Acta Chimica Slovenica 41/3/1994, pp. 327-352

Introduction to Artificial Neural Network (ANN) Methods: What They Are and How to Use Them*.

Jure Zupan[1),
Department of Chemistry, University Rovira i Virgili,
Tarragona, Spain

Basic concepts of ANNs together with three most widely used ANN learning strategies (error back-propagation, Kohonen, and counter-propagation) are explained and discussed. In order to show how the explained methods can be applied to chemical problems, one simple example, the classification and the prediction of the origin of different olive oil samples, each represented by eigtht fatty acid concentrations, is worked out in detail.

Introduction

In the last few decades, as the chemists have get accustomed to the use of computers and consequently to the implementation of different complex statistical methods, they are trying to explore multi-variate correlations between the output and input variables more and more in detail. With the increasing accuracy and precision of analytical measuring methods it become clear that all effects that are of interest cannot be described by simple uni-variate and even not by the linear multi-variate correlations precise, a set of methods, that have recently found very intensive use among chemists are the artificial neural networks (or ANNs for short).

*) The lecture presented at the VI-th COMETT Italian School on Chemometrics, Alghero, Sardinia, Italy, 26-30-st September 1994.

[1)] On leave from the National Institute of Chemistry, Ljubljana, Slovenia

Therefore, the analytical chemists are always eager to try all new methods that are available to solve such problems. One of the methods, or to say more Due to the fact that this is not one, but several different methods featuring a wide variety of different architectures learning strategies and applications, we shall first start with explaining the overall strategy, goals, implications, advantages and disadvantages, and only after explaining that, we shall discuss the fundamental aspects of different approaches to these methods and how they can be put to use in analytical chemistry.

The ANNs are difficult to describe with a simple definition. Maybe the closest description would be a comparison with a black box having multiple input and multiple output which operates using a large number of mostly parallel connected simple arithmetic units. The most important thing to remember about all ANN methods is that they work **best** if they are dealing with **non-linear dependence between the inputs and outputs** (Figure 1).

ANNs can be employed to describe or to find linear relationship as well, but the final result might often be worse than that if using another simpler standard statistical techniques. Due to the fact that at the beginning of experiments we often do not know whether the responses are related to the inputs in a linear on in a non-linear way, a good advise is to try always some standard statistical technique for interpreting the data parallel to the use of ANNs.

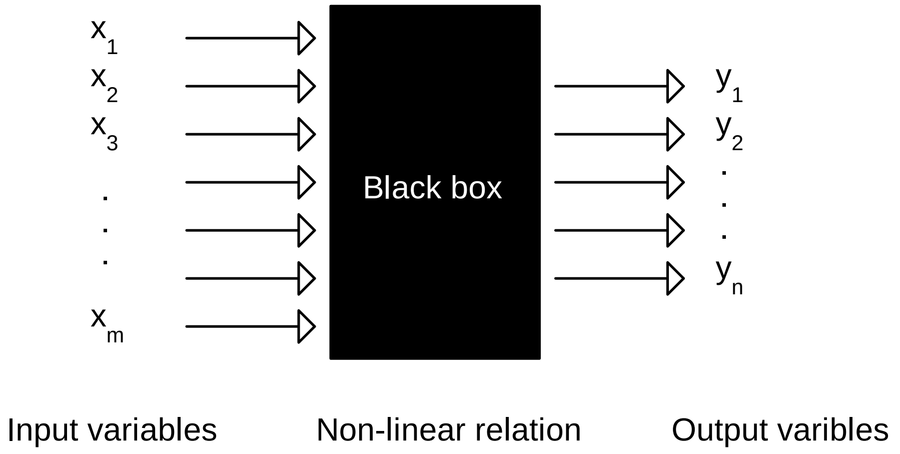


Figure 1. Neural network as a black-box featuring the non-linear relationship between the multi-variate input variables and multi-variate responses

Problem consideration

The first thing to be aware of in our consideration of employing the ANNs is the nature of the problem we are trying to solve: does the problem require a supervised or an unsupervised approach. The supervised problem means that the chemist has already a set of experiments with **known outcomes for specific inputs** at hand, while the unsupervised problem means that one deals with a set of experimental data which have no specific associated answers (or supplemental information) attached. Typically, unsupervised problems arise when the chemist has to explore the experimental data collected during pollution monitoring, or always he or she is faced with the measured data at the first time or if one must find a good method to display the data in a most appropriate way.

Usually, first problems associated with handling data require unsupervised methods. Only further after we became more familiar with the measurement space (the measurement regions) of input and output variables and with the behaviours of the responses, we can select sets of data on which the supervised methods (modelling for example) can be carried on.

The basic types of goals or problems in analytical chemistry for solution of which the ANNs can be used are the following:

- election of samples from a large quantity of the existing ones for further andling,
- classification of an unknown sample into a class out of several pre-defined (known in advance) number of existing classes,
- clustering of objects, i.e., finding the inner structure of the measurement space to which the samples belong, and
- making direct and inverse models for predicting behaviours or effects of unknown samples in a quantitative manner.

Once we have decided which type of the problem we have, we can look for the best strategy or method to solve it. Of, course in any of the above aspects we can employ one or more different ANN architectures and different ANN learning strategies.

Basic concepts of ANNs

Now we will briefly discuss the basic concepts of ANNs. It is wise to keep in mind that in the phrase 'neural network' the emphasise is on the word 'network' rather than on the word 'neural'. The meaning of this remark is that the way how the 'artificial neurons' are connected or networked together is much more important than the way how each neuron performs its simple operation for which it is designed for.

Artificial neuron is supposed to mimic the action of a biological neuron, i.e., to accept many different signals, x_i, from many neighbouring neurons and to process them in a pre-defined simple way. Depending on the outcome of this processing, the neuron j decides either to fire an output signal y_j or not. The output signal (if it is triggered) can be either 0 or 1, or can have any real value between 0 and 1 (Fig. 2) depending on whether we are dealing with 'binary' or with 'real valued' artificial neurons, respectively.

Mainly from the historical point of view the function which calculates the output from the m-dimensional input vector **X**, f(**X**), is regarded as being composed of **two** parts. The first part evaluates the so called 'net input', Net, while the second one 'transfers' the net input Net in a **non-linear** manner to the output value y.

The first function is a linear combination of the input variables, x_1, x_2, ... x_i, ... x_m, multiplied with the coefficients, w_{ji}, called 'weights', while the second function serves as a 'transfer function' because it 'transfers' the signal(s) through the neuron's axon to the other neurons' dendrites. Here, we shall show now how the output, y_j, on the j-th neuron is calculated. First, the net input is calculated according to equation /1/:

$$Net_j = \sum_{i=1}^{m} w_{ji} x_i \qquad /1/$$

and next, Net_j, is put as the argument into the transfer function /2/:

$$y_j = out_j = 1/\{1 + \exp[-\alpha_j(Net_j + \theta_j)]\} \qquad /2/$$

The weights w_{ji} in the artificial neurons are the analogues to the real neural synapse strengths between the axons firing the signals and the dendrites receiving those signals (see Figure 2). Each synapse strength between an axon and a dendrite (and, therefore, each weight) decides **what proportion of the incoming signal is transmitted into the neurons body**.

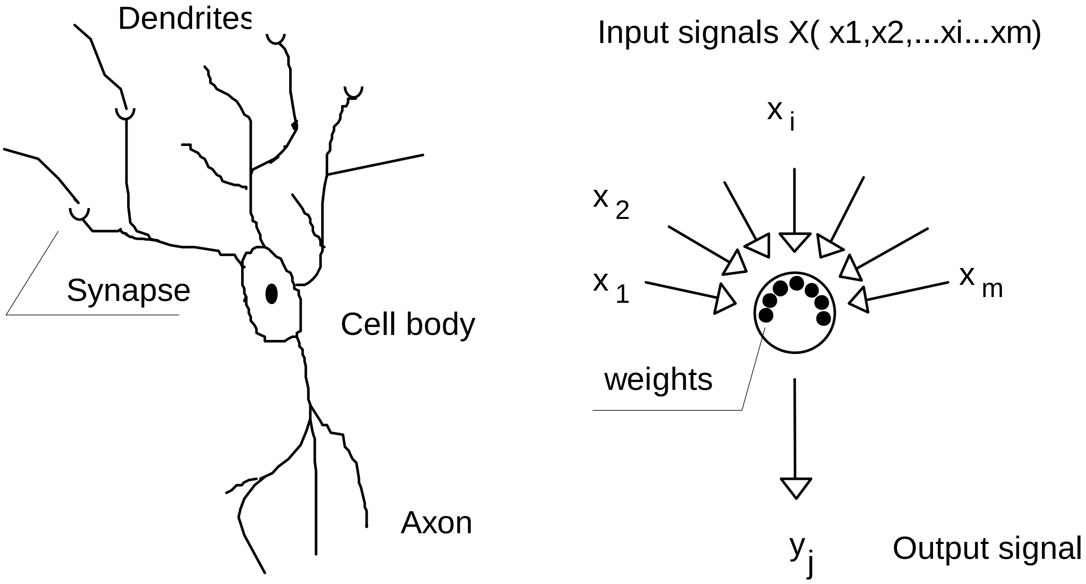


Figure 2. Comparison between the biological and artificial neuron. The circle mimicking the neuron's cell body represents simple mathematical procedure that makes one output signal y_j from the set input signals represented by the multi-variate vector X.

It is believed (assumed) that the 'knowledge' in the brain is gained by constant adapting of synapses to different input signals causing better and better output signals, i.e., to such signals that would cause proper reactions or answers of the body. The results are constantly feed-back as new inputs. Analogous to the real brain, the artificial neurons try to mimic the adaption of synapse strengths by iterative adaption of weights w_{ji} in neurons according to the differences between the actually obtained outputs y_j and desired answers or targets t_j.

If the output function f(Net$_j$) is a binary threshold function (Fig. 3a), the output, y_j, of one neuron has only two values: zero or one. However, in most cases of the ANN approach the networks of neurons are composed of neurons having the transfer function of a **sigmoidal** shape (eq. /2/, Fig. 3b). In some cases the transfer function in the artificial neurons can have a so called radial form:

$$y_j = out_j = e^{-[\alpha_j(Net_j+\theta_j)]^2} \qquad /3/$$

Some possible forms for the transfer function are plotted in Figure 3. It is important to understand that the **form** of the transfer function, once it is chosen, is used for **all** neurons in the network, regardless of where they are placed or how they are connected with other neurons. What changes during the learning or training is **not** the function, but the weights and the function parameters that control the **position** of the threshold value, θ_j , and the **slope** of the transfer function α_j. (eqs. /2/, /3/).

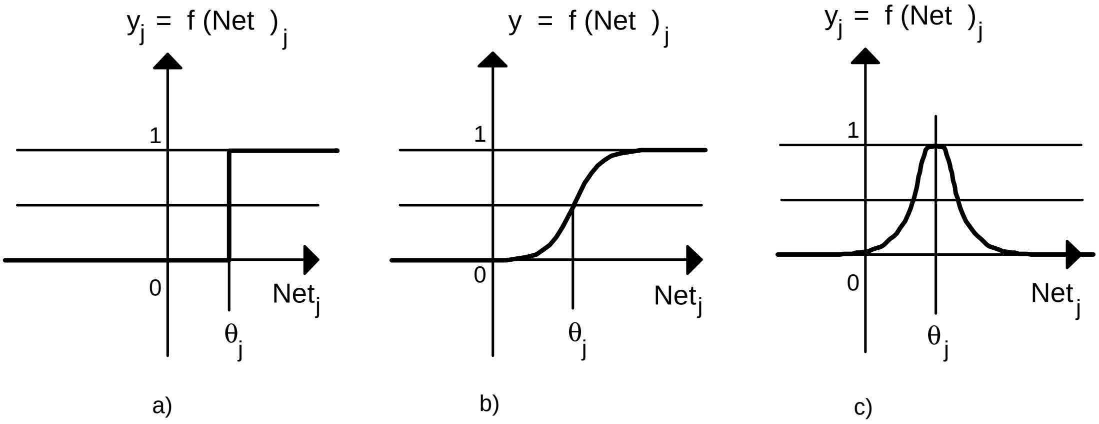


Figure 3. Three different transfer functions: a threshold (a), a sigmoidal (b) and a radial function (c). The parameter θ_j in all three functions decides the Net_j value around which the neuron is most selective. The other parameter, α_j seen in equations /1/ and /2/ affects the **slope** of the transfer function (not applicable in case a) in the neighbourhood of θ_j.

It can be shown that the two mentioned parameters, αj and θj, can be treated (corrected in the learning procedure) in all ANN having this kind of transfer function in an exactly the same way as the weights w_{ji}. Let us look, why.

First, we shall write the argument u_j of the function $f(u_j)$ (eq. /3/) in an extended form by including eq. /1!:

$$u_j = \alpha_j(Net_j + \theta_j) = \sum_{i=1}^{m} \alpha_j w_{ji} x_i + \alpha_j \theta_j \qquad /4/$$

$$= \alpha_j w_{j1} x_1 + \alpha_j w_{j2}\, x_2 + ... + \alpha_j w_{ji} x_i + ... + \alpha_j w_{jm} x_m + \alpha_j \theta_j \qquad /5/$$

At the beginning of the learning process none of the constants, α_j, w_{ji}, or θ_j are known, hence, it does not matter if the products like $\alpha_j w_{ji}$ and $\alpha_j\theta_j$ are written simply as **one** constant value or as a product of **two** constants. So, by simply writing the products of two unknowns as only one unknown parameter we obtain equation /5/ in the form:

$$u_j = w_{j1}x_1 + w_{j2}x_2 +...+ w_{ji}x_i +...+ w_{jm}x_m + w_{jm+1} \qquad /6/$$

Now, by **adding** one variable x_{m+1} to all input vectors X we will obtain the so called augmented vector X' = (x_1,x_2, ... x_i ... x_m, x_{m+1}). After this additional variable is set to 1 in all cases without an exception we can write all augmented input vectors X' as (x_1,x_2, ... x_i ... x_m, 1).

The reason why this manipulation is made is simple. We want to incorporate the last weight w_{jm+1} originating from the product $\alpha_j\theta_j$ into a single new Net_j term containing **all** parameters (weights, threshold, and slope) for adapting (learning) the neuron in the same form. The only thing needed to obtain and to use equation /7/ is to add one component equal to 1 to each input vector.

This additional input variable which is always equal 1 is called **'bias'**. Bias makes the neurons much more flexible and adaptable. The equation for obtaining the output signal from a given neuron is now derived by combining equation /6/ with either equation /2/ or /3/ - depending on our preference for the form of the transfer function:

$$y_j = out_j = 1/[1 + \exp(-Net_j)] \qquad /2a/$$

$$y_j = out_j = \exp(-Net_j{}^2) \qquad /3a/$$

with

$$Net_j = \sum_{i=1}^{m+1} w_{ji}\, x_i \qquad /1a/$$

Note that the only difference between eq. /1/ and /1a/ is the summation in the later being performed from 1 to m+1 instead from 1 to m as in the former.

Now, as we gain some familiarity about the artificial neurons and how they manipulate the input signals, we can inspect large ensembles of neurons. Large ensembles of neurons in which most of the neurons are interconnected are called neural networks. If single neurons are simulated by the computers then the ensembles of neurons can be simulated as well. the ensembles of neurons simulated on computers are called Artificial Neural Networks (ANNs).

Artificial neural networks (ANNs) can be composed of different number of neurons. In chemical applications, the sizes of ANNs, i.e., the number of neurons, are ranging from tens of thousands to only as little as less than ten (1-3). The neurons in ANNs can be all put into one layer or two, three or even more layers of neurons can be formed. Figure 4 show us the difference between the one and multi-layer ANN structure.

In Figure 4 the one-layer network has four neurons (sometimes called nodes), each having four weights. Altogether there are 16 weights in this one-layer ANN. Each of four neurons accept all input signals plus the additional input from the bias which is always equal to one. The fact, that the input is equal to 1, however, does not prevent the weights leading from the bias towards the nodes to be changed!

The two-layer ANN (Fig. 4, right) has six neurons (nodes): two in the first layer and four in the second or output layer. Again, all neurons in one layer obtain **all** signals that are coming from the layer above. The two-layer network has (4 x 2) + (3 x 4) = 20 weights: 8 in the first and 12 in the second layer. It is understood that the input signals are normalised between 0 and 1.

In Figure 4 the neurons are drawn as circles and the weights as lines connecting the circles (nodes) in different layers. As can be seen, the lines (weights) between the input variables and the first layer have no nodes or neurons marked at the top. To overcome this inconsistency the input is regarded as a **non active** layer of neurons serving only to distribute the signals to the first layer of **active** neurons.

Therefore, Figure 4 shows actually a 2-layer and a 3-layer networks with the input layer being inactive. The reader should be careful when reading the literature on ANNs because authors sometimes actually refer to the above ANNs as to the two- and three-layer ones. We shall regard only the active layer of neurons as actual layer and will therefore name this networks as one and two-layer ANNs.

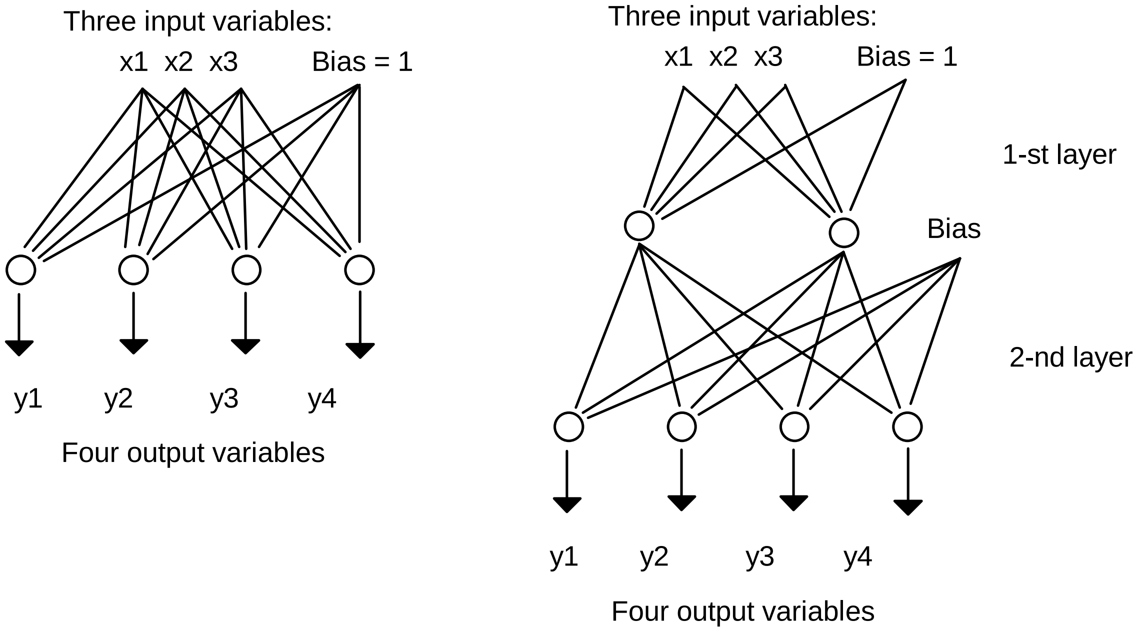


Figure 4. One-layer (left) and two-layer (right) ANNs. The ANNs shown can be applied to solve a 3-variable input 4-responses output problem.

The two-layer network is probably the most used ANN design for chemical applications. It can be safely said that more than 75% of all ANN applications in chemistry are made using this type of network architecture - of course with different number of nodes in each level (1,2). Due to the fact that there are different learning techniques (or training procedures) in ANN methodology, another type of network is often used in the so called Kohonen network. This network will be explained in the next paragraph when learning techniques will be discussed.

Selection of the training set is usually the first task the user has to do when trying to apply any learning method for classical modelling, for pattern recognition, for expert system learning, or for ANNs. It has to be kept in mind that for any learning procedure the best strategy is to divide the available data not on **two,** but on **three** data sets! The first one is the **training** set, the second the **control** (or fine tuning set) and the third one is the final **test** set. Each of these three data sets should contain approximately one third of data with a tendency of the training set to be the smallest and the test set to be the largest one. The control set which is most often ignored is used as a supplier of feed-back information for correcting the initially selected ANN architecture in the cases when the training is performing good, but the results on the control set which was, of course, not used in the training, are bad. Unfortunately, this happens very often in any learning procedure. The users normally leave this task to the test set, but according to the philosophy of learning and testing, the test objects should in no way be involved in obtaining the model, hence, the true test should be executed with a completely 'noncommittal' or unbiased set of data, what the objects in the control set are not!

Learning by ANN

Error Back-Propagation (4,5) normally requires at least two layers of neurons: the hidden layer and the output layer. The input layer is non active and is therefore not counted in the scheme. The learning with the error back-propagation method requires a supervised problem, i.e., we must have a set of r input and target pairs $\{X_S,T_S\}$. In other words, for the training we must have a set of m-variable input objects X_S (m-intensity spectrum, m-component analysis, m-consecutive readings of a time dependent variable, etc.,) and with each X_S, the corresponding n-dimensional target (response) T_S should associated (n structural fragments). The error back-propagation method has obtained its name due to its learning procedure: the weights of neurons are first corrected in the output layer, then in the second hidden layer (if there is one), and at the end in the first hidden layer, i.e., in the first layer that obtains the signals directly from the input.

The reason behind this order of weight correction is simple: the exact error which is made by the ANN is known only in the output layer. After the n output layer neurons fire their outputs y_i, which can be regarded as n-dimensional output vector Y ($y_1, y_2, .. y_i, ... y_n$), these answers are compared to the target values t_i of the target vector T_S that accompanies the input vector X_S. In this way, the error $\delta\varepsilon_i$ on each output node i is known exactly.

$$\delta\varepsilon_i = y_i - t_i \qquad /7/$$

It is outside the scope of this chapter to explain in detail how the formulas for the correction of weights are derived from the first principles, i.e., from the so called δ-rule and by the gradient descent method. The interested reader can consult any introductory book on the subject (3,4). In Figure 5 the order in which the corrections of weights are made in the error back-propagation learning is shown

First corrected are the weights in the output layer and after having this weights corrected together with the assumption that the errors have been evenly distributed when the signals were passing from the last hidden to the output layer, the weights in the last hidden layer are corrected and so on. The last weights to be corrected are the weights in the upper layer. In the same way (except for the output layer - see eq. /8/, /9/ and /10/) the weights in all additional layers are calculated in the case that the ANN has more than three layers - which is unusual.

The learning algorithm in error back-propagation functions much better if the so called 'momentum' term is added to the standard term for weights correction. The final expressions according which the weights are calculated are as follows:

$$\Delta w_{ji}^{\ell} = \eta \delta_j^{\ell} y_i^{\ell-1} + \mu \Delta w_{ji}^{\ell(\text{previous})} \qquad /8/$$

The parameters η and μ are called learning rate and momentum and are mainly between 0.1 and 0.9. Keeping in mind that the output of the $\ell-1$-st layer is equal to the input to the ℓ-th layer (Fig. 5), we can see that the term $y_i^{\ell-1}$ is equivalent to the term inp_i^{ℓ}. When correcting the first layer of weights (ℓ = 1) this variables are equal to the components x_1, x_2, ... x_m of the input vector X.

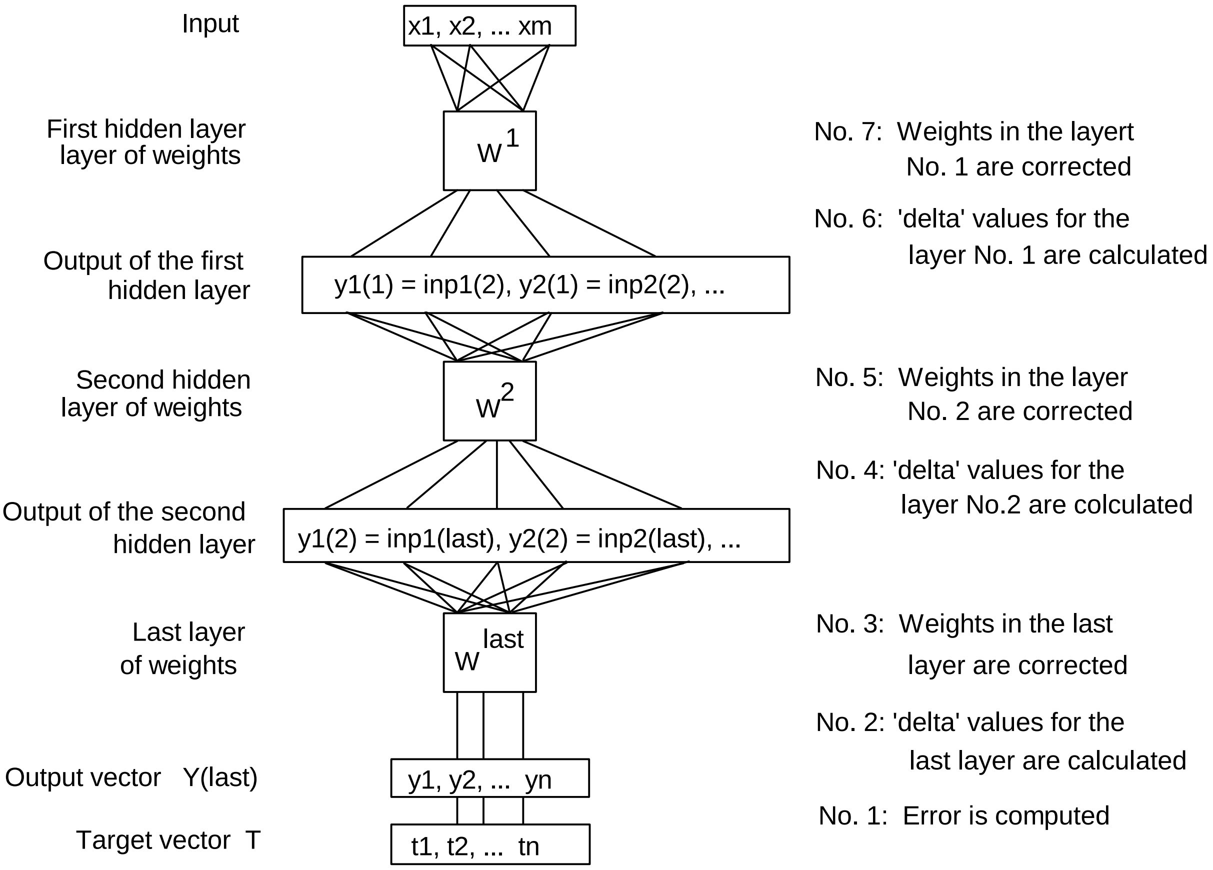

Input
x1, x2, ... xm
First hidden layer layer of weights
W1
No. 7: Weights in the layert No. 1 are corrected
No. 6: 'delta' values for the layer No. 1 are calculated
Output of the first hidden layer
y1(1) = inp1(2), y2(1) = inp2(2), ...
Second hidden layer of weights
W2
No. 5: Weights in the layer No. 2 are corrected
No. 4: 'delta' values for the layer No.2 are colculated
Output of the second hidden layer
y1(2) = inp1(last), y2(2) = inp2(last), ...
Last layer of weights
W last
No. 3: Weights in the last layer are corrected
No. 2: 'delta' values for the last layer are calculated
Output vector Y(last)
y1, y2, ... yn
No. 1: Error is computed
Target vector T
t1, t2, ... tn

Figure 5. Order for correction of weights in the error back-propagation learning is 'bottom-top' oriented. The i-th output of the k-th layer is marked as yi(k). The same is true for inputs.

The superscript 'previous' indicates that for proper correction of weights besides taking into the account errors it is necessary to store all weights from the previous correction and use them for additional correction. The most interesting is, of course, the δ_j^{ℓ} term. It explains the error that occurs on the j-th neuron (node) of the ℓ-th layer. The δ_j^{ℓ} terms for the calculation of weight corrections (eq. /8/) are different for the output layer (ℓ = $last$ or output) compared to all the δ_j^{ℓ} terms used in the corrections of weights in hidden layers (ℓ = 1 to $last-1$):

$$\delta_j^{last} = (t_j - y_j^{last})\, y_j^{last}\, (1-y_j^{last}) \qquad /9/$$

$$\delta_j^{\ell} = \left(\sum_{k=1}^{r} \delta_j^{\ell+1}\ w_{kj}^{\ell+1} \right)\ y_j^{\ell}\ (1 - y_j^{\ell}) \qquad \text{for all } \ell = 1 ... last-1 \qquad /10/$$

The summation in eq. /10/ involves all r $\delta^{\ell+1}$-terms originating in the neuron layer $\ell+1$, i.e., one level **below** the level ℓ. Now, it becomes clear why we need to go in the backward direction to correct the weights. The one last thing worth mentioning is, that as a consequence of the δ-rule principle (4), in the correction of each weight in a given layer ℓ, three layers of neurons are involved: the weights of the layer ℓ, the input to level ℓ from the layer $\ell-1$, and the sum of $\delta_j^{\ell+1}\ w_{kj}^{\ell+1}$ products from the layer $\ell+1$ bellow the level ℓ.

Learning by error back-propagation (like in all supervised methods) is carried out in cycles, called 'epochs'. One epoch is a period in which all input-target pairs $\{X_S,T_S\}$ are presented once to the network. The weights are corrected after each input-target pair (X_S,T_S) produces an output vector Y_S and the errors from all output components y_i are squared and summed together. After each epoch, the RMS (root-mean-square) error is reported:

$$\mathrm{RMS} = \left(\frac{\sum_{s=1}^{r}\sum_{i=1}^{n} (t_{si} - y_{si})^2}{r\,n} \right)^{1/2} \qquad /11/$$

The RMS value measures how good the outputs Y_S (produced for the entire set of r input vectors X_S) are in comparison with all r n-dimensional target values T_S. The aim of the supervised training, of course, is to reach as small RMS value as possible in a shortest possible time. It should be mentioned that when using the error back-propagation learning to solve some problems may require days if not weeks of computation time even on the super-computers. However, smaller tasks involving less than ten variables on input and/or on the output side, require much less computational effort and can be solved on personal computers.

The Kohonen ANN (6,7) offers considerably different approach to ANNs than the error back-propagation method. The main reason is that the Kohonen ANN is a 'self-organising' system which is capable to solve the unsupervised rather than the supervised problems. In unsupervised problems (like clustering) it is not necessary to know in advance to which cluster or group the training objects X_S belongs. The Kohonen ANN automatically adapts itself in such a way that the similar input objects are associated with the **topological** close neurons in the ANN. The phrase 'topological close neurons' means that neurons that are physically located close to each other will react similar to similar inputs, while the neurons that are far apart in the lay-out of the ANN react quite different to similar inputs.

The most important difference the novice in the filed of ANN has to remember is that the neurons in the error back propagation learning tries to yield **quantitatively** an answer as close as possible to the target, while in the Kohonen approach the neurons learn to pin-point the **location** of the neuron in the ANN that is most 'similar' to the input vector X_S. Here, the phrase 'location of the most similar neuron' has to be taken in a very broad sense. It can mean the location of the closest neuron with the smallest or with the largest Euclidean distance to the input object X_S, or it can mean the neuron with the largest output in the entire network for this particular input vector X_S, etc. With other words, in the Kohonen ANNs a **rule** deciding which of all neurons will be selected after the input vector X_S enters the ANN is mandatory.

Because the Kohonen ANN has only one layer of neurons the specific input variable, let us say the i-th variable, x_i, is always received in **all** neurones of the ANN, by the weight placed at the i-th position. If the neurons are presented as columns of weights rather than as circles and connecting lines (cf. Fig. 2) then all i-th weights in all neurons can be regarded as the weights of the i-th level. This is specially important because the neurons are usually ordered in a two-dimensional formation (Fig. 6). The layout of neurons in the Kohonen ANN is specially important as it will be explained later on.

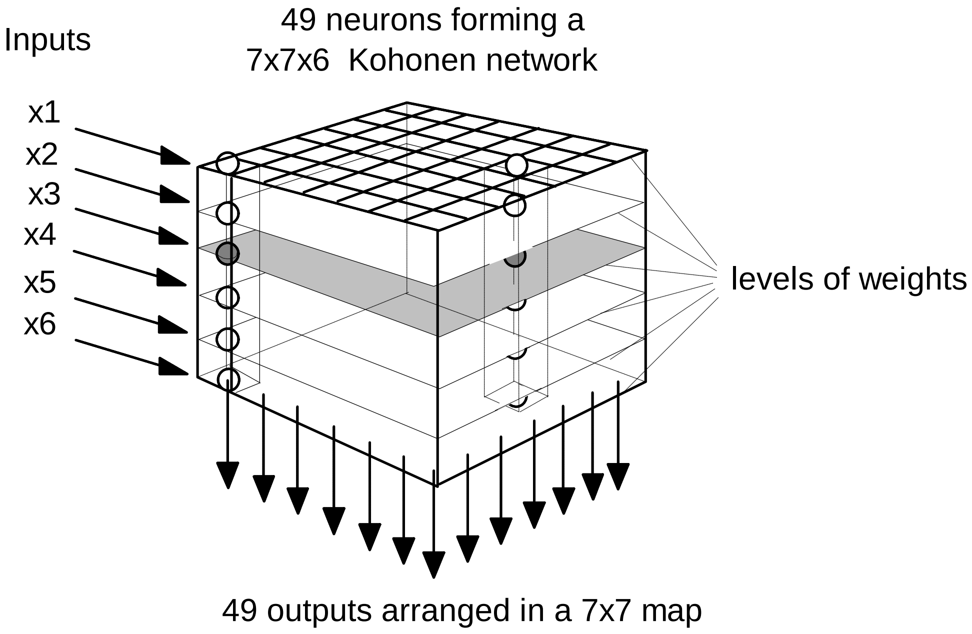


Figure 6. Kohonen ANN represented as a block containing neurons as columns and weights (circles) in levels. The level of weights which the handles the third input variable x3 is shaded. Dark shaded are the third weight on the neuron (1,1) and the third weight on the neuron (6,5), respectively

During the training in the Kohonen ANN the m-dimensional neurons are 'self-organising' themselves in the two-dimensional plane in such a way that the objects from the m-dimensional measurement space are mapped into the plane of neurons with the respect to some internal property correlated to the m-dimensional measurement space of objects. The correction of weights is carried out after the input of each input object X_S in the following four steps:

1 the neuron with the most 'distinguished' response of all (in a sense explained above) is selected and named the 'central' or the 'most excited' neuron (eq. /12/),

2 the maximal neighbourhood around this central neuron is determined (two rings of neurons around the central neuron, Fig. 7 (left)),

3 the correction factor is calculated for each neighbourhood ring separately (the correction changes according to the distance and time of training (Fig. 7 - middle and right),

4 the weights in neurons of each neighbourhood are corrected according to the equation /13/

For selecting the central neuron the distance between the input X_S and all neurons is calculated and the neuron with the smallest distance is taken as the central one:

$$\text{neuron } c \quad \leftarrow\!\!- \quad \min \{ \sum_{i=1}^{m} (x_{si} - w_{ji})^2 \} \qquad /12/$$

During the training, the correction of all m (i=1....m) weights on the j-th neuron is carried out according to the following equation:

$$w_{ji}^{new} = w_{ji}^{\ old} + \eta(t)\ a[d(c\text{-}j),t]\ (x_{si} - w_{ji}^{old}) \qquad /13/$$

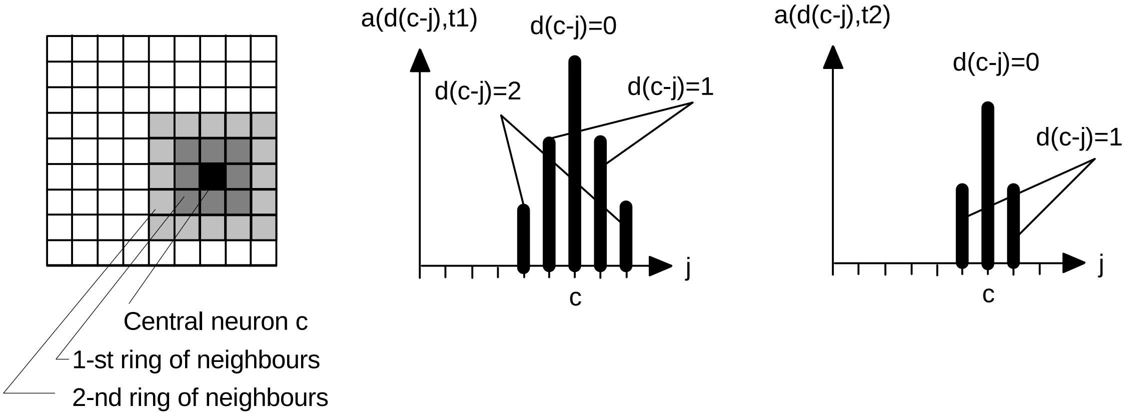


Figure 7. Neighbourhood and correction function a(.) estimation. At the beginning of the training (t=0), the correction function a(.) expands for each central neuron over the entire network. Because ANN at the left side has 9x9 neurons each (1/9)-th period of training, the correction neighbourhood will shrink for one ring. In the middle, the correction expands only over three rings, hence, t should be at about 3/9 of its final value. Note the change of parameter t from t1 to t2 in the right two pictures. At the right side a(.) covers only two rings, hence, the learning is approaching the final stage.

The learning rate term $\eta(t)$ is decreasing (usually linearly) between a given starting and final value, let us say, between 0.5 at the beginning of the training and 0.001 at the end of it. The neighbourhood function a(.) depends on two parameters. One is the topological distance d(c-j) of the j-th neuron from the central one (see Figure 7) and the second parameter is time of training. The second parameter, which is not necessarily included into function a(.), makes the maximum value at the central neuron (d(c-j)=0) to decrease during the training.

Counter-propagation ANN (8-10) is very similar to the Kohonen ANN. In fact, it is the Kohonen ANN augmented by another output layer of neurons placed exactly below it (Fig. 6). The counter-propagation ANN is an adaption of the Kohonen network to solve the supervised problems. The output layer of neurons, which has as many planes of weights as the target vectors have responses, is corrected with equation /14/ which resembles very much to equation /13/. There are only two differences. First, the neuron c around which the corrections are made (and consequently the neighbours j) is not chosen according to the distance closest to the target, but **is selected at the position c of the central neuron in the Kohonen layer** and second, the weights are adapting to the target values t_{si} instead to the input x_{si} values:

$$w_{ji}^{new} = w_{ji}^{\ old} + \eta(t)\ a[d(c-j),t]\ (t_{si} - w_{ji}^{old}) \qquad /14/$$

After the training is completed, the Kohonen layer acts as a pointer device which for any X determines the position at which the answer is stored in the output layer. Due to the fact that the answers (target values) are 'distributed' during the learning period (eq. /14/) over the entire assembly of output neuron weights, **each** output neuron contains an answer even if its above counterpart in the Kohonen layer was **never** excited during the training! Because **all** weights carry values related to the variables each plane level can be regarded as a two-dimensional map of this variable.

The overlap between the maps give us the insight how the variables are correlated among themselves (11). This is valid for weight maps from both layers. The larger is the counter-propagation ANN (the more neurons is in the plane), the better is the resolution of the maps.

In the example shown in Figure 8, five 7x7 maps of input variables in the Kohonen, and six 7x7 maps of corresponding target variables in the output layer have been developed, respectively.

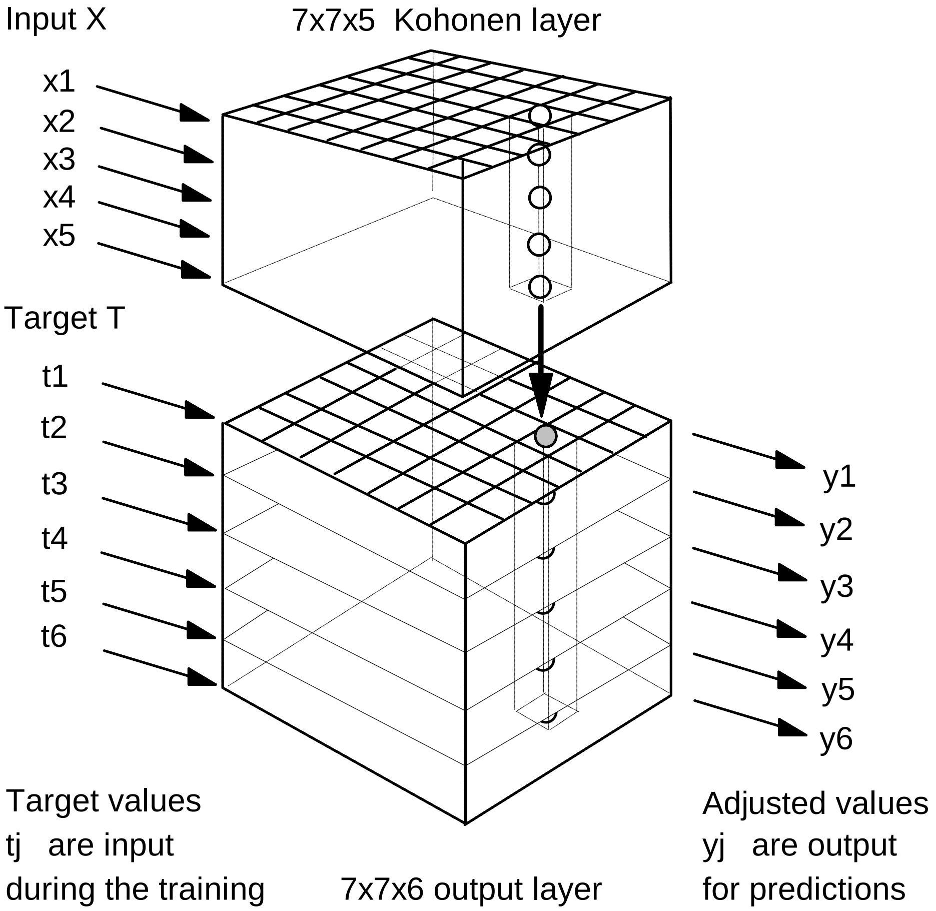


Figure 8. Counter-propagation ANN consists of two layers of neurons: of the Kohonen and of the output layer. In the Kohonen layer to which the objects X_S are input the central neuron is selected and then corrections of weights are made around its position (bold arrow) in the Kohonen **and** in the output layer using eqs. /13/ and /14/.

Applications

In a short introduction to the ANNs in which only few most often used architectures and techniques have been briefly discussed. it is not possible to give even an overview on the possibilities the ANNs are offering to the chemists. The number of scientific papers using ANNs in various areas and for solving different problems in chemistry is growing rapidly (1,2). In this paragraph only one example (11) in which error-back-propagation and Kohonen ANN techniques will be used. Because the counter-propagation method is similar to that of Kohonen, it will not be discussed here. The reader can than choose the explained techniques at his or hers own needs.

The problem we will discuss is the classification of the Italian olive oils, a problem well known in the chemometric literature (12-15). This data set has already been extensively studied by various statistical and pattern recognition methods (16). It consists of analytical data from 572 Italian olive oils (12). For each of the 572 olive oils produced at nine different regions in Italy (South and North Apulia, Calabria, Sicily, Inner and Coastal Sardinia, East and West Liguria, and Umbria) a chemical analysis was made to determine the percentage of eight different fatty acids (palmitic, palmitoleic, stearic, oleic, linoleic, arachidic (eicosanoic), linolenic, and eicosenoic). All concentrations are scaled from 0 to 100 with the respect to the range between the smallest and the largest concentration of the particular fatty acid in the set of 572 analyses.

In order to see how the data are distributed in the entire 8-dimensional measurement space, a mapping of all 572 objects was made by a 20 x 20 x 8 Kohonen ANN. The result of the Kohonen learning was a 20 x 20 map consisting of 400 neurons each of which has carried a 'tag' or a 'label' of the oil (region in Italy) that has excited it at the final test. There were no conflicts, i.e., not a single neuron was 'excited' by two or more oils of **different** origins, although some of the neurons were hit by more than seven oils. There were 237 neurons tagged and 163 neurons not excited by any of the 572 oil analyses. The later neurons were called 'empty spaces'.

For making good prediction about classes a training set that will cover the measurement space as evenly as possible is needed. Therefore, we have selected from each labelled (tagged) neuron only one object (analysed oil sample). The reasoning was the following: if two **different** objects excite the **same** neuron, they must be **extremely similar**, hence it is no use to employ **both** of them for making a model. For such purpose it is better to select those objects that excite as many **different** neurons as possible. After making selection of 237 objects we supplement them with 13 more samples from different origins to approximately balance the presence of oils from all regions. In this way 250 oil samples were selected for the training set. The rest of 322 objects was used as a test set.

For turning clustering into a supervised classification problem for each of the 8-dimensional input vector X_S (eight-concentration oil analysis) one 9-dimensional target vector T_S has to be provided in the form:

Class 1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
T(class 1) = (1 0 0 0 0 0 0 0 0)	**T**(class 2) = (0 1 0 0 0 0 0 0 0)
T(class 3) = (0 0 1 0 0 0 0 0 0)	**T**(class 4) = (0 0 0 1 0 0 0 0 0)
T(class 5) = (0 0 0 0 1 0 0 0 0)	**T**(class 6) = (0 0 0 0 0 1 0 0 0)
T(class 7) = (0 0 0 0 0 0 1 0 0)	**T**(class 8) = (0 0 0 0 0 0 0 1 0)
T(class 9) = (0 0 0 0 0 0 0 0 1)	

This scheme already implies the architecture of the ANN. The architecture should have eight input nodes (plus one for bias, but that is seldom mentioned) and nine output nodes. The only room for experimentation offers the selection of a number of neurons in the hidden layer. If the preliminary results are highly insufficient, two hidden layer architecture can be tried.

After the training, in the ideal case, only one output neuron produces the output signals equal to one and all other output neurons have an output of zero. The non-zero signal is on the neuron that was trained to 'recognise' the particular region. The actual output from the ANNs is seldom like this. More often the output values of neurons that should be zero are close to it and the values of the neurons signalling the correct class have values ranging from 0.5 to 0.99.

All learning procedure were carried out on several error-back-propagation ANNs of different architectures by applying the equations /1a/, /2a/, /8/-/10/. The values of 0.2 and 0.9 for the learning rate η and the momentum μ were used because they yielded the fastest learning convergence to the minimal number of errors during the experimental phase of this work. The best performer was the ANN with an 8 x 5 x 9 architecture giving only 25 (8 %) wrong classifications for the 322 data in the test set.

All of the 25 **wrong** predictions have produced the largest signal on the wrong neuron. There were two additional interesting cases during the test. Two objects from the 322 member test set had in fact the **largest** output on the neuron signalling the **correct** class. However, in one case this largest signal amounted only to 0.45, with all other signals being much smaller, while in the second case the largest output on the correct output was 0.9, but there was another high signal of 0.55 on another output neuron. Strictly speaking, the **total** error in both controversial answers was more than 0.5, hence both could be rejected as wrong. On the other hand, acquiring the criterion of the largest signal as a deciding factor, both can be regarded as correct.

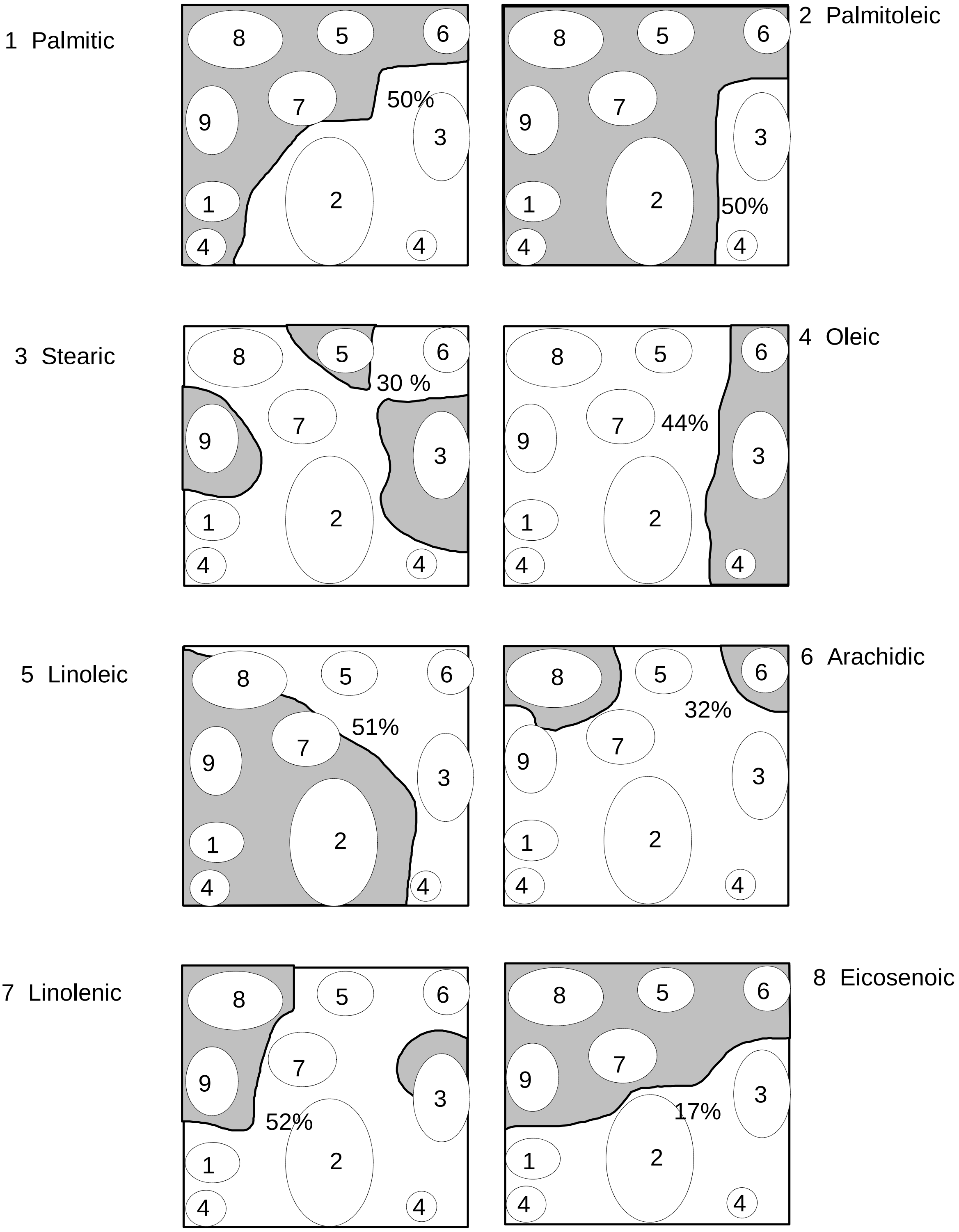
In the next experiment the 15 x 15 x 8 Kohonen ANN was trained with the same 250 objects. At the end of the training all 225 neurons were labelled with labels displaying the origin of the oils that excite them. There were, however, 64 empty labels (neurons not excited by any object) and one neuron featuring a 'conflict' (see above). Nevertheless, the test with the 322 objects not used in the training have shown surprisingly good performance. There were only 10 wrong classifications (predictions) made and additional 83 oils hit the empty-labelled neurons. Because each neuron in a square lattice has eight closest neighbours the 'empty hits' were decided according to the inspection of these neighbours: the majority count prevailed. The majority count has brought an additional amount of 66 correct, 11 undecided and 6 wrong classifications. Altogether, the result of 16 errors (10 + 6) or

5% errors and 11 or (3%) undecided answers is in excellent agreement with the previous results.

For the last experiment, an additional map, the map of oil-class labels was made on the top of the Kohonen ANN. This so called 'top-level' map consists of labels put onto the excited neurons during the learning period as already explained. With this maps (as a substitute for a counter-propagation output layer) the way how weight levels in the Kohonen and in the counter-propagation ANNs can be used to provide more information will be explained. By overlapping each weight level with the top-level map and inspecting the contours (different fatty acid concentration), it is possible to find in each map the contour line (or threshold concentration line) that most clearly divides or separates two or more classes from the rest. Note, that the classes that this contour line should separate are **not** specified. What is sought, is just **any** more or less clear division between the classes on each single map.

To use this procedure a 15 x 15 x 9 Kohonen ANN was trained with all 572 objects. In Figure 9 the maps of all eight fatty acids, i.e., eight weight levels in this 15 x 15 x 8 Kohonen ANN obtained after 50 epochs of learning are shown. For the sake of clarity, on each map only the contour line of the concentration which divides the olive oils into two clearly separated areas is drawn. For example: on the first map, the 50% contour (50% of the scaled concentration of the palmitic fatty acid) divides the classes No. 2, No. 3, and No. 4 from the rest (classes No. 1 and Nos. 5 - 9). Similarly, in the map of the last layer the 17% contour line divides clearly southern oils (classes No. 1, No. 2, No. 3, and No. 4) from the northern oils (classes No. 5, No. 6, No. 7, No. 8, and No. 9).There were some problems with oils of the class No. 4, (Sicilian oils) because this class was divided into two groups and has caused most problems in the prediction.

In the final step we have overlapped all one-contour maps obtained in a previous step and the result is shown in Figure 10. By slightly smoothing these 'most significant' contours, but still in qualitative accordance with the actual values, a good nine class division was obtained (Fig. 10 - right).



1 Palmitic

2 Palmitoleic

3 Stearic

4 Oleic

5 Linoleic

6 Arachidic

7 Linolenic

8 Eicosenoic

Figure 9. Eight weight level maps of 15 x 15 x 8 Kohonen ANN. The overlapping 'top-level' map of nine oil regions is drawn with a dotted line on each of the eight maps. Each map contains only the contour line (fatty acid concentration) that divides the 'top-level' map into two areas.

As a matter of fact, on Figure 10 only one separation is not outlined by the 'one-contour-overlap'. This is the separation between the class No. 1 (North Apulian oils) and one part of the class No.4 (Sicilian oils) which is apparently different from the rest. As a matter of fact, the group of oils in the lower left corner belong to class No. 4 which is the one that contributes most of the errors not only in the ANN applications, but as well when using standard chemometric techniques.

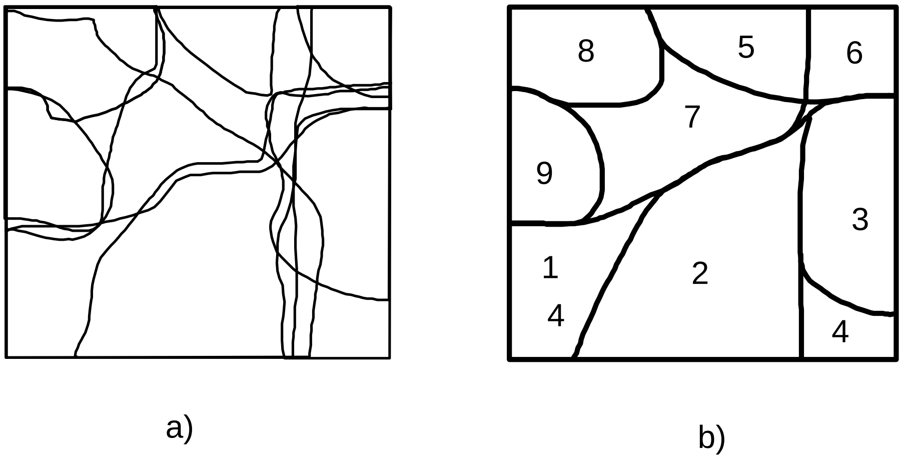


Figure 10. Eight one-contour maps overlapped (a). If the contours are 'thickened' and 'smoothed' (b) the nine classes can be clearly separated. The only 'trouble maker' is the class No 4 (see text for explanation).

As it was said before, other statistical studies on the classification of these oils, including principal component analysis (PCA(19)), nearest neighbour technique (KNN(15)), SIMCA (16), or three distance clustering (3-DC(13)), gave a comparable number from 10 to 20 mis-classifications or errors. Most of the objects (oils) that cause the errors are either wrongly assigned or subjects of too large an experimental error. In any case, in a pool of almost 45,000 (572 x 8) measurements it is hard to make less than 3 % errors. Hence, even an ideal method could not yield significantly better predictions. Therefore, it is not the quality of the results itself, but the presentation of them and the possibility for obtaining some additional information that should be possible with a new method of neural networks.

From the presentation point of view, clustering of objects on a two-dimensional map is much more informative compared to the clustering made by standard clustering techniques or a 'black-box' device which separates the objects by simple labelling them. The two-dimensional map, in our case the map of nine regions in Italy, has shown a clear separation between the southern and northern regions which cannot be easily perceived from the measured results or by a black-box prediction of classes.

By inspection of weight maps that overlap certain labelled cluster the rules for classification of this cluster can be derived. For example: cluster No. 8 West Liguria oils lies in the upper left corner of the 15 x 15 map. The inspection of the weight maps (i.e. concentrations of the fatty acids (Fig. 9) reveals that characteristic concentrations for this particular group are: low palmitic acid, low palmitoleic, high stearic, high oleic, arbitrary value of linoleic (see upper left corner of the corresponding map on Figure 9), low arachidic, low linolenic, and finally, low concentration of the eicosenoic fatty acid. Such rules can be obtained for groups formed on the map of labels. Hence, it can be safely said that the weight maps of Kohonen or counter-propagation ANNs have the ability to produce complex decision rules.

Conclusions

As it was shown in the above example, the ANNs have a broad field of applications. They can do classifications, clustering, experimental design, modelling, mapping, etc. The ANNs are quite flexible for adaption to different type of problems and can be custom-designed to almost any type of data representations. The warning, however, should be issued here to the reader not to go over-excited upon a new tool just because it is new. A method itself, no matter how powerful it may seem to be can fail easily, first, if the data do not represent or are not correlated good enough to the information sought, secondly, if the user does not know exactly what should be achieved, and third, if other standard methods have not been tried as well - just in order to gain as much insight to the measurement and information space of data set as possible. ANNs are not 'take-them-from-the-shelf' black-box methods. They require a lot of study, good knowledge on the theory behind it, and above all, a lot of experimental work before they are applied to their full extend and power.

Acknowledgements

The author wish to thank the University Rovira i Virgili, Tarragona, Spain, awarding him with a Visiting Professor fellowship, during which one of the results was this lecture. Financial assistance by the Ministry of Science and Technology of Slovenia on the application part of this lecture is gratefully acknowledged.

Literature

1. J. Zupan, J. Gasteiger, Neural Networks: A New Method for Solving Chemical Problems or Just a Passing Phase?, (A Review), Anal. Chim. Acta **248,** (1991) 1-30.

2. J. Gasteiger, J. Zupan, Neuronale Netze in der Chemie, Angew. Chem. **105,** (1993) 510-536; Neural Networks in Chemistry, Angew. Chem. Intl. Ed. Engl. **32,** (1993) 503-527.

3. J. Zupan, J. Gasteiger, Neural Networks for Chemists: An Introduction, VCH, Weinheim, 1993

4. D. E. Rumelhart, G.E. Hinton, R.J. Williams, 'Learning Internal Representations by Error Back-propagation', in Distributed Parallel Processing: Explorations in the Microstructures of Cognition, Eds. D.E. Rumelhart, J.L. MacClelland, Vol. 1. MIT Press, Cambridge, MA, USA, 1986, pp. 318-362.

5. R. P. Lipmann, An Introduction to Computing with Neural Nets, IEEE ASSP Magazine, April 1987, 155-162

6. T. Kohonen, Self-Organization and Associative Memory Third Edition, Springer Verlag, Berlin, FRG, 1989,

7. T. Kohonen, An Introduction to Neural Computing, Neural Networks, **1**, (1988), 3-16

8. R. Hecht-Nielsen, Counter-propagation Networks, Proc. IEEE First Int. Conf. on Neurall Networks, 1987 (II), p. 19-32

9. R. Hecht-Nielsen, Counter-propagation Networks, Appl. Optics, **26**, (1987) 4979-4984,

10. R. Hecht-Nielsen, Application of Counter-propagation Networks, Neural Networks, 1, (1988) 131-140,

11. J. Zupan, M. Novic, X. Li, J. Gasteiger, Classification of Multicomponent Analytical Data of Olive Oils Using Different Neural Networks, Anal. Chim. Acta, **292,** (1994) 219-234,

12. M. Forina, C. Armanino, Eigenvector Projection and Simplified Non-linear Mapping of Fatty Acid Content of Italian Olive Oils, Ann. Chim. (Rome), **72**, (1982), 127-143; M. Forina, E. Tiscornia, ibid. 144-155.

13. M. P. Derde, D. L. Massart, Extraction of Information from Large Data Sets by Pattern Recognition, Fresenius Z. Anal. Chem., **313**, (1982), 484-495.

14. J. Zupan, D. L. Massart, Evaluation of the 3-D Method With the Application in Analytical Chemistry, Anal. Chem., **61**, (1989), 2098-2182.

15. M. P. Derde, D. L. Massart, Supervised Pattern Recognition: The Ideal Method? Anal. Chim. Acta, **191**, (1986), 1-16.

16. D. L. Massart, B.G.M. Vandeginste, S.N. Deming, Y. Michotte, L. Kaufmann, Chemometrics: a Textbook, Elsevier, Amsterdam, 1988.

17. K. Varmuza, Pattern Recognition in Chemistry, Springer Verlag, Berlin, 1980.

18. J. Zupan, Algorithms for Chemists, John Wiley, Chichester, 1989,

19. K. Varmuza, H. Lohninger, Principal Component Analysis of Chemical Data, in PCs for Chemists, Ed. J. Zupan, Elsevier, Amsterdam, 1990, pp. 43-64.

Izvlecek

Opisani in razlozeni so osnovni pojmi umetnih nevronskih mrez (UNM), kakor tudi tri najbolj pogosto uporabljene strategije ucenja pri UNM. Obravnavane tri metode so: ucenje z vzvratnim sirjenjem napake, Kohonenovo ucenje in ucenje z vhodom podatkov iz nasprotnih smeri. Da bi pokazali, kako je obravnavane tri metode mozno uporabiti v kemiji, je predstavljen primer klasifikacije in napovedovanja izvora olivnih olj. Vsako olivno olje je predstavljeno z osmimi koncentracijami razlicnih mascobnih kislin.