *Physica D*, 22 (1986) 318–336.
- [29] D.H. Ackley, G.E. Hinton and T.J. Sejnowski, A learning algorithm for Boltzmann machines, *Cognitive Science*, 9 (1985) 147–169.
- [30] D.E. Rumelhart, G.E. Hinton and R.J. Williams, Learning representations by back-propagating errors, *Nature*, 323 (1986) 533–536.
- [31] R. Hecht-Nielsen, Counterpropagation networks, in M. Caudill and C. Butler (Editors), *Proceedings of the IEEE First International Conference on Neural Networks*, Vol. 2, SOS Printing, San Diego, CA, 1987, pp. 19–32.
- [32] S. Grossberg (Editor), *The Adaptive Brain*, Vols. 1 and 2, Elsevier, Amsterdam, 1987.
- [33] G.A. Carpenter and S. Grossberg, The ART of adaptive pattern recognition by a self-organizing neural network, *IEEE Computer*, 21(3) (1988) 77–88.