THE APPLICATION OF ARTIFICIAL NEURAL NETWORKS IN THE ELECTRONIC NOSE FOR ODOUR MEASUREMENT

BY

TONG LI

A Thesis
Submitted to the Faculty of Graduate Studies
In Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

Department of Biosystems Engineering
University of Manitoba
Winnipeg, Manitoba

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION

THE APPLICATION OF ARTIFICIAL NEURAL NETWORKS IN THE ELECTRONIC NOSE FOR ODOUR MEASUREMENT

BY

Tong Li

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirement of the degree

Master Of Science

ABSTRACT

A JAVA program (ENBrain) was developed to implement artificial neural networks in electronic noses (e-noses) for odour assessment. Three-layer feed-forward neural networks with sigmoid function neurons were built up by ENBrain. The Back-propagation algorithm was applied to train the networks. Networks with different number of hidden units and initial weights were experimented. N-butanol was used as the test odorant and measured with an Alpha MOS Fox 3000 e-nose equipped with 12 metal oxide sensors. Each n-butanol sample was analysed with 4 replications, and a total of 80 data vectors were collected. Among these 80 data vectors, 32 ones were used for training the neural networks and the other 48 were used for testing the neural networks. Odour intensities of n-butanol samples were assessed by human panels using an eight-point and a 0-200 intensity scale. PCA was performed on training and testing data sets. For training set, all data vectors are linearly separable, while for testing set, four n-butanol samples are more difficult to be linearly separated.

High accuracy (100%) was achieved for predicting odour concentrations and intensities during training. With the increase in the number of hidden units, an increase in accuracy and decreases in the SSE and MSE were observed from 1 to 14 hidden units and little changes were observed after the number of hidden units reached 14.

Trained networks were tested over the testing data set. High coefficient of determination ($R^2 = 0.89$) was obtained between network predicted and actual odour concentrations. A high correlation ($R^2 > 0.95$) was obtained between network predicted and human panel assessed odour intensities in both the eight-point and 0-200 intensity scales.

ACKNOWLEDGEMENTS

First, I wish to thank those who provided guidance and assistance to me throughout the project, in particular, my advisor, Dr. Qiang Zhang. I would also like to thank my committee members, Dr. Michael S. Freund and Dr. Jitendra Paliwal.

Second, I want to thank Roberta K. York for her technical support and providing e-nose sensor data used in the thesis, and Xiaojing Zhou for her discussion about the thesis.

Finally, I would like to thank all those who supported me through this project. I want to thank my parents for their tolerance and love. I also would like to thank all my friends. I must also thank all the staff and students of Biosystems Engineering for making the past two years a real joy.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The e-nose technology provides cost-effective alternatives for accurate, reliable and speedy identification of environmental pollutants such as odours (Kress-Rogers 1996; Srivastava and Levy 2002). However, until now the recognizing abilities of electronic nose are quite limited. Some well-trained human experts are able to recognize more than 4000 different odours. In comparison, the recognizing abilities of electronic noses are still far behind trained human noses. The reasons are that the sensitivity and selectivity of gas sensors in e-noses are not ideal; and the abilities of the statistical and the neural-network-based classification methods for large-sample, multi-class and high-dimensional problems are not as good as expected (Gao et al. 2004). However, in many applications, we need to detect odours, instead of recognizing them. This leads to measurement of odour concentration or intensity. Recent work has shown a clear relationship between odour concentration and the averaged e-nose sensor response for agricultural odours at low concentrations, suggesting the possibility of developing an instrument for field measurements of agricultural odours (Misselbrook et al. 1997). The main advantage of an e-nose is that once calibrated, it can be used to perform odour assessment on a continuous basis at a minimal cost (Hudon and Guy 2000). However, odour is difficult to define as it is the relationship of a physiological effect to the stimuli (Dowdeswell and Payne 1999). Future development is needed to address the relationship between sensor output and odour intensity, rather than odour type (Gostelow 2001).

Odour is the sensations that occur when a complex mixture of odorants stimulate receptors in the nasal cavity. In other words, odour is a complex physiological variable, not a simple physical or chemical variable. Typical e-nose consists an array of gas

sensors, which generates multiple dimension data, e.g., 10 gas sensors produce a data vector with 10 dimensions. Considering the multiple dimension problems, the conventional statistical approaches lack of power to analyse e-nose data. Artificial neural network can be applied appropriately for certain types of problems, where observations are represented by many attribute-value pairs, the training examples may contain errors, long training times are acceptable, fast evaluation of the learned target function may be required, and the ability of humans to understand the learned target function is not important (Mitchell 1997). With these problem characteristics, the e-nose problem is an appropriate problem, which artificial neural networks can be applied for.

This study was focused on developing artificial neural networks to correlate e-nose measurements to human assessment of odour intensity. In this study, I am trying to find answers to the following questions: 1) Is it possible for artificial neural networks to predict odour intensities and concentrations? 2) How do the network parameters affect network performance?

2. OBJECTIVE

The goal of this research was to develop a computer Graphical User Interface (GUI) software to implement artificial neural networks for predicting livestock odours based on data collected by e-noses. Because data for training and validating the artificial neural networks were collected with a 12-sensor commercial e-nose for a standard odorant (n-butanol), the objectives were centered on these specific conditions. These objectives were:

1. to develop a back-propagation artificial neural networks to correlate data from 12-metal oxides e-nose sensors to human assessed odour intensity for n-butanol, and
2. to conduct numerical experiments to evaluate the performance of the artificial neural networks.

3. LITERATURE REVIEW

3.1 Introduction to Electronic Nose (e-nose)

The electronic nose typically comprises a sensor system and a recognition system. The sensor system for odours is typically an array of gas sensors. The recognition system is often implemented on a computer. Figure 1 shows the basic structure and general processes in an e-nose. The odour sample is presented to the sensor system. The sensor responses are recorded in a predetermined information format (e.g., electrical signals), which is readable to the recognition system. The recognition system analyses this information or data and tries to "define" the odour. The recognition can be viewed as data analysis. Both conventional statistical approaches and pattern recognition approaches can be used in data analysis. However, pattern recognition is more commonly used in recent electronic nose studies.

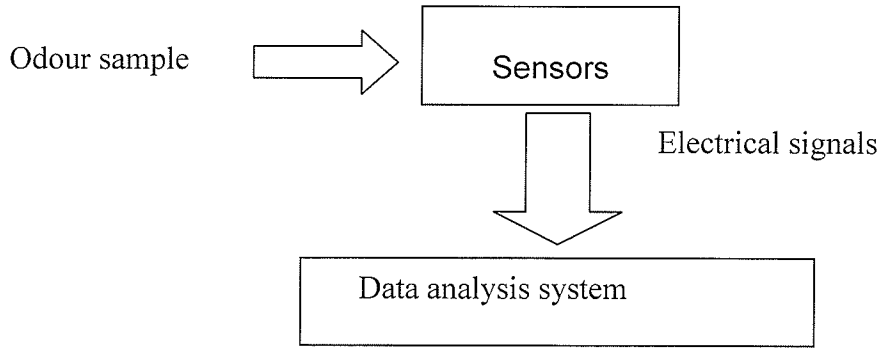

Figure 1. Basic structure and general processes of an e-nose

Nearly 20 years after the first idea of the electronic nose was published, the use of electronic noses started to take place in industry in the mid-1990s. This delay is due to the complexity of such a system and the requirement of advanced technologies (e.g., the

more sensitive sensors) (Gardner 1999). Another reason for the delay is that the electronic nose is an interdisciplinary research area. It requires the knowledge of odour molecules (chemistry), biological mechanism (biology), odour sensors (electronic engineering), and data processing (statistics and computer science).

3.2 Issues about odours

3.2.1 The human perception of odour

The sense of smell makes the most significant contribution to humans' perception of odour (Gardner 1999). Since the olfactory receptors are located in a high up membrane within the nose, humans can only perceive volatile odour compounds that can reach these receptors through the air. Dutta et al. (2003a) stated that the mammalian olfactory system consists of a large number of non-specific receptors - with about 300 different olfactory binding proteins having been identified - in a total of about 50 million. Receptor cells are not specific to individual molecules - instead they have broad responses with a large overlap between different cell classes (Pearce, 1997). These cells send their signals to secondary nodes and then cells located in the olfactory bulb. There is a marked convergence at this stage with between 1000 and 20,000 primary receptor cells connecting to each secondary cell followed by limited divergence (Pearce et al. 1993). Secondary cells interact with each other and with higher cells as well. Thus the system is complex and non-linear.

The response and perception varies with the sensitivity of individual olfactory systems which are affected by genetics and age, physical condition of individual, and the individual's experience and memory of exposures to the odour (Barth 1973; Anonymous 1998). In Gilbert and Wysocki (1987)'s study, which involved 1.5 million individuals, it

was shown that for both sexes the highest sensitivity to odours is reached after puberty and declines after the age of 70. At all ages, females are more sensitive to odours than males. The individuals' state of health and their endocrinological condition have an effect on their sensitivity to odours, e.g., influenza, a cold, or sinus infection can result in a temporary loss of odour perception.

Odour intensity is defined as the perceived magnitude of a stimulus (ASTM 1991). Odours of equal concentration do not necessarily be of equally perceived intensity (Sarkar and Hobbs 2002). The odour intensity depends on the concentration of each compound and the combination of these compounds as well (Zhou 2001). Stevens (1957) proposed that the perceived psychological intensity is a function of the odorant concentration as $I = k(C - C_0)^n$, where I is the perceived psychological intensity; k is the constant dependent on the choice of units of C, I and odorant; C is the physical intensity expressed as concentration of odour compound; C_0 is an estimate of the odour detection threshold; n is the constant dependent on odorant. Katz and Talbert (Cain and Moskowitz 1974) found that psychophysical functions for most substances were in accord with the logarithmic relation and they tested it for about 55 odorants.

3.2.2 Simple and complex odours

Simple odours are defined as a single, well-defined compound. Complex odours are mixtures of odour compounds. Naturally occurring odours from foodstuffs, beverages, and plants and the odour of perfumes are not due to single molecular species but rather are the result of complex mixtures of molecules made up of tens, hundreds, or even thousands of separate compounds (Gardner 1999). In the case of coffee, over 670

different components have been identified as playing a role in determining the coffee odour (Ho et al. 1993).

3.2.3 Conventional techniques for odour measurement

When considering the odour measurement, it is important to discuss the difference between odorants and odours. An odorant is the compound responsible for imparting an odour, whereas an odour is the perceived effect of the odorant as detected and interpreted by the olfactory system (Gostelow et al. 2001). Because of the lack of a comprehensive theory of olfaction, the exact relationship between odorant properties and odour perception is not clear. This difference between odorants and odours leads to two broad classes of odour measurement techniques: instrumental analysis referring to odorants and sensory evaluation referring to odours.

Instrumental (analytical) measurement concerns the physical or chemical properties of odorants, most commonly measuring odorant concentration. Instrumental methods such as gas chromatography (GC), gas chromatography with mass spectrometry (GC-MS), sniffing GC-MS, and/or high performance liquid chromatography (HPLC), combined with appropriate sample preparation techniques, are routinely employed for the analysis of concentrations of specific gases in odour (Berna et al. 2004). These techniques provide objective, repeatable and accrual odorant measurement. Most importantly they relate directly to theoretical models relating to odorant formation or emission (Gostelow et al. 2001). The disadvantages of instrumental analysis are summarised as follows (Brennan 1993; Preti et al. 1993; Stordeur et al. 1981; Young 1984):

1. Since most naturally occurring odours are complex mixtures of a number of odorants, employing instrumental analytical techniques need separation prior to analysis.
2. Non-odorous compounds may be detected by instrumental analysis.
3. Instrumental analysis cannot measure the odour intensity, which is perceived by human nose as a result of a mixture of odorants.

Sensory evaluation using the human nose is the basis for most of the experimental measurements of odour and the use of human panel is currently widely accepted as the best method for odour evaluation (Riskowski 1991; Clanton et al. 1999). Berna et al. (2004) stated that the most important issues in sensory analysis include the standardisation of measurements, the correctness of training human assessors, and the stability and reproducibility of the evaluation. The relatively high costs for training and use of sensory panels are also a major drawback of this technique. It is well known that olfactometry results vary widely between laboratories (Schulz and van Harreveld 1996) and human factors affect sensory panel performance (Bliss et al. 1996). Considering that a judgment of odour depends not only on the permanent or fundamental makeup, such as race and sex, but also on the temporary make-up, as age, occupation, the previous history of odour experiences, a cold in the head, or even on such a fleeting phenomenon as menstruation, a mental upset, and others, anyone who has worked on a panel, or has studied the methods employed and read panel reports, will admit that panel findings are subjective, and that even a trained panel member is not a constant adjudicator (Summer 1971).

In summary, the instrumental analysis measures odorants, whereas sensory evaluation measures odours. However, these techniques are cost and time consuming. There is a need for the e-nose, which can complement these conventional techniques.

3.3 Odour sensors in e-noses

3.3.1 The relationships between the molecular shape and odours

The size, shape, and polar properties of the molecules determine its odour properties (Dutta et al. 2003a). It is possible to predict odours by their molecular shapes or chemical structures. However, the current discovery of the relationships between molecular shapes and odours is insufficient to do so in practice. Gardner (1999) stated two rules for the relationships between the molecular shape and odours:

(i). Compounds with same functional groups, similar overall shape and size, could have same odour.

(ii) Moving the position of a functional group can alter the odour. For example, enantiomeric molecules and diastereomeric molecules can exhibit very different odours.

However, these rules are not infallible and there are many exceptions. Moncrieff (1951), Stoll (1957), Naves (1957) and Beets (1957) had the following general conclusions:

(i) Although much information on the relation between specific areas of odours and chemical structure has been elucidated, little can be generalized to the whole field.

(ii) It is still not possible to say why certain molecules are odorous when often very similar molecules with very similar vapour pressures are not.

(iii) There is general agreement that both the overall shape and size of the molecules and the presence of particular functional groups are important features, which determine the odour of the molecules.

There is a lack of an overall correlation pattern between odour and chemical constitution (Harper et al. 1968). Even a very superficial perusal of the results shows that the different sets of rules are embarrassingly unconnected (Beets 1964).

3.3.2 The principle of odour sensors

Since the molecular shapes are essential in determining the properties of odours, we would design odour sensors by the structure-odour relationship. Unfortunately, our understanding of the relationship is insufficient to design such sensors (Gardner 1999). Indeed, the current odour sensors comprise a chemically sensitive material interfaced to a transducer. The chemically sensitive material generates a physical change by interacting with odours, and then the transducer senses that physical change and converts it into an output signal, usually an electrical signal. The responsiveness of sensors is not highly specific to a particular chemical species, and thus a single sensor cannot be expected to perform reliably as the detector of single substances (Mccoy et al. 2003). Instead, an array of sensors is commonly used in e-noses, and the sensors responses are collected as an odour-specific pattern. This is thought to be analogous to the human olfactory system, where receptor cells are not specific to individual molecules but have broad responses with a large overlap between different cell classes (Dickinson et al. 1998; Bartlett et al. 1997; Byfield and May 1996; Gardner and Bartlett 1994).

3.3.3 General requirements of odour sensors

The odour sensors used in e-nose applications are different from other chemical sensors in many aspects. Desirable characteristics of electronic nose sensors have been described by Bartlett et al. (1997). Moreover, Gardner (1999) discussed the general requirements for odour sensors used for e-nose as follow:

The sensors should respond in a reasonable time. The response time is defined as the time taken for the device to reach some predetermined fraction of its final output. Ideally in practice, it means just a few seconds. However, in many applications, a longer response time is adequate. On the other hand, the recovery time defined as the time taken for recovering from exposure to samples is also critical. The response time and recovery time together determine the sample throughput, i.e., the number of samples we can test for each hour.

The sensors should respond at an appropriate concentration level. This is difficult since odours have different detection thresholds. Some odours are effective at a very low concentration, but others have a much higher threshold. The range for proper concentrations depends on the sensor device and the odour molecules. Furthermore, this concentration range should be wide enough for different odours and the sensors do not saturate in such a normally encountered concentration range.

The sensors should be reproducible. The reproducibility includes two aspects, the reproducibility of a particular sensor throughout the lifetime of that sensor, and the reproducibility between the same types of sensors. Reproducibility of a particular sensor can provide us with a stable sensor response. This is very important for pattern recognition. Furthermore, this reproducibility can avoid the drift problem, which is the

general, slow change in sensitivity due to aging effects. It is interesting to note that the biological solution to this problem is pooling response from a large population of receptor cells. The other aspect of reproducibility is reproducing between the same types of sensors. This reproducibility can solve the sensor-poisoning problem, which arises when a sensor is exposed, often inadvertently, to a material that irreversibly binds to, or interacts with, the sensing material leading to a reduction or even total loss of sensitivity. If a sensor is poisoned, we can replace it without the need of recalibrating the system. Furthermore, this reproducibility means we can use the same sensor data to train (calibrate) different e-noses that are located in different sites. It makes the distributive training possible. The potential approach to achieve this reproducibility is automated and controlled manufacture of sensors.

The sensors should be insensitive to environment changes, such as temperature, humidity, and flow rate. However, all current sensors show some sensitivity to these environmental parameters. In principle, we can eliminate the sensitivity by careful system design and sample handling. However, this leads to a complex and expensive system, and limited sample throughput. One possible solution to this problem is to view sensor array as time-varying dynamic systems, whose variation has to be tracked by adaptive estimation algorithm (Holmberg 1997), i.e., considering environmental effects in the pattern recognition system.

Since the array of sensors used in e-noses generates odour patterns (fingerprints), which are discernibly different for different samples, the sensors should respond to a broad range of compounds. At the same time, the response shown by different elements within the array to a particular compound should not be highly correlated otherwise the

information content of the pattern produced by the array will be low (Gardner and Bartlett 1996).

The odour sensors should be responsive to molecules in the gas phase, since most odours are in gas phase. The cost and size of a sensor should be minimized, since we are using an array of sensors for e-noses. Smaller sensors need less volume of samples, which can speed up response time and improve sensitivity by avoiding sample dilution.

3.4 Data analysis in e-noses

3.4.1 Odour concentration vector and sensor response vector

Since each odour comprises of a number of chemical components (compounds), we can use the concentration of each component to describe an odour. A vector, called a concentration vector, represents these concentrations. Gardner (1999) defined the concentration vector as:

$$c_j = \begin{bmatrix} c_{1,j} \\ c_{2,j} \\ \cdots \\ c_{mj} \end{bmatrix} \qquad (1)$$

where

c_j = the concentration vector for the j'th odour sample

c_{mj} = the concentration of the m'th component in the j'th odour sample.

In most cases, we employ a sensor array for an e-nose system. The sensor array consists of a number of gas sensors. Each sensor generates an output based on odour samples. Therefore, the output of the sensor array produces a vector, which is defined by Gardner (1999) as the following:

$$x_j = \begin{bmatrix} x_{1j} \\ x_{2j} \\ \cdots \\ x_{dj} \end{bmatrix} \tag{2}$$

where

x_{dj} = the output from the d'th sensor corresponding to the j'th odour component

x_j = the sensor response vector for the j'th odour sample.

3.4.2 Statistical approaches for sensor data analysis

In conventional statistic approaches, we assume a linear regression model to describe the relationship between sensor data and odour concentrations. This model is presented as:

$$\underset{(d\times 1)}{x_j} = \underset{(d\times m)}{A} \times \underset{(m\times 1)}{c_j} \tag{3}$$

where

A = transformation matrix

c_j = the concentration vector for the j'th odour sample

x_j = the sensor response vector for the j'th odour sample

d = the number of sensors

m= the number of components.

In this model, we are trying to find the transformation matrix $\underset{(d\times m)}{A}$ by exploring a number of odour samples. Linear regression techniques, such as multiple linear

regression (MLR) and partial least squares (PLS), can be used to find such a transformation matrix. Given the sensor data from nine coated bulk acoustic wave (BAW) gas sensors, Carey et al. (1987) successfully applied MLR to analyse certain organic vapours. Hierlemann et al. (1995) applied a PLS analysis on the sensor output of an array of six polymer-coated BAW devices to predict concentrations of n-octane and toluene.

There are two important assumptions for this linear regression model (Gardner 1999):

1) A linear model is valid, i.e., the response of each sensor is proportional to the component concentration.

2) The principle of superposition holds, i.e., an additive linear model of the response of a sensor to the m components. Therefore, we have $A_{i1}c_1 + A_{i2}c_2 ... + A_{im}c_m = x_i$, where $1 \le i \le d$, c_m represents the m'th concentration for one odour, x_i represents the i'th sensor output corresponding to this odour.

The second assumption means the response of a sensor to a mixture of, say, two components equals the sum of the responses to the individual components (Gardner 1999). Since the interactions between components within an odour are weak ones in the electronic nose's low component concentration experience, the second assumption (additive effect) holds in most electronic nose applications (Gardner 1999). Therefore, the matrix multiplication holds in equation 3. However, in many situations, there is no linear relationship between sensor response vector and concentration vector. Linear model is not valid in these situations.

Furthermore, Gardner (1999) pointed out that if we try to predict concentrations from the regression model, the number of observations (samples) increases approximately to the power of the number of components. Unfortunately, most naturally occurring complex odours consist of hundreds or thousands of components. Therefore, we need a huge number of observations. This is not applicable in practice.

Because of the invalidity of linear models and the need of a huge number of observations, conventional statistical approaches are not suitable in electronic nose applications. It is neither possible, nor indeed necessary for an electronic nose to predict the individual component concentration (Gardner 1999). Alternatively, the data analysis problem can be simplified to a pattern recognition problem. In this case, we do not need to predict the individual concentration for each component. Indeed, the sensor response vector is viewed as patterns or fingerprints of odours, and we distinguish these patterns by their differences.

3.4.3 Pattern recognition

3.4.3.1 Feature extraction

If we use concentrations of odour components to describe odours, each concentration could be used as a feature of the odour. Ideally, if we find each component concentration, we can distinguish odours by their concentration vector. These concentration vectors are actually odours' pattern. However, it is difficult and almost impossible to find the whole concentration vector. Most complex odours consist of hundreds or thousands of odour components. Considering the e-nose situation in which a number of gas sensors are applied, there is a need of hundreds or thousands of sensors since each sensor should only respond to a specific component in order to obtain the

whole concentration pattern. This ideal case is not applicable in practice. The number of sensors is restricted by their cost and size and current sensors respond to a number of components. In an e-nose, the use of a sensor array results in partially over-lapping sensitivity such that each sensor responds to a range or class of gases rather than to a specific one (Zaromb and Stellar 1984).

In statistical data analysis approaches, we are trying to find a transformation matrix A (eq. 3). However, from the pattern recognition's point of view, it is not necessary for us to find such a transformation matrix. Furthermore, we do not even need to know the concentration vector c_j. What we are looking for is the response vector x_j, since these vectors represent odour patterns. Our objective is discriminating odours by their patterns, i.e., response vectors from gas sensors. Therefore, the features we actually use are sensor responses, although the ideal feature set consists of all component concentrations. The transformation from concentrations to sensor response can be viewed as feature extraction, which can also be represented by the linear regression model equation. The elements of the $(d \times m)$ transformation matrix A are defined by the sensitivity and specificity of each of the d sensors to the different m components in the odour of class j (Gardner, 1999). A_{ik} represents the sensitivity and specificity of the i'th gas sensor to the k'th component in such an odour. Therefore, the feature extraction depends on the type of gas sensors.

Generally, the number of components (m) is much greater than the number of sensors (d), i.e., $m>>d$, since most odours are complex odours, which comprise a great number of components (e.g., 670 identified components in coffee). On the other hand, d is always small, typically from 6 to 32. The reason is that odour sensors are expensive,

and more sensors means larger size of sensor array, which leads to larger volume of odour sample. Considering the limited number of sensors, the sensors used in electronic noses should respond to a broad range of odour components and be not correlated. However, since the number of different odour components is huge, there are no such ideal sensors. Therefore, we might lose information after feature extraction.

3.4.3.2 Overview of pattern recognition techniques applied in e-noses

The PCA (Principal Component Analysis) is a technique generally used for data compression and reduction of dimensionality while simultaneously retaining the information present in the data (Lavine 2000). The PCA is a linear orthogonal transformation from multi-dimensional input space to a two or three dimensional space, such that the coordinates of data in the two or three dimensional space are uncorrelated and maximal amount of variance from the original data is preserved by only a small number of coordinates (Byun et al. 1997). Therefore, a multiple dimensional problem can be transformed to a two or three-dimensional problem in order to be plotted and visualized by human observers. In the analysis of responses from the electronic nose, the PCA acts as a decorrelator at the pre-processing stage and maximises the variance within the data before the classification procedure is performed (Kermit and Tomic 2001). Sohn et al. (2003) concluded that a PCA pre-processing has a role not only to reduce dimensionality of training data but also to enhance the generalisation of the neural network. Byun et al. (1997) explored the possibility of using PCA for visualisation of the difference between odour patterns obtained from an array of 20 conducting polymer sensors. In their experiment, how different one cluster is from another depends only on human observations. In Gardner's (1991) experimental study, twelve tin oxide gas

sensors were exposed to five alcohols and six beverages, and the response was studied using cluster analysis (CA) and PCA. He reported that the individual alcohols separated out into five distinct clusters, whereas the beverages clustered into only three distinct classes.

Among the potentially useful Pattern Recognition (PARC) techniques, artificial neural networks (ANNs), to a certain extent, mimic the human olfactory system of processing information. Therefore, artificial neural networks are generally considered as most promising tools for understanding the PARC problems in chemical sensing (Gardner et al. 1992). Unlike other pattern recognition methods, an ANN inspired by biological nervous systems is a dynamic, self-adapting system that can modify its response to external forces using previous experience, offering a more flexible and faster method of analysis (Chelani et al. 2002). ANN has proved more to be adaptable to events occurring in real analytical situations because it is much more resistant to random errors, and drift in sensor signal magnitudes (Sohn et al. 2003). For an artificial neural network, topology as well as weight factors of neurons determine the main properties of the network (Korenman and Kalach 2003).

Back-propagation neural network is the most commonly used neural network, and is discussed in the next section. Another commonly used network, Radial Basis Function (RBF) neural network, is an approach to function approximation that is closely related to distance-weighted regression and also to artificial neural networks (Mitchell 1997). The early discussions about the RBF neural networks were given by Powell (1987), Broomhead and Lowe (1988), and Moody and Darken (1989). RBF is an attractive classification technique and many researchers use the RBF network for e-nose

applications. Evans et al. (2000) evaluated a RBF neural network for the determination of wheat quality from electronic nose data. They stated that because other neural network architectures require long training time and sometimes reach false minima in the training cycle, the use of RBF neural network circumvents this and provides for very rapid training and robust predictive power. They found that the size of training set is an important factor in determining the accuracy of any predictions made using the network, and reversing the training and unknown sets in the example clearly improves the predictive power of the network to this particular problem. Finally, they reported that the RBF neural network achieved a predictive success of 92.3% with no bad samples misclassified as good in a 40-sample population (24 good, 17 bad) using a training set of 92 samples (72 good, 20 bad). Furthermore, Ali et al. (2003) applied RBF neural networks for an e-nose, which classifies fresh edible oils. In their research, an electronic nose utilising an array of six-bulk acoustic wave polymer coated Piezoelectric Quartz (PZQ) sensors was developed. A 6-6-3 (6 input neurons, 6 hidden neurons, 3 output neurons) neural network was implemented to classify three oils, i.e., Extra virgin olive (EVO), Non-virgin olive oil (OI), and Sunflower oil (SFO). Their report showed, after being trained on 233 data sets and tested on 113 sets selected randomly from the full data set; only one sunflower oil sample was misclassified as non-virgin olive oil. It was found that the results of RBF were excellent, giving classifications of above 99% for the vegetable oil test samples. However, unfortunately, the abilities of RBF networks heavily depend upon whether the number, locations, and widths of radial basis function kernels are appropriate (Gao et al. 2004). Furthermore, since RBF is based on the distance

between patterns, how we define the distance in input space is essential to the performance of RBF.

The Genetic Algorithm (GA) was introduced by Holland (1975), and Goldberg (1989) described the steps in the GA. GAs are based on the evolutionary model, which maintains that the best-adapted species have better chances of survival (Lucasius and Kateman 1993; Siedlecki and Sklansky 1988). Applying GA to optimize the neural network has been investigated by many researchers, including Yao (1993), Balakrishnan and Honavar (1995), Schaffer et al. (1992), and Falco et al. (1998). A number of studies of genetic algorithm neural networks (GANN) have been conducted for e-nose applications. Srivastava et al. (1998) explored neuro-genetic applications in processing electronic nose data corrupted with additive Gaussian noise. In their study, published sensor data for different polymer-coated surface-acoustic wave (SAW) sensor arrays exposed to fixed concentrations of hazardous vapours like diethyl sulphide (DES) and iso-octane (ISO) were used. They implemented a standard back-propagation neural network (BPNN) and a genetically trained neural network (GANN). In the GANN, they encoded the neural network into binary strings called chromosomes by concatenating the weight values one after the other and found that 10 or twenty bits per weight seem to be an adequate resolution (10 bits in their study). They also introduced a new crossover operator, which is called "Multipoint Restricted Crossover (MRX)" to overcome the difficulty in convergence with the standard crossover operator. To compare BPNN with GANN, they varied the dimensionality of instances by taking different numbers of sensors. In their report, the BPNN and GANN showed comparable performance for low dimensionality instances, but GANN was far superior for high dimensionality instances.

Kermani et al. (1999) combined the GA and Levenberg-Marquardt (LM) neural network algorithm to enhance performance in an e-nose. In their study, they reduced the network optimization problem to the selection of the number of neurons, and GA was employed to manage this selection. According to the results, they suggested that by combining a GA with an NN, one can enhance the performance of an e-nose classifier system for three odour classes, i.e., fragrance, hog farm air, and soft beverages. Boilot et al. (2002) implemented a GA to reduce the instance dimensionality in an e-nose application for discriminating standard fruits. They reduced the dimensionality by 60-70% and achieved accuracy of around 94.3%.

There are many pattern recognition techniques that have been applied to e-nose data analysis, but there is no agreement that one technique is superior to others. The choice of pattern recognition techniques depends on the available data and the type of result that is required (Schaller et al. 1998). The nature of data also decides which a technique is more suitable in a particular application than others.

3.5 Artificial neural network for e-noses

3.5.1 Typical structures

Neural networks comprise connected units and weighted connections. These units, which are called neurons, are typically divided into a few layers, and each unit in one layer connects with all units in the next layer. Typically, there is no connection between neurons within the same layer. This is a common structure of a feed-forward layered network, although there are many other network structures.

Most layered networks consist of three layers, namely the input layer, hidden layer, and the output layer. Figure 2 illustrates this network structure. It is common to use one or two layers of sigmoid units, and, occasionally, three layers (Mitchell 1997). It is not common to use more than three layers because training times become very long and because networks with three layers of sigmoid units are sufficient to express a rich variety of target functions (Mitchell 1997).

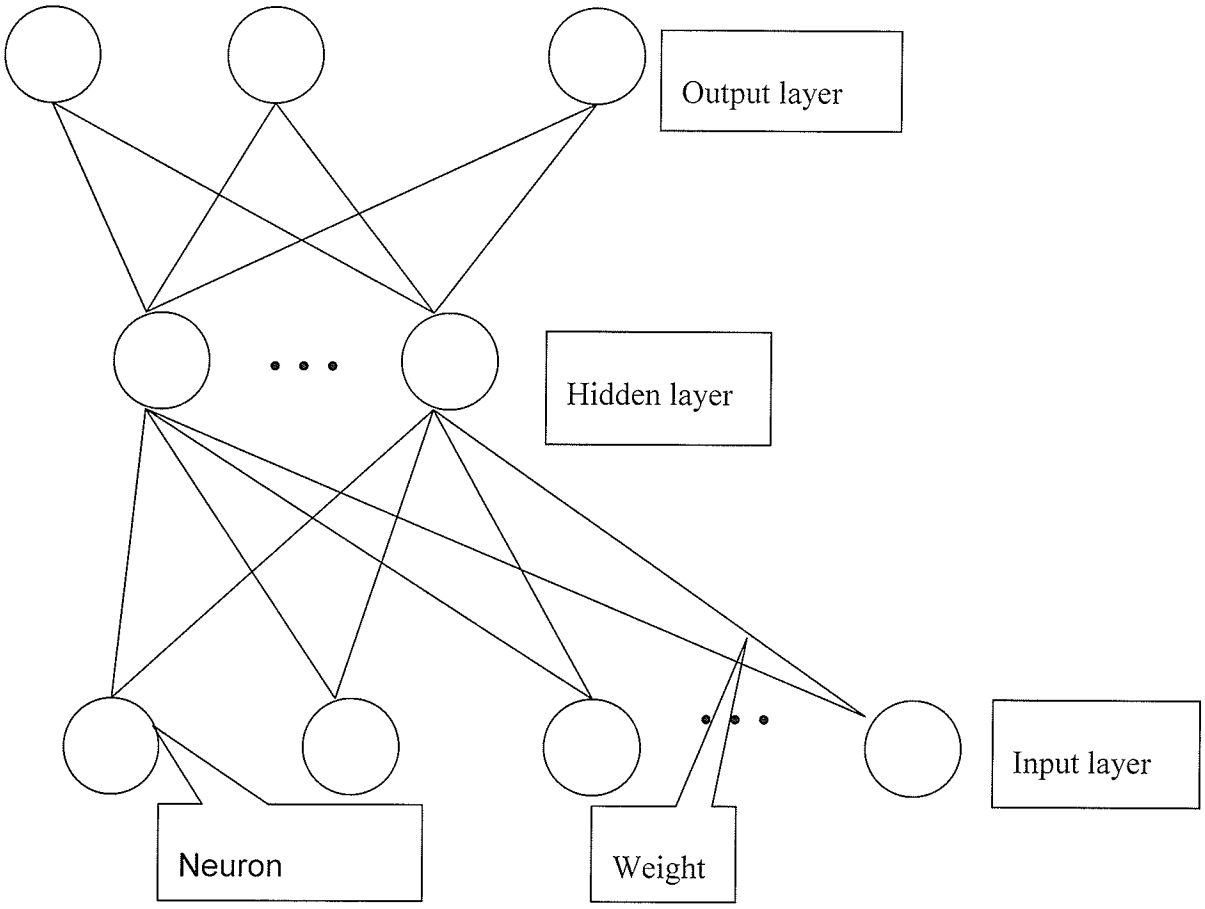

Figure 2. A typical three layer feed-forward network

3.5.2 Neurons

Each neuron has a few weighted inputs, and generates one output. The output is a function on these weighted inputs. Various types of functions can be used to calculate the output in a neuron.

One type of neuron is called a perceptron, which takes a vector of real-valued inputs, and calculates a linear combination of these inputs. It outputs 1 if the result is

greater than some threshold and -1 otherwise. Given inputs $x_1 ... x_n$, this perceptron function is:

$$o(x_1 .....x_n) = \begin{cases} 1, & if\ w_1x_1 + w_2x_2 + ... + w_nx_n \geq w_0 \\ -1, & otherwise \end{cases} \quad (4)$$

where

w_0 = a threshold

w_i = real-valued constant weights

x_n = inputs.

An unthresholded perceptron is a linear unit, whose output is:

$$o(\vec{x}) = \vec{w} \cdot \vec{x} \quad (5)$$

where

$\vec{w}$ = weights vector

$\vec{x}$ = inputs vector.

Another commonly used neuron is sigmoid unit, whose output is:

$$o = g(\vec{w} \cdot \vec{x}) \quad (6)$$

where

g = sigmoid function

The logistic form of sigmoid function is:

$$g(y) = \frac{1}{1 + e^{-y}} \quad (7)$$

The "tanh" form of a sigmoid function is:

$$g(y) = \frac{e^{y} - e^{-y}}{e^{y} + e^{-y}} \quad (8)$$

The logistic and "tanh" forms of a sigmoid function were plotted in figure 3 and 4.

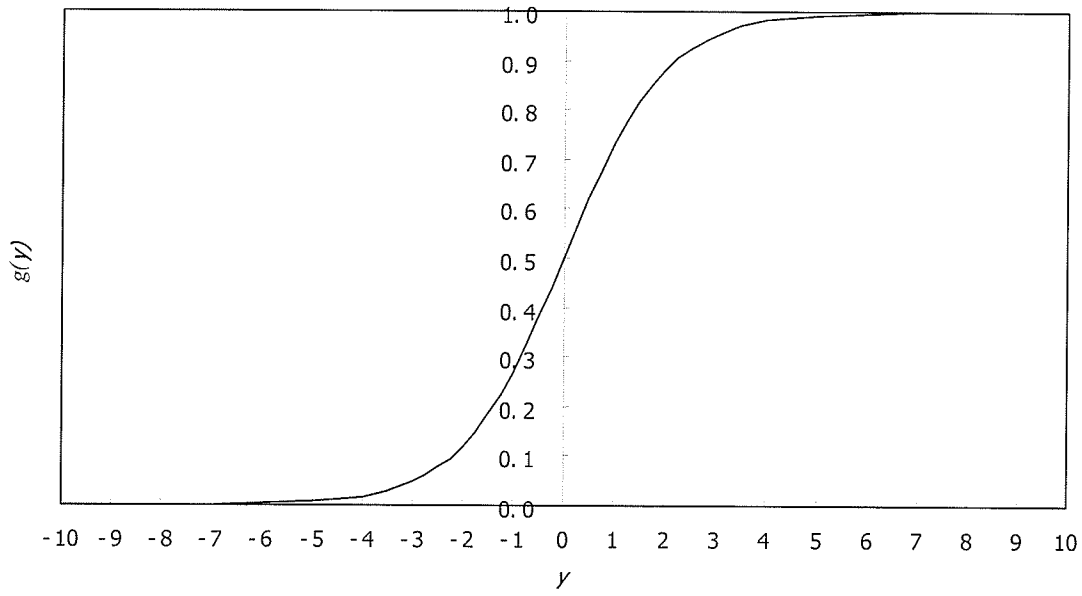

Figure 3. The logistic form of a sigmoid function

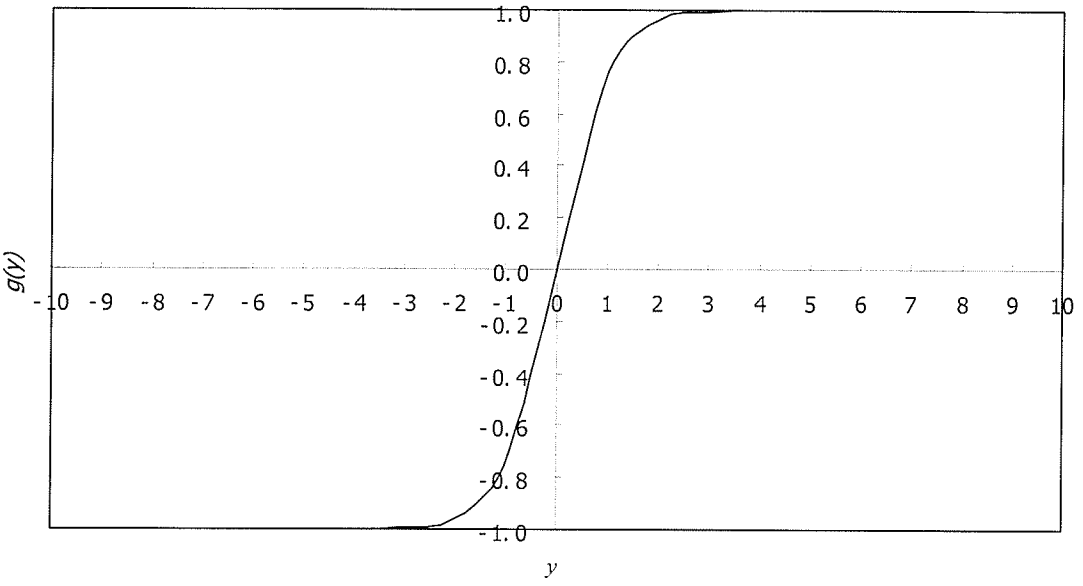

Figure 4. The "tanh" form of a sigmoid function

A sigmoid unit is similar to a perceptron. It first calculates the linear combination of its inputs, and then applies a threshold to the result. However, this threshold is continuous

comparing with a perceptron with a discontinuous threshold. The output of sigmoid function is between 0 and 1 (or between –1 and 1), and increases monotonically with its inputs. The sigmoid unit maps a very large input domain $(-\infty, \infty)$ onto a small range of outputs $(0,1)$ or $(-1,1)$.

3.5.3 Training algorithms

Given a set of inputs, the neural network calculates the output based on the neuron's function. First, each neuron produces an output based on the inputs and their associated weights. Then, each neuron forwards its output to the next connected neurons (i.e., the neurons in the next layer). This process continues, until each neuron in the output layer generates an output. When the actual network output is different from the target output, there exist "errors" in the network and the network needs to be changed. Most neural network algorithms are about how to change the weights. To find the correct weights, we need to present some examples to the neural network. The key idea is to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples (Mitchell 1997). The neural network adjusts their weights based on these known examples, and then generates the output for unknown examples. This process is called a training process. The training process can be viewed as a learning process, in which we teach a network by giving it examples.

To use a neural network in an e-nose for odour measurement, we present the response vector to the network (e.g., each input represents an output from a sensor), and then the network generates outputs. Depending on how we encode odours, these outputs tell us the predicted odour class. However, before the classification process, we need to

train the network by using a number of examples. There are a lot of strategies or algorithms to train a network.

Back-propagation algorithm is a standard gradient descent algorithm and the most commonly used network (Rumelhart et al. 1986). When the predicted output is not the target output, there are "errors". The basic idea of the backpropagation algorithm is to change weights based on the errors, i.e., the errors are back propagated to the network, so that the errors are minimized. The minimization is implemented by adjusting the weights following the derivatives of errors.

The back-propagation algorithm could be viewed as a search through a hypothesis space. The hypothesis space consists of a number of weight vectors. The algorithm is seeking the best weight vector to produce the target output. A hypothesis space is visualized in figure 5 (Mitchell 1997). In this example network, there are two weights, w_0 and w_1. Therefore, the w_0 and w_1 plane represents the hypothesis space. The vertical axis represents the error on training examples. The network searches through the whole hypothesis space, and tries to find the global minimum point on the error surface. This search strategy is called gradient descent.

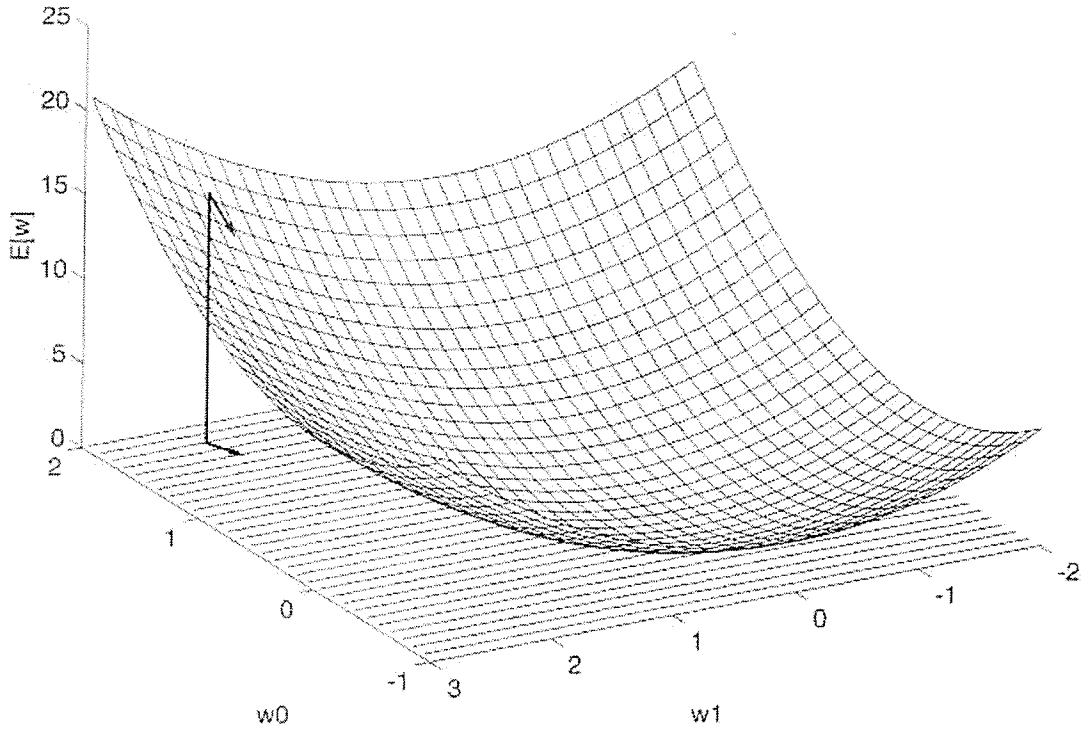

Figure 5. Visualization of search strategy in the backpropagation algorithm (Mitchell 1997)

The local minimum (maximum) problem is a classic problem in A.I. As the figure 5 shows, the neural network is seeking a global minimum (maximum) point on the error surface. In the example shown above, the error surface is parabolic. Therefore, there is only one global minimum point on the error surface. However, when we have more than two weights in a network, the error surface could be very complex. There may be a few local minimum (maximum) points on the error surface (Fig. 6). The backpropagation algorithm is only guaranteed to find some local minimum (maximum) solution, not the global minimum (maximum) (Mitchell 1997). One strategy to handle this problem is restarting the search from a different point on the error surface. Changing the start-up point could avoid sticking into one local minimum/maximum. In practice, it means

initiating the network weights with different values. This strategy still does not guarantee to achieve the global minimum/maximum. However, in most cases, we do not need to find the true global minimum/maximum. What we are searching for is a good enough solution, i.e., a minimum/maximum, local or global, which is good enough to solve our problems.

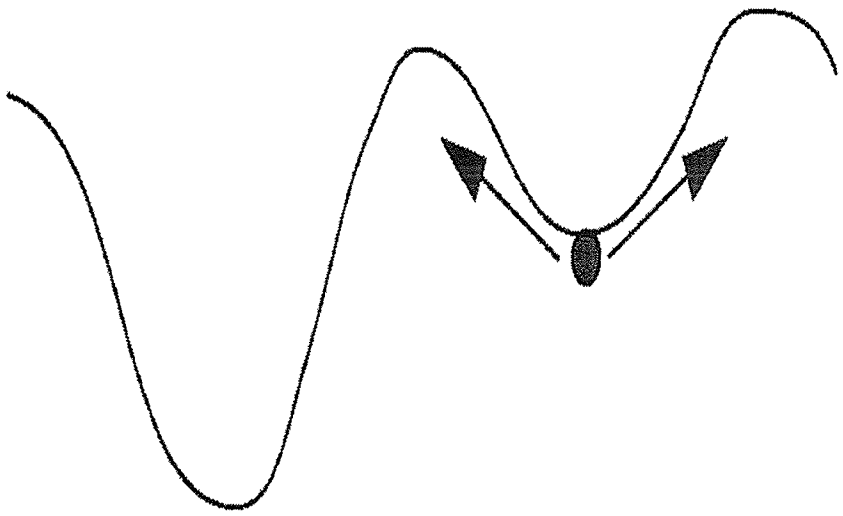

Figure 6. A complex error surface with several local minimum points

To train a neural network, the algorithm needs to run on the same set of training examples a number of times. The training process is an iteration process. Therefore, when to stop the training is an important issue. Overfitting could be a problem, in which a network has high accuracy on training examples, but low accuracy on unknown instances. One possible reason is that the training set may not present the whole population. Overfitting happens when the network fits the training examples too "well" so that it loses its generalization power, i.e., the power to classify unknown instances. There are six common approaches to avoiding overfitting, including model selection, jittering, early stopping, weighting decay, Bayesian learning, combining networks, and

regularisation method using modified performance function (Sarle 2001). The most successful solution to overfitting is early stopping by providing a validation data set to monitor the training process. Figure 7 illustrates the overfitting problem and the early stopping solution. As indicated by the figure, the training should be stopped when the error on the validation set starts to increase. When the validation error increases for a specified number of iterations, the training is stopped, and the weightings and biases at the minimum of the validation error are returned (Demuth and Beale 1994). However, it is necessary to note that in the second figure, the error on the validation set increases for a short time, and decreases constantly. Therefore, determining the right time to stop training is not certain.

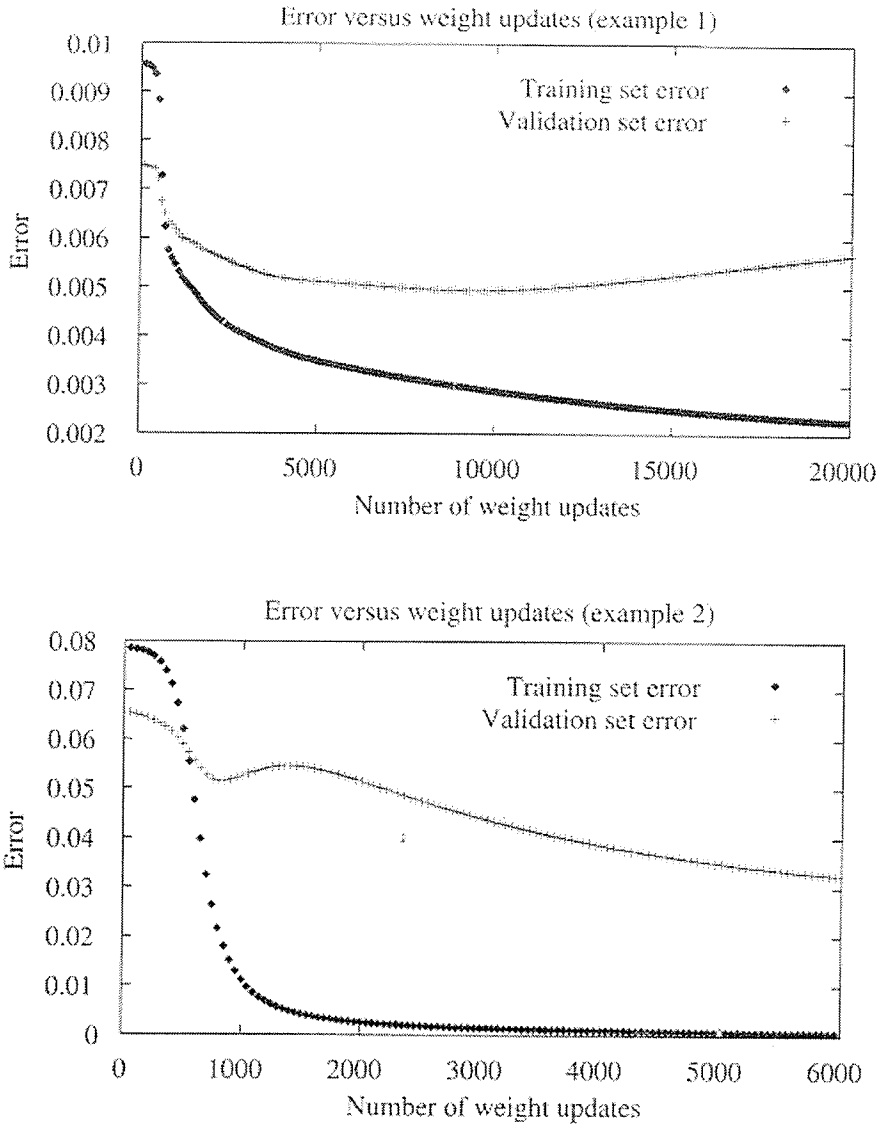

Figure 7. Two examples of overfitting (Mitchell 1997)

3.5.4 Back-propagation neural network for odour classification

Many researchers reported their work on the BPNN used in the e-nose problem domain. Gardner and Hines (1988) first reported the application of a 12-7-5 BPNN to the analysis of alcohols using a 12-element MOS array. Nakamoto et al. (1993) used the BPNN to analyze perfumes and flavours using an array of eight lipid-coated BAW

devices, and correctly classified 28 out of 50 odour samples, i.e., 56%. Gardner et al. (1994) implemented an 18-6-3 BPNN with 18 conducting polymer chemoresistors. This BPNN was used to predict flavours of beer samples and achieved an accuracy of 93%. Although the BPNN performs well in many applications, it suffers from the long training time and being trapped into local minima (Haykin 1994).

Nanto et al. (1995) used a commercially available 5MHz quartz-crystal resonator gas sensor and a BPNN for identification of aromas from wine. They defined 9 parameters, which characterize the transient response curves, as the inputs to the network. They sampled 9 values on the continuous transient response curves. The network had three layers that consisted of nine input units, nine hidden units, and three output units. The network gave 100% accuracy on 14 trials.

Srivastava (2003) used 4 SnO_2 gas sensors to detect volatile organic compounds (VOCs). In this experiment, data were pre-processed by the transformed cluster analysis (TCA). Then a three-layer feed-forward BPNN (4-7-7) with sigmoidal activation function was developed. Learning parameters such as the learning rate and momentum term were optimized as 0.7 and 0.3 by grid experiments for a fixed small number of learning cycles. A large number of experiments with repeated restarts and different weight initializations were performed to achieve the best classification. The network gave 100% classification accuracy, and it was shown that the neural network processing of transformed data had not only better noise tolerance but also could classify the vapours with the array composed of fewer number of sensors as compared to that for the raw (untransformed) data.

3.5.5 Applying neural networks to odour measurement

In odour measurement, concentration and intensity of odours are predicted by neural networks, whereas in odour classification applications, networks predict the type or the class of odours. Compared to odour classification, much fewer studies have been done for odour measurement using neural networks.

Dowdeswell and Payne (1999) first used an e-nose, which consisted of an array of 32 conducting organic polymers, to predict odour intensity. Odour samples from a local wastewater treatment works were analysed by the e-nose, and the e-nose response was used to train a feed-forward BPNN with the inclusion of the humidity as an input node. After training a number of networks, they found that a configuration with 7 hidden nodes gave the best result. Furthermore, they reported that the neural network predicted values to within 6% of those of the sensory panel.

Stuetz et al. (1999) used an electronic nose for assessment of odours from sewage treatment works. In their study, 46 odour samples were collected and sensor data were produced using a Neotronics NOSE (Model D, Neotronics Scientific Ltd.), which incorporated 12 polypyrrole conducting polymer sensors. Canonical correlation analysis was performed, and it showed that there was no universal relationship between the electronic nose responses and odour concentrations for sewage odours from a range of locations within different treatment works. However, when the data were re-analysed using only the lower odour concentration ranges (i.e., threshold odour numbers (TON) values less than 4000 ou/m^3), a clear correlation could be detected between the e-nose output and the olfactometry data.

Hudon and Guy (2000) used an Aromascan A32S (Aromascan) with conducting polymer sensors and an Alpha MOS. Fox 3000 (Alpha M.O.S. SA) with metal oxide sensors, as well as an experimental e-nose made of Taguchi-type tin oxide sensors for measurement of odour intensity of odorous compounds (n-butanol, CH_3COCH_3, and C_2H_5SH) and binary mixtures (n-butanol and CH_3COCH_3). Sensor response data were used to train a BPNN with 6 hidden units and 1 output unit. The whole data set was divided at random into two subsets: the training set (80% of the samples) and the validation set (remaining 20%). Very strong correlations between calculated and measured values were obtained for Aromascan and Alpha MOS. e-noses, with a coefficient of correlation of 0.99 in both cases. The correlation between calculated and measured odour values was much weaker (r=0.84) for the experimental e-nose than for the two commercial instruments. They concluded that 1) e-nose sensor response to samples of equivalent odour intensity could be very different depending on the nature of the compounds present; 2) significant differences in sensitivity to odorous compounds exist among e-nose sensor technologies; 3) for binary mixtures of odorous compounds, a linear regression between odour intensity and averaged sensor response was appropriate to represent the relationship between odour intensity and e-nose measurement when conducting polymer sensors are used, but was inadequate for metal oxide sensors; 4) for binary mixtures of odorous compounds, ANN could be trained to accurately predict odour intensity from commercial e-nose sensor responses. Furthermore, the following general conclusions were made in their study: 1) in order to use an e-nose to measure odour intensity associated with mixtures of odorous compounds, its sensors' ability to respond to the compounds potentially present in the mixtures must first be assessed and 2)

the use of ANN to model the relationship between odour intensity measurement made by sensory analysis and e-nose sensor response was very promising and its applicability to environmental odours should, therefore, be investigated in future studies.

Sohn et al. (2003) quantified odours from piggery effluent ponds using an electronic nose and an ANN. AromaScan, an e-nose consisting of 32 conducting polymer based sensors, was applied on odour samples from five different piggery effluent ponds. A total of 246 samples were collected from five ponds. The data obtained from AromaScan analysis were divided into four equal subsets. The first and the third subsets were used as the training set, the second set as the test set, and the fourth set as the validation set. A back-propagation neural network with two hidden layers was implemented. A tan-sigmoid transfer function was used in the hidden layers and a linear transfer function in the output layer. The training data were pre-processed using the PCA, with the intention of dimensionality reduction and to find a coordinate system that makes the original responses as independent of each other as possible. An early stopping technique was used to get good generalisation performance and to decrease the number of epochs. It was observed that the optimal number of hidden neurons was 20, which gave the regression coefficient R^2 = 0.96 between the predicted odour concentration using artificial neural network and actual odour concentration. The trained artificial neural network model was able to predict the odour concentration of unknown nine air samples with a coefficient of determination R^2 of 0.59.

3.5.6 Comparison of algorithms

There is no doubt that there is no such algorithm that is superior to other algorithms in all problem domains. Actually, the performance of an algorithm depends on

the nature of the problem. Even in the electronic nose applications, there is no agreement on which technique is superior. Since odours are so complex, a pattern recognition technique might work well on a few particular odours, but poorly on other ones.

Several algorithms were compared by Dutta et al. (2003b), in which they implemented an e-nose approach to non-destructive egg freshness determination. In their study, the BPNN, the learning vector quantisation (LVQ), the probabilistic neural network (PNN) and the RBF network were implemented for classifying egg odours. They reported that RBF had the highest classification accuracy (up to 95%) on training data comparing with BPNN (71%), LVQ (92%), and PNN (90%). Besides, the RBF achieved the highest accuracy on test data. Trained on an 866 MHz PC with a Pentium 3 processor, RBF required 170 training iterations (57 minutes), PNN180 iterations (61 minute), LVQ 3000 iterations (2 hours), and BPNN 20,000 iterations (7 hours). According to the experiment results, they believed that a RBF based e-nose provided an attractive means of identifying the freshness of 'commercial' eggs.

Furthermore, Dutta et al. (2003c) analyzed tea quality using an array of 4 metal oxide sensors based electronic nose. Seven hundred fifty data vectors for 5 different tea samples were collected, and PCA, fuzzy C means (FCM), and the self-organizing map (SOM) were used to explore clustering. Then they used 4 networks to predict tea quality. BPNN network (6-5-5) achieved an accuracy of 88%, LVQ 89%, PNN 94%, and RBF 100%, when environmental factors (temperature and humidity) were used as extra input nodes.

4. MATERIALS AND METHODS

4.1 Development of a neural network system- ENBrain

4.1.1 Programming language

An artificial neural network was implemented using the Java programming language. Java probably is the most popular programming language in recent years. Although Java has many advantages in developing programs, e.g., writing less and better codes more easily and quickly, and distributing software more easily, The most attractive issue for this study is its cross-platform ability. The platform is loosely defined as a combination of hardware and system software, which is typically referred as the operating system. However, a platform can be a non-PC system, e.g., cell phones, digital phones, security systems, and stereo systems. With a cross-platform language, it becomes easier to develop applications for non-PC use. In future studies, we can build a portable e-nose by using a few electronic chips instead of a computer. Moreover, we can build a distributive e-nose system, in which data can be communicated by different types of devices, e.g., cell phones, computers, and control systems.

To execute a Java program, the platform independent byte codes have to be interpreted by the Java virtual machine, which is actually an interpreter translating these byte codes into native machine codes of the host platform. Figure 8 shows the procedure of running a Java program from its source codes. Java 2 Platform, Standard Edition, v 1.5.0 Beta 1 release, which includes the Sun Java virtual machine, was used to implement the ENBrain.

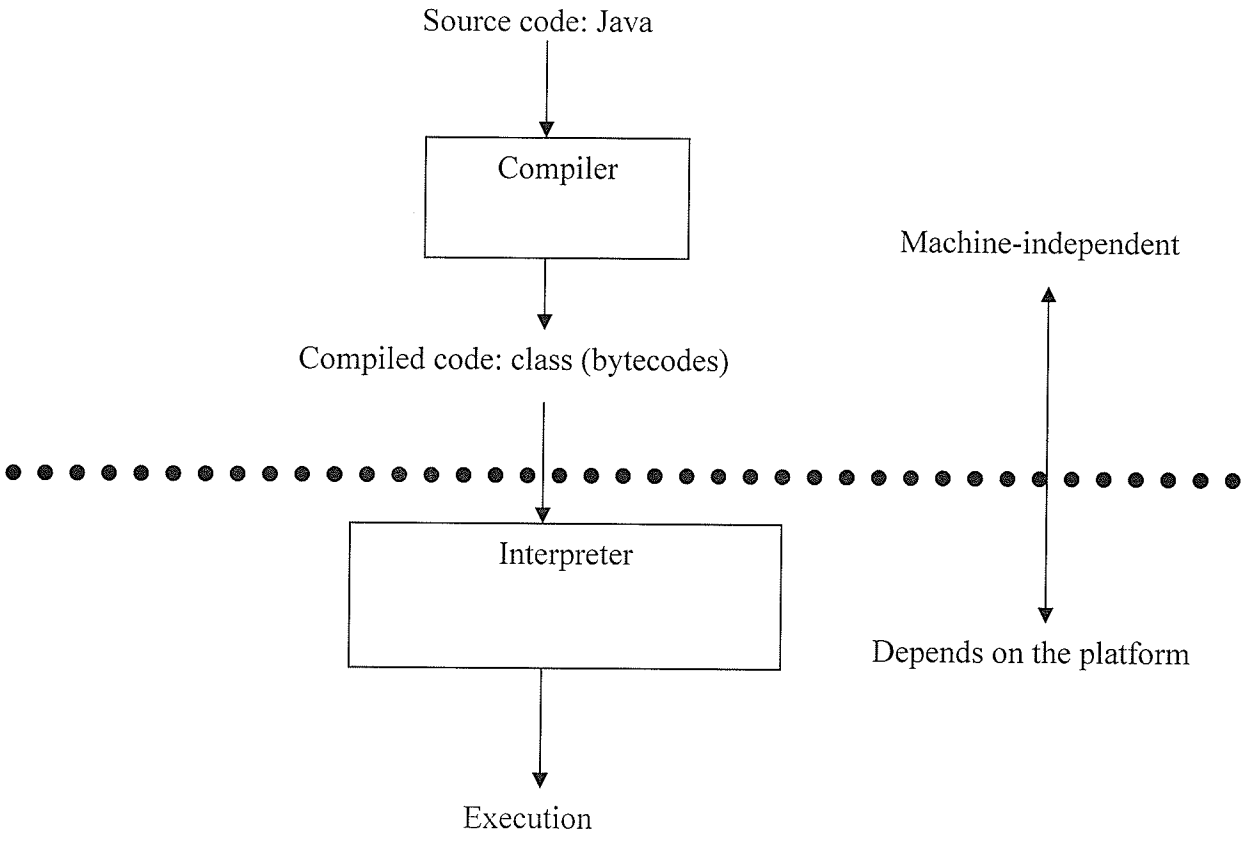

Figure 8. The mechanism of platform independent for JAVA technology

(http://www.iua.upf.es/~dani/java1.htm)

4.1.2 Code description

The Java program for neural networks in this thesis is a GUI program, which can be run on different operating systems. The author of the thesis coded this program without using any developed JAVA component of artificial neural networks. Within the program, multiple networks can be built up. Each network holds a thread in the program. Therefore, multiple networks can be trained simultaneously. After training, the program can save the trained network into a computer file, and then the saved file can be loaded

by the program in the future. Some essential source codes of the program are shown in appendix A.

4.1.3 Structure of neural networks in ENBrain

The neural network, which is illustrated in figure 9, consists of three layers of neurons. The first layer is the input layer, in which there are 12 neurons corresponding to 12 gas sensors in the e-nose. The second layer is the hidden layer. The number of neurons in this layer is predetermined. The third layer is the output layer. The number of output neurons depends on how many classes we are trying to discriminate. Each neuron connects to all neurons in the next layer. Neurons within the same layer are not connected to each other. Each neuron has a few inputs and generates only one output.

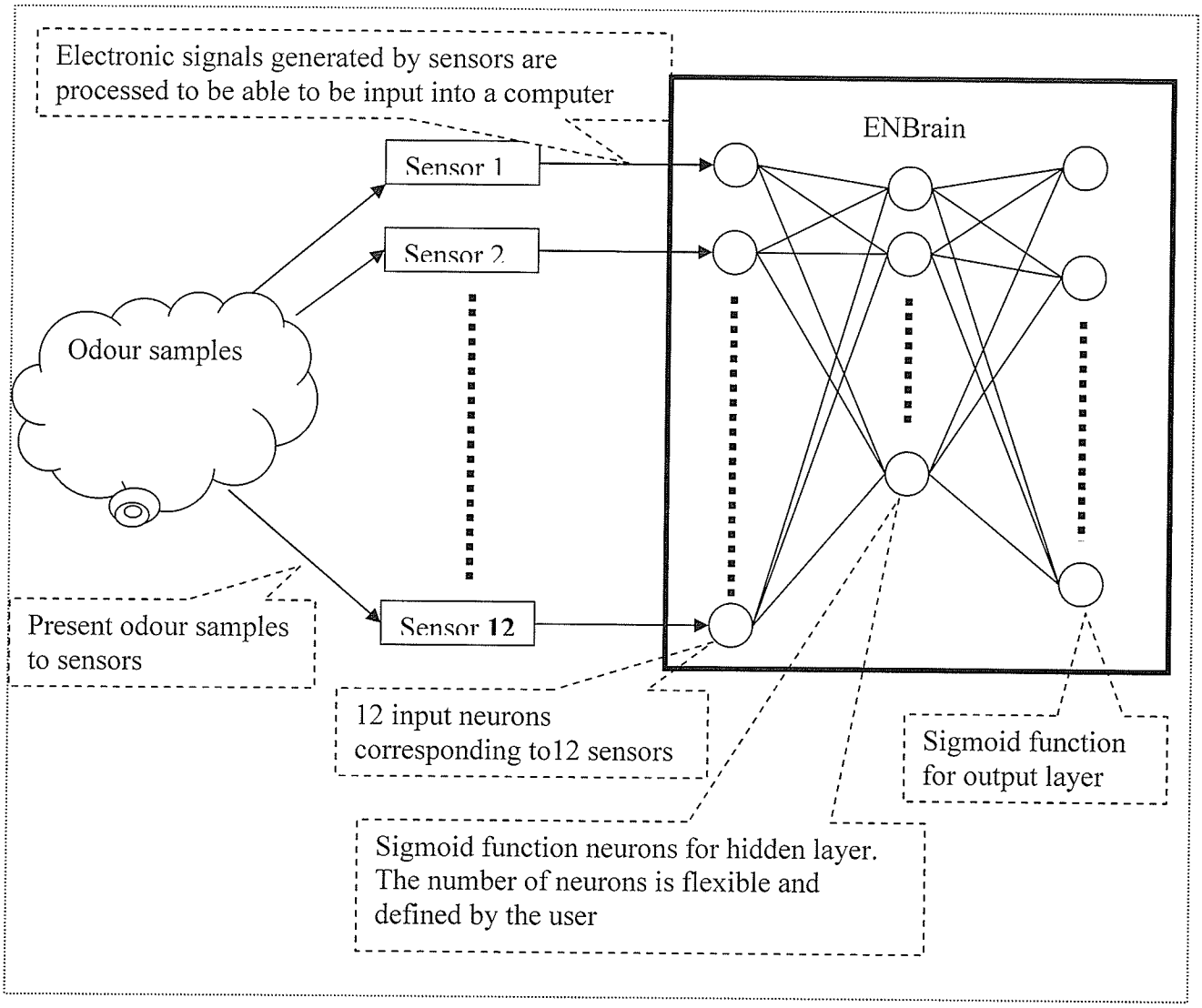

Figure 9. Structure of neural networks in ENBrain

The neurons in the input layer are different from the neurons in the hidden and output layer. Each input neuron has only one input, which is the output of a gas sensor. They do nothing to the inputs from gas sensors, but simply forward the gas sensor signals to the next layer. In other words, the output from each input neuron is the same as its input. On the other hand, each neuron in the hidden and output layer has a few inputs, and generates one output. The number of inputs for each neuron is equal to the number of neurons in the previous layer. Each neuron produces an output based on these inputs, and

forwards the output to the next layer, i.e., the output is forwarded as the input to each neuron it connects in the next layer.

Sigmoid units were used as the hidden and output neurons. Linear units and perceptron discussed early in this thesis are not suitable to predict odour intensity. Mitchell (1997) stated that multiple layers of cascaded linear units still produce only linear function. In electronic noses for odour measurement, sensors generally produce non-linear data, which means the data are not linearly separable, i.e., it is difficult to discriminate data vectors (patterns or fingerprint) of odours by using linear models. Perceptron is non-linear; it could be a solution to non-linear case. However, considering that we use the gradient descent in training, which means we need to find the derivative of the training error, perceptron is not suitable since its output is undifferentiable. What we need is a neuron whose output is a non-linear function of its inputs, and its output is a differentiable function of its inputs (Mitchell 1997). The sigmoid neuron meets these requirements, and therefore was selected.

Furthermore, logistic sigmoid functions were used for hidden and output units in ENBrain. A logistic sigmoid function $g(y) = \frac{1}{1+e^{-y}}$ (eq. 7) is equivalent to a "tanh" sigmoid function $g(\tilde{y}) = \frac{e^{\tilde{y}} - e^{-\tilde{y}}}{e^{\tilde{y}} + e^{-\tilde{y}}}$ (eq. 8), if we apply a linear transformation $\tilde{y} = y/2$ to the input and a linear transformation $\tilde{g} = 2g - 1$ to the output (Bishop 1995). Thus a neural network whose hidden units using the "tanh" sigmoid function is equivalent to one with hidden units using the logistic sigmoid function but having different values for the weights and biases (Bishop 1995). Logistic sigmoid functions were used in ENBrain, since their input range (approximately from –5 to 5) corresponding to a drastic change in

output is wider than the one (approximately from –3 to 3) of "tanh" sigmoid functions.

4.1.4 Method of training

Training of the ENBrain refers to updating weights in the network. Given inputs that are sensor responses for a number of odour samples, the network forwards these inputs to the hidden layer and finally the output layer to produce a network output. Then the network back propagates errors between the target output and the network output. Based on the errors, the network adjusts its weights to minimize errors. This process iterates on the same set of odour samples, which is called the training set, until it satisfies some predefined criterion.

Training is conducted as incremental (stochastic) gradient descent and the training error is calculated as:

$$E_d(\vec{w}) = \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2 \qquad (9)$$

where

d = training instance

t_k = target value of the output unit k for instance d

o_k = actual value of the output unit k for instance d.

Equation 9 is the square of an individual training sample. Therefore, during each training epoch, the incremental gradient descent calculates the training error for each training sample, and updates weights following the error calculation of each individual sample. The incremental gradient descent can sometimes avoid falling into local minima, because

it updates weights with respect to individual training examples rather than the whole training set.

The key idea of back-propagation is to use the derivative of the training error to guide the gradient descent in searching for a minimum in a hypothesis space. Since our network uses the sigmoid function and stochastic gradient descent, the rule for updating each weight between output unit j and hidden unit i can be derived from equation 9 as:

$$w_{ji} = w_{ji} + \Delta w_{ji} \tag{10}$$

$$\Delta w_{ji} = -\eta \frac{\partial E(\vec{w})}{\partial \vec{w}} = \eta (t_j - o_j) o_j (1 - o_j) x_{ji} \tag{11}$$

where

w_{ji} = the weight between output unit j and hidden unit i

x_{ji} = the input from hidden unit i to output unit j

o_j = the output computed by unit j

t_j = the target output for unit j

η = learning rate.

η is a positive constant called the learning rate, which determines the step size in the training progress. By making the value of η (the gradient descent step size) sufficiently small, stochastic gradient descent can be made to approximate the true gradient descent arbitrarily closely (Mitchell 1997). The range of η is between 0 and 1, and a typical value of 0.3 was used in this study.

For weights between the hidden units and the input units, the indirect ways that they influence the network outputs and hence the training error must be taken into

account. Therefore, the updating rule for each weight between hidden unit j and input unit i can be derived as:

$$\Delta w_{ji} = -\eta \frac{\partial E(\vec{w})}{\partial \vec{w}} = \eta o_j (1 - o_j) \sum_{k \in outputs} w_{kj} o_k (1 - o_k)(t_k - o_k) \qquad (12)$$

where

o_j = the output computed by unit j

w_{kj} = the weight between output unit k and hidden unit j

o_k = the output computed by unit k

t_k = the target output for unit k

η = learning rate.

This rule sums the errors for each output unit influenced by the individual hidden unit, weights each error by the weight between the output unit and the hidden unit. This weight characterizes the degree that an individual hidden unit is responsible for the error in the output unit connected to it. Mitchell (1997) discussed how the above equations were derived in more details.

Mitchell (1997) described the stochastic gradient descent version of the back-propagation algorithm for feed-forward networks containing two layers of sigmoid units as follows.

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.

- Initialize all network weights to small random numbers (e.g. between -.05 and .05).
- Until the termination condition is met, Do
  - For each $\langle \vec{x}, \vec{t} \rangle$ in training examples ($\vec{x}$ is a vector consists of all input unit values to the network for an individual training example. $\vec{t}$ is a vector consists of all target values from the network output units corresponding to an individual training example.), Do

    Propagate the input forward through the network:

    1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

    Propagate the errors backward through the network:

    2. For each network output unit k, calculates its error term δ_k

$$\delta_k \leftarrow o_k(1-o_k)(t_k - o_k)$$

    3. For each hidden unit h, calculates its error term δ_h

$$\delta_h \leftarrow o_h(1-o_h) \sum_{k \in outputs} w_{kh}\delta_k$$

    4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

    where $\Delta w_{ji} = \eta \delta_j x_{ji}$ (This equation actually summaries equation 11 and equation 12.)

Many variations of the back-propagation algorithm have been developed. Adding momentum to the weight-updating rule, which is used widely for neural network

applications, is employed in ENBrain. The weight updating equation with considering momentum used in ENBrain is given as follows (Mitchell 1997):

$$\Delta w_{ji}(n) = \eta \delta_j x_{ji} + \alpha \Delta w_{ji}(n-1) \tag{13}$$

where

$\Delta w_{ji}(n)$ = the weight update performed in the n'th iteration

$\Delta w_{ji}(n-1)$ =the weight update performed in the (n-1)'th iteration

α = a constant called the momentum and $0 \leq \alpha < 1$.

The momentum makes the weight updating on the *n*th iteration depend partially on the (n-1)th weight update iteration. It tends to keep the gradient descent search in the same direction from one iteration to the next. It also can sometimes skip local minima. While the error surface is flat, momentum can avoid stopping search and gradually increase the step size of the search in regions where the gradient is unchanging, thereby accelerating convergence. The human designer (i.e., the author of the thesis) decided the value of momentum, and a typical value of 0.2 was selected in this study (Baltes 2003).

4.1.5 Networks Parameters

A commercial electronic nose Alpha MOS Fox 3000 (Alpha MOS, Toulouse, France) with an array of twelve gas sensors was used to collect data for training and validating the networks. Each of the 12 sensors generates a numeric signal (relative change in electrical resistance) when subjected to an odour sample. Therefore the input layer of the network consists of 13 input units, in which 12 input units correspond to 12 sensor outputs and an additional constant input whose value is 1. The additional constant input connects to all hidden units and is bias of these hidden units. Generally, the bias

determines the position of the classification boundary in x-space (i.e., the input space), where the classification boundary is a (d-1)-dimensional hyperplane in d-dimensional x-space (Bishop, 1995). In our case of using sigmoid function, the bias determines the location of the sigmoid function in terms of its distance from the origin (i.e., all input units are 0).

The number of units in the output layer depends on the way we encode the network output, which in turn depends on how odour intensity is measured. In this study, odour intensity is measured with an eight-point scale and a 0-200 magnitude estimation. Each output unit is used to represent one intensity level. Therefore, there are 8 output units for the 8-point scale and 200 output units for the 0-200 odour intensity scale. We applied a 1-of-n output encoding, in which the highest-valued output is taken as the network prediction.

For odour concentration prediction, we applied the similar output encoding. However, the odour concentration is a continuous variable, i.e., there are infinite scales for odour concentration. To apply the 1-of-n output encoding, we divided the whole range of concentration into sub-ranges, and each could be represented by an output unit. In this study, n-butanol at concentrations from 0 to 20000 ppm was used. Since human perception of odour intensity is related to odour concentration by the logarithm function, we used logarithm of concentration to map the range of concentrations. We defined 50 output units for the network, which mapped 50 sub-ranges of odour concentration. The first unit represents the logarithm of concentration from 0 to 0.1 (i.e., $0 - 10^{0.1}$ *ppm*), the second 0.1-0.2 ($10^{0.1} - 10^{0.2}$ *ppm*), and so on. In other words, the increment size between output units is 0.1. Furthermore, the networks are flexible in decreasing the increment

size by adding more output units for higher resolutions. For example, defining an increment size of 0.05 would require 100 output units, and result in a resolution of $10^{0.05}$ *ppm*.

There is no "formula" to calculate the optimal number of hidden units. Decisions are commonly made by the rule-of-thumb. Generally, a number somewhere between the input layer size and the output layer size is recommended (Blum 1992; Swingler 1996; Bery and Linoff 1997; Boger and Guterman 1997). For example, the number of hidden units is recommended as 10% of the number of input units. Nevertheless, the performance of networks in predicting unknown sensor data depends upon the number of hidden units. Too many hidden units will result in the network memorizing input patterns. On the other hand, too few hidden units means that network will not have the needed flexibility to model the high order properties of the data. Both cases lead to poor generalization of networks. Searching for an optimal number of hidden units is performed using a trial and error process.

Furthermore, there is no rule for setting the learning rate and momentum. If these parameters are set too small, the network requires more time to converge, which results in more training time. However, if these parameters are set too large, the network might skip global minima during training. Therefore, finding optimal parameters for networks also depends on a trial of error process. A number of networks were experimented in this study to find the optimal number of hidden units, which is the essential issue about network performance. Furthermore, different initial weights were experimented since ENBrain randomized the initial weights for each new network. The

learning rate and momentum were set to typical values. Section 4.3 discussed the detailed experiment method.

4.2 E-nose data for training and validation

The data used in this study was collected by York (2005). The networks were trained on a training set that consisted of 32 data vectors, and were tested on a set of 48 data vectors. Both training and testing data sets were collected with a commercial electronic nose Alpha MOS Fox 3000 (Alpha MOS, Toulouse, France). The e-nose was equipped with 12 metal oxide sensors and n-butanol was used as the test odorant. In York's (2005) study, the n-butanol samples prepared in vials were placed into a sampling tray for the auto-sampler and held at room temperature during the sampling process. Samples were transferred sequentially to the incubator/heating block and gently agitated at constant rpm/directional cycle. The headspace gas was then drawn into a syringe and transferred to the injection port of the electronic nose. Sensor response data were collected for 120 s, followed by a 1080 s delay before injection of the next sample. The carrier gas at a flow rate of 150 mL/ min was a mixture of oxygen and nitrogen (20%±1% O_2 and $O_2 + N_2 > 99.95\%$). The specific test conditions were: incubation temperature = 50°C; incubation time = 5 minutes (using standard agitation conditions); and sample size = 1 mL in 10 mL vial. Concentrations of n-butanol were 5 to 20000 *ppm*. Each sample had four replications, which were four complete runs of the sample set. The order (from low concentrations to high concentrations) was set for the sample set and then each sample within the set was tested. This was done so that any minor variation, which may occur in the sensors, happened to all of the samples, not just to the four repeats of one sample.

The intensities of n-butanol samples were assessed by trained human panels (York 2005). After sniffing an n-butanol sample, each panellist rated the sample in a 0-200 scale, with 0 being no odour and 200 the strongest odour. A panel with 20 human panellists was set up to assess intensities of eight standard n-butanol concentrations. The panellists assessed n-butanol samples in two sessions. In each session, the geometric means of 20 odour intensities assessed by 20 panellists were calculated as the intensities for this session, and then the geometric means of two sessional results were calculated as the final assessed intensities, which were used to train neural networks. Another panel with 12 panellists was set up to assess odour intensities of 12 n-butanol concentrations, which were different from the eight standard concentrations. The geometric means of 12 panellists' assessments were calculated as the panel's assessment. Furthermore, each n-butanol concentration was assessed by this panel three times, and the geometric means of the three replications were calculated as the final odour intensities that were used to test neural networks.

An eight-point odour intensity referencing scale is often used to assess odours (Zhang et al. 2001). The scale is based on a set of standard n-butanol and water mixtures (Table 1) (ASTM E544 1975). Odour intensity data used in this study were collected according to a 0-200 Labelled Magnitude Scale (LMS) (York 2004). The correlation between the 8-point and 0-200 scales were investigated.

Table 1. Eight-point odour intensity referencing scale

Intensity level	n-butanol concentration in air (*ppm*)	Annoyance sacle
0	0	No odour
1	12	Not annoying
2	24	A little annoying
3	48	A little annoying
4	96	Annoying
5	194	Annoying
6	388	Very annoying
7	775	Very annoying
8	1550	Extremely annoying

Based on the assessments of 20 individual human panellists, the following regression equation was found with $R^2 = 0.76$ (fig. 10):

$$y = 11.338x \tag{14}$$

where

y = odour intensity in 0-200 intensity scale

x = odour intensity in 8-point intensity scale.

This equation was used to convert odour intensity from the 0-200 scale to the 0-8 scale for training the ENBrain.

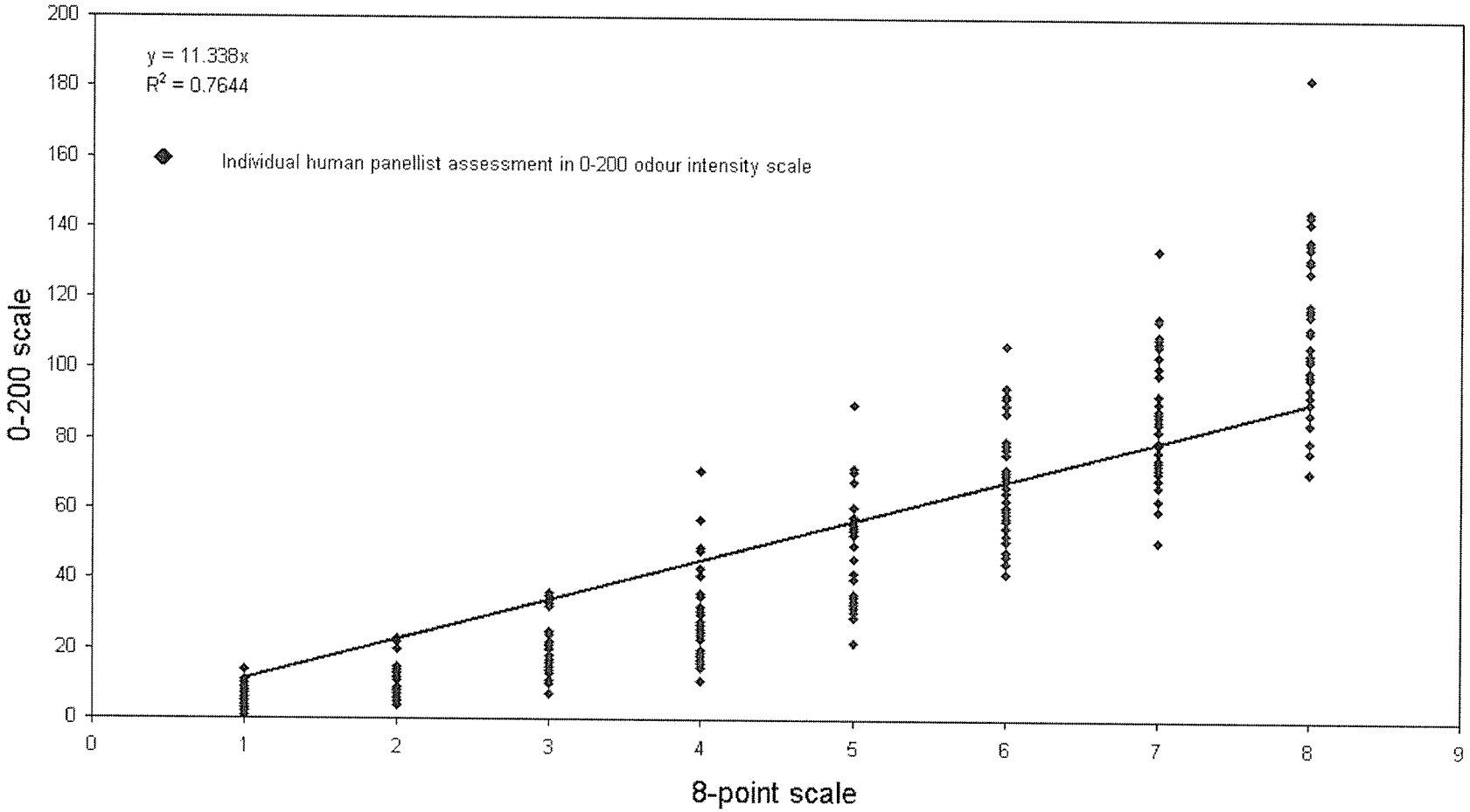

Figure 10. A linear regression between the 8-point scale and the 0-200 scale

Tables 2 and 3 describe the training and testing data set. The sensor response for each odour sample is stored in a text file on the computer. In such a text file, En_code is the identification for each sample replication. Since there are 12 gas sensors in the e-nose, one data file for each EN_code was created to store 12 numeric sensor responses (Appendix D). The convenience of making each sample data as one data file is that a particular sample could be added or removed from the sample set without modifying other data files.

Table 2. Training n-butanol samples without actual sensor response data

EN_code	Concentration (ppm)	$\log_{10}$ of concentration	Assessed Intensity [a]
F1_10	120	2.08	4.61
F1_18	120	2.08	4.61
F1_2	120	2.08	4.61
F1_26	120	2.08	4.61
F2_11	240	2.38	9.25
F2_19	240	2.38	9.25
F2_27	240	2.38	9.25
F2_3	240	2.38	9.25
F3_12	480	2.68	17.07
F3_20	480	2.68	17.07
F3_28	480	2.68	17.07
F3_4	480	2.68	17.07
F4_13	960	2.98	28.40
F4_21	960	2.98	28.40
F4_29	960	2.98	28.40
F4_5	960	2.98	28.40
F5_14	1940	3.29	43.37
F5_22	1940	3.29	43.37
F5_30	1940	3.29	43.37
F5_6	1940	3.29	43.37
F6_15	3880	3.59	63.97
F6_23	3880	3.59	63.97
F6_31	3880	3.59	63.97
F6_7	3880	3.59	63.97
F7_16	7750	3.89	84.72
F7_24	7750	3.89	84.72
F7_32	7750	3.89	84.72
F7_8	7750	3.89	84.72
F8_17	15550	4.19	108.46

[a] geometric mean of 20 measurements in 0-200 scale

F8_25	15550	4.19	108.46
F8_33	15550	4.19	108.46
F8_9	15550	4.19	108.46

Table 3. Testing n-butanol samples without actual sensor response data

En_Code	Concentration (ppm)	$\log_{10}$ of Concentration	Assessed Intensity[a]
C01	5	0.70	1.58
C02	5	0.70	1.58
C03	5	0.70	1.58
C04	5	0.70	1.58
I01	10	1.00	2.41
I02	10	1.00	2.41
I03	10	1.00	2.41
I04	10	1.00	2.41
D01	25	1.40	8.56
D02	25	1.40	8.56
D03	25	1.40	8.56
D04	25	1.40	8.56
K01	50	1.70	9.52
K02	50	1.70	9.52
K03	50	1.70	9.52
K04	50	1.70	9.52
A01	100	2.00	15.04
A02	100	2.00	15.04
A03	100	2.00	15.04
A04	100	2.00	15.04
H01	250	2.40	20.89
H02	250	2.40	20.89
H03	250	2.40	20.89
H04	250	2.40	20.89
B01	500	2.70	28.84
B02	500	2.70	28.84
B03	500	2.70	28.84
B04	500	2.70	28.84
L01	1000	3.00	34.69
L02	1000	3.00	34.69
L03	1000	3.00	34.69
L04	1000	3.00	34.69
E01	2500	3.40	54.41
E02	2500	3.40	54.41
E03	2500	3.40	54.41
E04	2500	3.40	54.41
J01	5000	3.70	63.04
J02	5000	3.70	63.04
J03	5000	3.70	63.04
J04	5000	3.70	63.04
G01	10000	4.00	71.62
G02	10000	4.00	71.62
G03	10000	4.00	71.62

[a] geometric mean of 12 measurements in 0-200 scale

G04	10000	4.00	71.62
F01	20000	4.30	90.49
F02	20000	4.30	90.49
F03	20000	4.30	90.49
F04	20000	4.30	90.49

4.3 Training and testing networks

Networks were trained on the training set that consisted of 32 data vectors, and were tested on the testing set that consisted of 48 data vectors. A number of networks, which have the different number of hidden units, were trained and tested in order to explore the relationship between the number of hidden units and the network performance. For each number of hidden units, 3 replications (i.e., 3 networks with the same number of hidden units) were built up. The only difference between the 3 replications is different initial weights, i.e., different start points for searching a best weight set in a hypothesis space consisting all possible weight sets. Therefore, building up 3 replications can avoid the unusual case that the network fails because of a bad start point. We define the 3 replications as a "group" of networks for convenience.

For odour concentration predictions, 30 groups of networks, from 1 hidden unit to 30 hidden units, were trained and tested. Since each group consisted of 3 replications, 90 networks were built up for the odour concentration experiment. For odour intensity predictions, 3 groups of networks, which had 10, 20, and 30 hidden units, respectively were experimented for each odour intensity scale, i.e., a total of 9 networks were implemented for the 8 point scale and another 9 networks for the 0-200 scale. Since both concentration networks and intensity networks were trained on the same data and had similar network structure, the only difference was the number of output units. Only 3

groups of networks were explored for intensity predictions and an increase step size of 10 was selected for adding hidden units. In overall, 108 networks were implemented, and the evaluation methods and results are discussed in the following section.

5. RESULTS AND DISCUSSION

5.1 Discussion of sensor responses

Responses of gas sensors to n-butanol concentration are shown in figures 11 and 12, for training and testing data sets, respectively. A sensor response was is the relative change in electrical resistance, i.e., $(R-R_0)/R_0$, where R is resistance of a sensor and R_0 is baseline. Sensors 2, 3, 4, 5, and 6 generated negative output (decrease in resistance) while sensor 1, 7, 8, 9, 10, 11, and 12 generated positive output (increase n resistance). Sensors 2, 3, 4, 5, and 6 were more sensitive to concentration changes than others. In other words, sensors 2, 3, 4, 5, and 6 contributed more to generating differentiable sensor response patterns. The sensor responses were ranged from –3 to 1, which is a small range considering the magnitude of the width of sigmoid functions used in ENBrain. Therefore, there is no need to perform auto-scaling before inputting sensor responses into ENBrain. There was no trend showing that the sensor responses would level off after 20000 *ppm*.

Correlations between gas sensors were calculated by SAS (SAS Institute Inc., Cary, USA) using oth training data and testing data. High correlations (|correlation coefficient| $\geq$ 0.92) were found between gas sensors (table 4). These high correlations indicated that the sensor responses should be able to be normalized and then concentration independent patterns would be obtained. Using more than one gas sensor did not seem to contribute much in discriminating different n-butanol concentrations. However, the goal of this study was to develop neural networks for complex livestock odours. N-butanol is a simple odorant and was used here only as the first step of testing the neural networks. All data were retained in the following discussion without being normalized.

Table 4. Correlations between gas sensors
(S1 represents sensor 1, S2 represents sensor 2, etc.)

Sensors	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12
S1	**1.00**	-0.98	-0.99	-0.98	-0.98	-0.92	0.99	0.99	0.99	0.99	0.99	0.99
S2	-0.98	**1.00**	0.99	1.00	1.00	0.97	0.99	-1.00	-1.00	-1.00	-0.99	-0.99
S3	-0.99	0.99	**1.00**	0.99	0.99	0.94	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00
S4	-0.98	1.00	0.99	**1.00**	1.00	0.98	-0.99	-1.00	-0.99	-0.99	-0.99	-0.99
S5	-0.98	1.00	0.99	1.00	**1.00**	0.97	0.99	-1.00	-0.99	-1.00	-0.99	-0.99
S6	-0.92	0.97	0.94	0.98	0.97	**1.00**	0.94	-0.96	-0.95	-0.96	-0.94	-0.94
S7	0.99	-0.99	-1.00	-0.99	-0.99	-0.94	**1.00**	1.00	1.00	1.00	1.00	1.00
S8	0.99	-1.00	-1.00	-1.00	-1.00	-0.96	1.00	**1.00**	1.00	1.00	1.00	1.00
S9	0.99	-1.00	-1.00	-0.99	-0.99	-0.95	1.00	1.00	**1.00**	1.00	1.00	1.00
S10	0.99	-1.00	-1.00	-0.99	-1.00	-0.96	1.00	1.00	1.00	**1.00**	1.00	1.00
S11	0.99	-0.99	-1.00	-0.99	-0.99	-0.94	1.00	1.00	1.00	1.00	**1.00**	1.00
S12	0.99	-0.99	-1.00	-0.99	-0.99	-0.94	1.00	1.00	1.00	1.00	1.00	**1.00**

The PCA procedure in MATLAB (Release 13, The MathWorks Inc., Natick, USA) was employed to analyze both training data and testing data, It was found that first two principal components accounted for 99.90% of the variance in the training data set, while two principal components accounted for 99.91% of the variance in the testing data set (figs. 15 and 16).

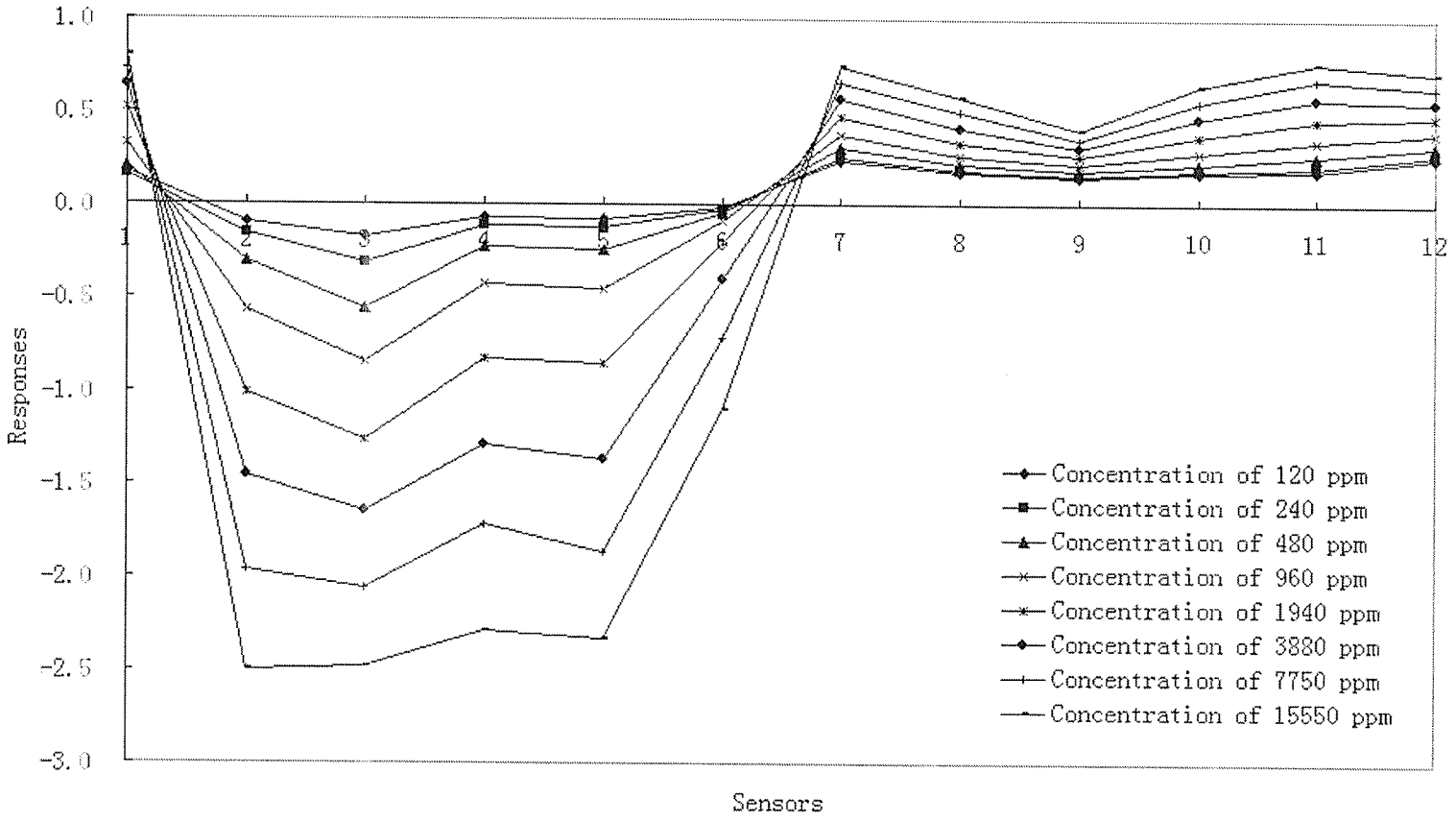

Figure 11. Sensor responses vs. sensors for training data

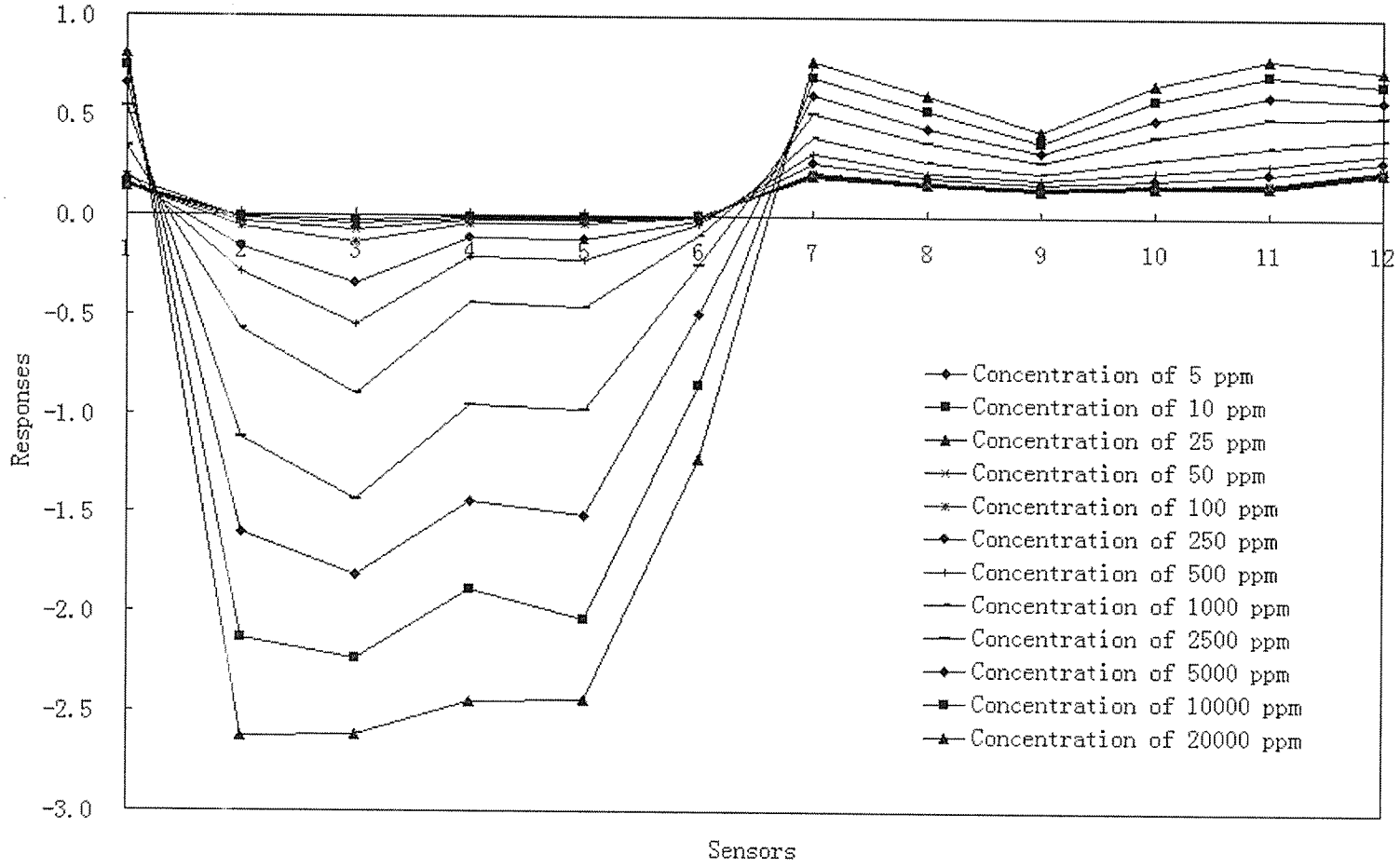

Figure 12. Sensor responses vs. sensors for testing data

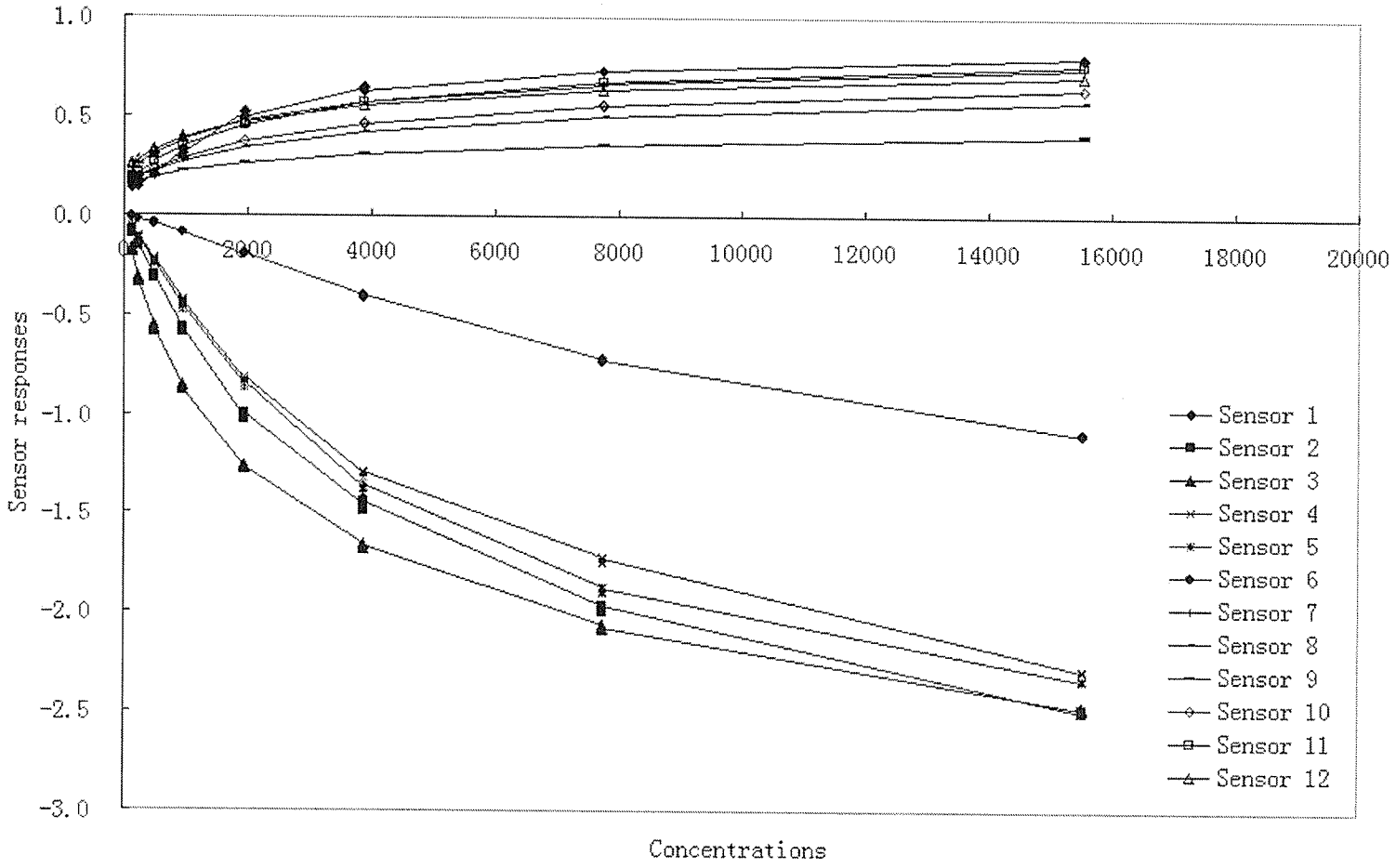

Figure 13. Sensor responses vs. concentrations for training data

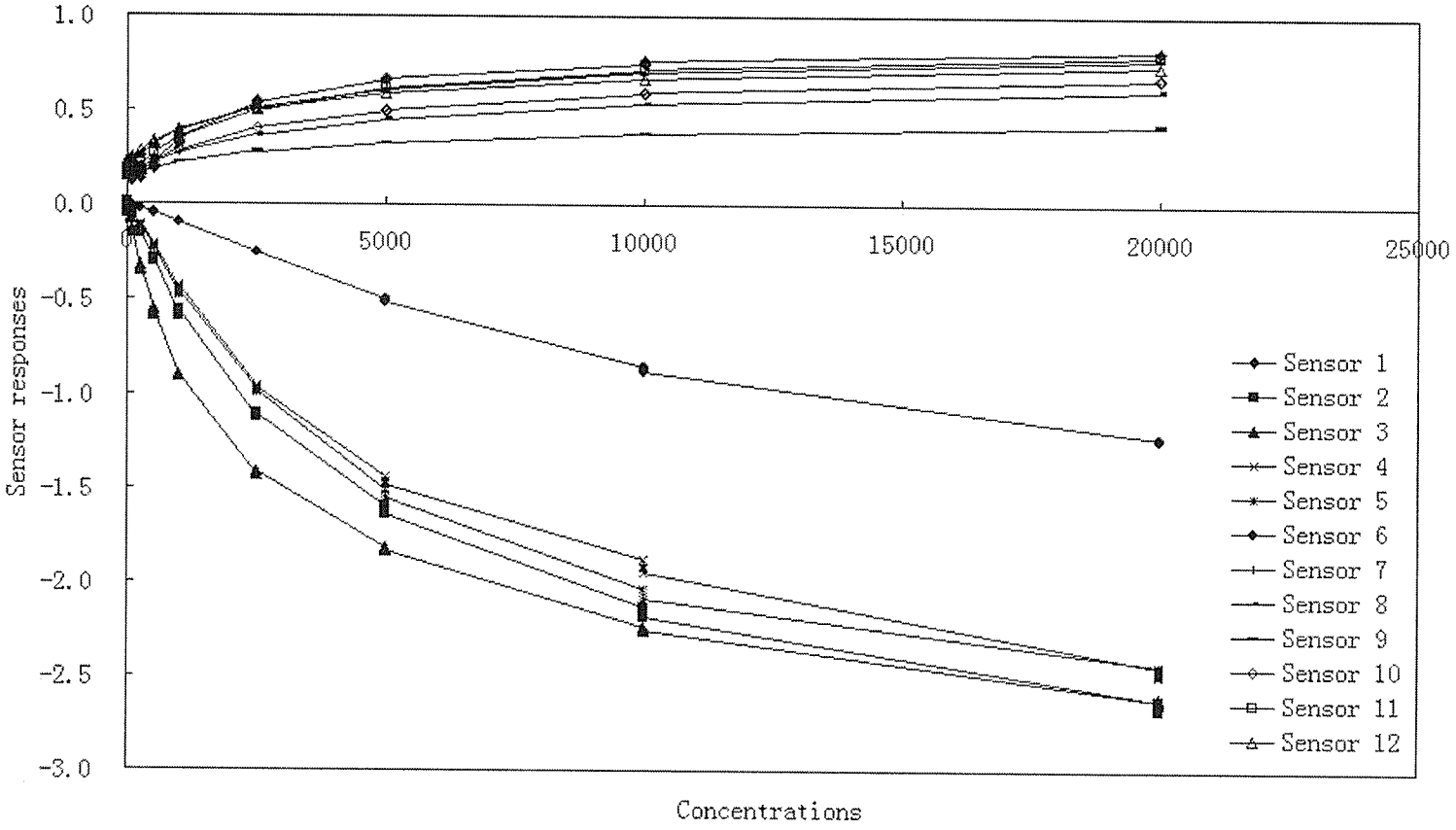

Figure 14. Sensor responses vs. concentrations for training data

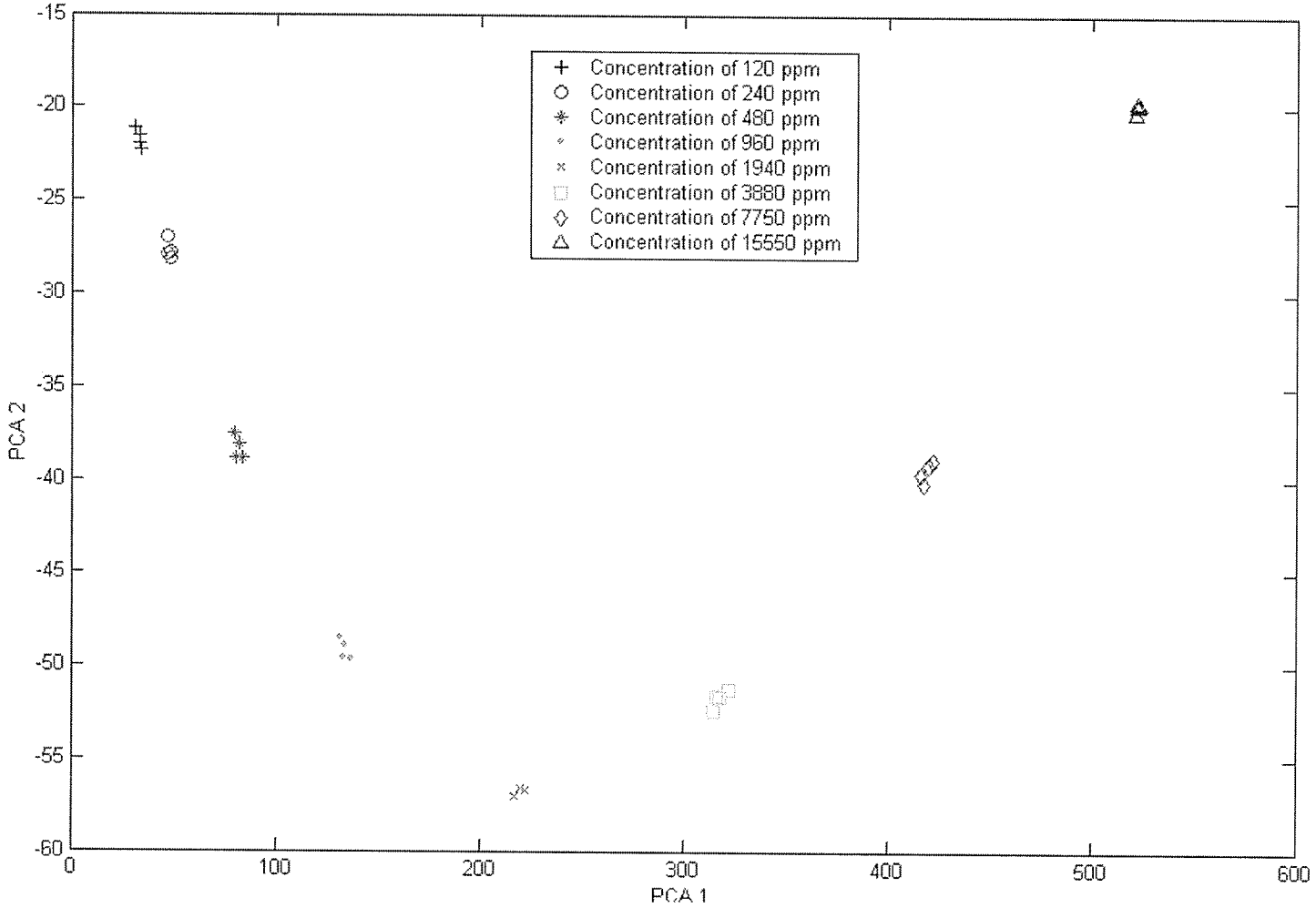

Figure 15. PCA for training data

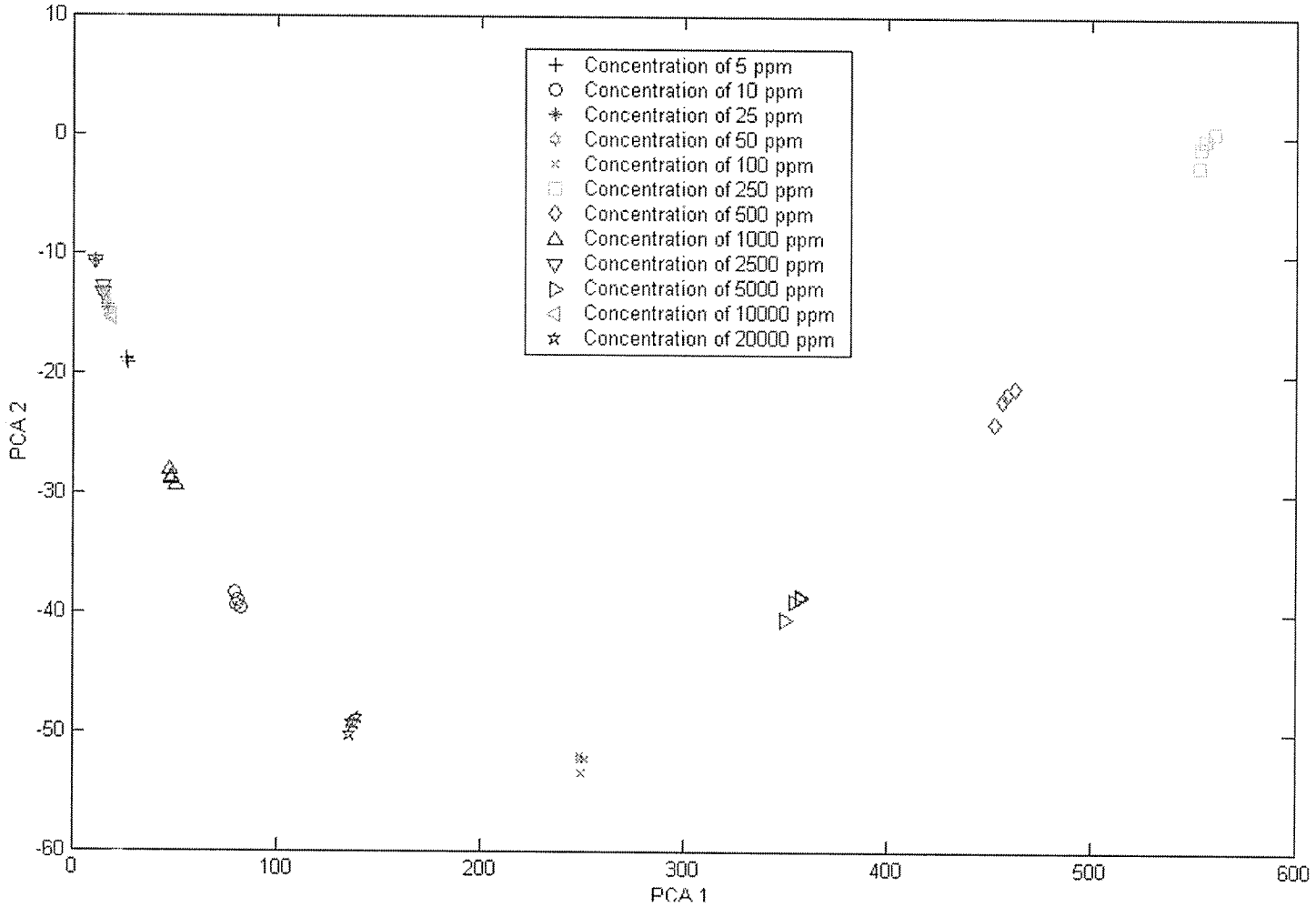

Figure 16. PCA for testing data

Each data point on the PCA graph represents a replicate of sensor response, where the 12-dimensional data has been reduced to two dimensions. The PCA plot for training data shows that eight data clusters are visible, and each cluster represents one n-butanol concentration with 4 replications. On the PCA graph for testing data, 9 clusters are visible, which corresponds 12 n-butanol concentrations. The groups established by PCA are consistent with the eight different n-butanol concentrations in training set. However only eight clusters appear to be evident in testing set which contained data for 12 n-butanol concentrations. Four clusters, i.e., n-butanol samples C, I, D, K, are very near and non-linear separable from each other. According to the theory of principal component analysis (Bishop 1995; Duda 2000), the categories that are non-linear separable in the principal component space are not certainly so in the input space. Conversely, if two categories are linear separable in the principal component space, they are also linear separable in the input space (Gao 2003). This implies that a linear classification technique can be used for training data, but a non-linear classification method is more suitable for testing data in order to obtain good performance in terms of pattern recognition.

5.2 Performance of networks on training data

5.2.1 Method of network performance evaluation

During training, the program calculates the sum of square error (SSE) and mean of square error (MSE) and displays them after every 10 epochs. The SSE is defined as the square of difference between target output and network output for a whole epoch, i.e.,

$$SSE = \sum_{i=1}^{N}\sum_{j=1}^{K}(t_{ij} - o_{ij})^2 \qquad (15)$$

where

N= the number of patterns in the data set

K= the number of output units

t_{ij} = the target output of j'th output unit for k'th pattern

o_{ij} = the network output of j'th output unit for k'th pattern.

The MSE is defined as the SSE divided by the number of patterns in the data set:

$$MSE = SSE / N \tag{16}$$

If a zero SSE is achieved in training a network, the network perfectly fits to the training set. However, it is very unusual and unnecessary for a network to achieve zero SSE because we often search for a "good enough" solution, instead of a perfect solution. Moreover, for a network to achieve zero error it may lead to overfitting problem (i.e., poor generalization ability). Another performance measure is the accuracy, defined as follows:

$$\text{Accuracy} = C / N \tag{17}$$

Where

C= the number of patterns correctly classified

N= the number of total patterns.

A network may achieve 100% accuracy on the data set even if it has a high SSE. In other words, the network is capable of discriminating patterns with acceptable errors. Our goal is to recognize or discriminate patterns, instead of approximating a function between dependent variables and independent variables as in statistical data analysis approach.

Therefore, a network selects the most activated output unit (i.e., the unit that outputs the largest value) as the network's prediction.

ENBrain calculates the SSE, MSE, and accuracy for every 10 epochs. This information is obtained by forwarding all training patterns to the network once and calculating SSE, MSE, and accuracy after every 10 epochs. After training, the trained network was tested over another data set. The testing was done in two different ways. One method was validating in the program, in which the network was given the n-butanol's concentration or intensity, and it read sensor responses to calculate SSE, MSE and accuracy. This procedure was (correct tense) as the same as what the program does during training process. The other method is called prediction, in which the network is given a data set consisting of sensor responses of n-butanol samples without being used in training networks. The network forwards input patterns and carries out predictions of concentration or intensity for these unknown n-butanol samples. The predictions were compared with actual odour concentrations or human assessed intensities.

5.2.2 Discussion of training results

Ninety networks, which were broken into 30 groups of networks from 1 hidden unit to 30 hidden units, were built and trained on the same training set for odour concentration prediction. The results of training for these networks are shown in table 5. The relationship between the number of hidden units and network performance was shown in figure 17. The best network, which is defined in this study as the network achieves higher accuracy and lower SSE, was selected within each group for making the graph. Specifically, the best network was selected in the following 3 steps:

1. Accuracies were compared and the higher accuracy was selected.

2. If accuracies are same, the lower SSE was selected.

3. If both accuracy and SSE are same, a network was selected randomly.

Table 5. Training results of 90 networks for odour concentration prediction (* indicates the network with the smallest SSE in the group)

Number of hidden units	Replication	SSE	accuracy
1	1*	18.13	12%
	2	18.13	12%
	3	18.13	12%
2	1	14.25	34%
	2*	14.25	34%
	3	14.25	34%
3	1*	11.36	71%
	2	11.39	71%
	3	11.40	65%
4	1	11.41	78%
	2	11.40	78%
	3*	11.32	78%
5	1*	6.80	100%
	2	7.58	100%
	3	7.52	100%
6	1	4.99	100%
	2*	3.72	100%
	3	4.53	100%
7	1	5.04	100%
	2*	3.98	100%
	3	4.89	100%
8	1	4.79	100%
	2	3.57	100%
	3*	3.53	100%
9	1*	4.40	100%
	2	5.06	100%
	3	5.58	100%
10	1*	2.40	100%
	2	5.21	100%
	3	3.71	100%
11	1	3.63	100%
	2*	2.99	100%
	3	3.66	100%
12	1	3.36	100%
	2	3.71	100%
	3*	3.07	100%
13	1	3.06	100%
	2*	1.71	100%
	3	2.97	100%
14	1	3.09	100%
	2*	1.43	100%
	3	2.99	100%
15	1	1.62	100%

	2	2.14	100%
	3*	1.46	100%
16	1	2.16	100%
	2	2.59	100%
	3*	1.56	100%
17	1	3.09	100%
	2*	2.11	100%
	3	3.20	100%
18	1	1.67	100%
	2	1.75	100%
	3*	1.57	100%
19	1	2.16	100%
	2	2.97	100%
	3*	1.52	100%
20	1	3.29	100%
	2*	2.03	100%
	3	2.30	100%
21	1*	1.68	100%
	2	1.97	100%
	3	2.70	100%
22	1	2.03	100%
	2	1.89	100%
	3*	1.79	100%
23	1	2.33	100%
	2	1.93	100%
	3*	1.91	100%
24	1	1.81	100%
	2*	1.65	100%
	3	1.87	100%
25	1*	1.87	100%
	2	2.88	100%
	3	3.26	100%
26	1	2.36	100%
	2	2.89	100%
	3*	1.90	100%
27	1	2.78	100%
	2	1.92	100%
	3*	1.69	100%
28	1	1.95	100%
	2*	1.65	100%
	3	2.16	100%
29	1	2.94	100%
	2	3.04	100%
	3*	2.05	100%
30	1*	2.02	100%
	2	2.22	100%
	3	2.92	100%

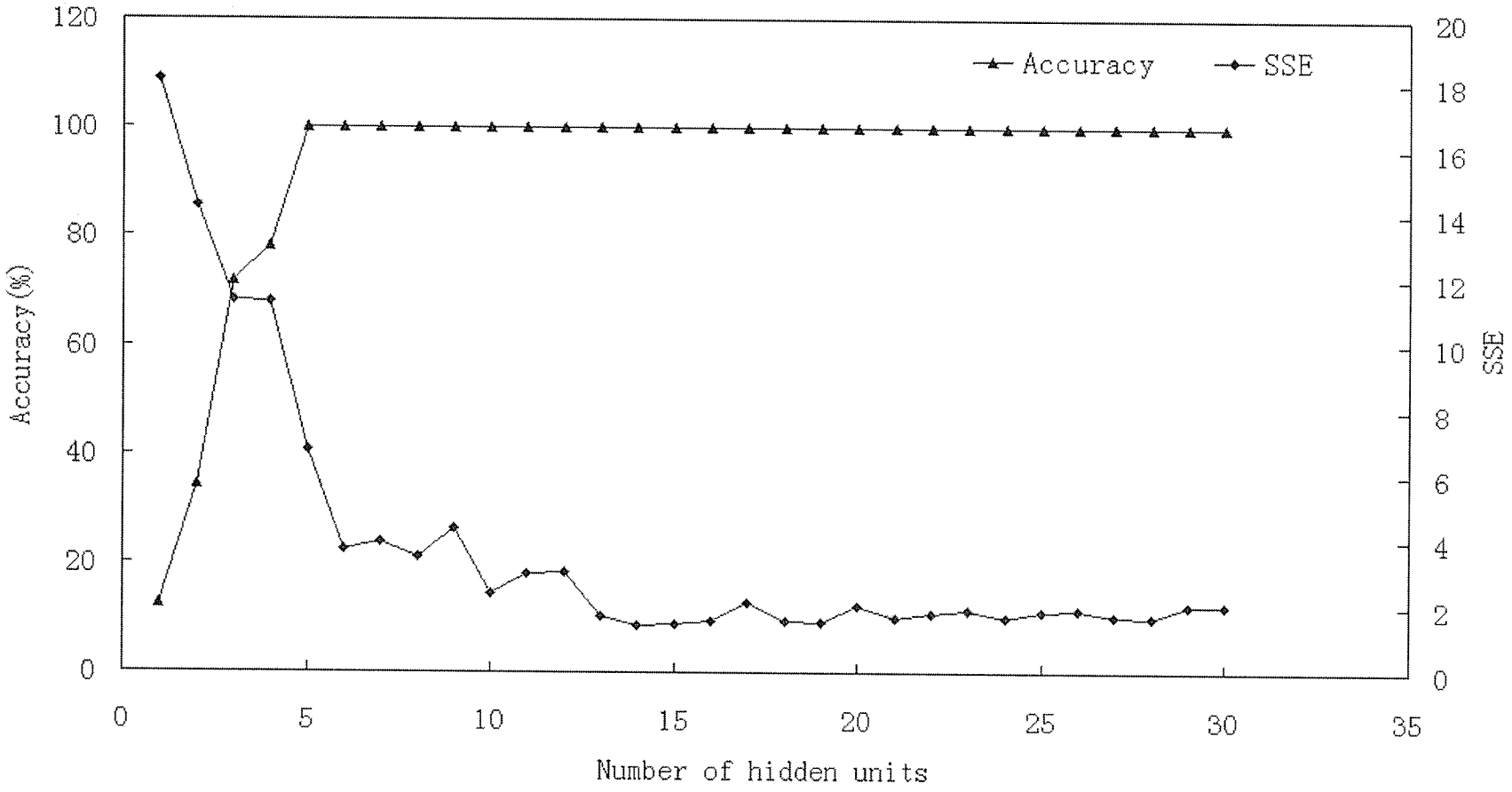

Figure 17. Performance of networks with different number of hidden units

With increase in the number of hidden units, a decrease in the SSE and an increase in accuracy were observed. For accuracy, a sharp increase was observed from 1 to 5 hidden units. An accuracy of 100% was achieved with 5 or more hidden units. A continuous decrease was observed in SSE from 1 to 14 hidden units, and no significant change was observed thereafter.

Among these 90 networks, a network with 14 hidden units had the lowest SSE at 1.43 and the highest accuracy 100% after trained 1000 epochs. During training, the network was tested on the test data set for every 10 epochs. Figure 18 illustrates the changes in SSE and accuracy during training, as well as changes in accuracy on test data. An accuracy of 100% was achieved at around the 400th epoch. There was a drastic drop in SSE for the first few epochs, and SSE continued to decrease until about 600 epochs and decreased slowly thereafter. There was trend of increase in accuracy for the testing until 750 epochs, reaching the highest value of 50%. No change was observed after the 750 epochs. In other words, the network did not overfit the training data when being trained for more than 750 epochs. On the other hand, there is no need to increase the training epoch beyond 750.

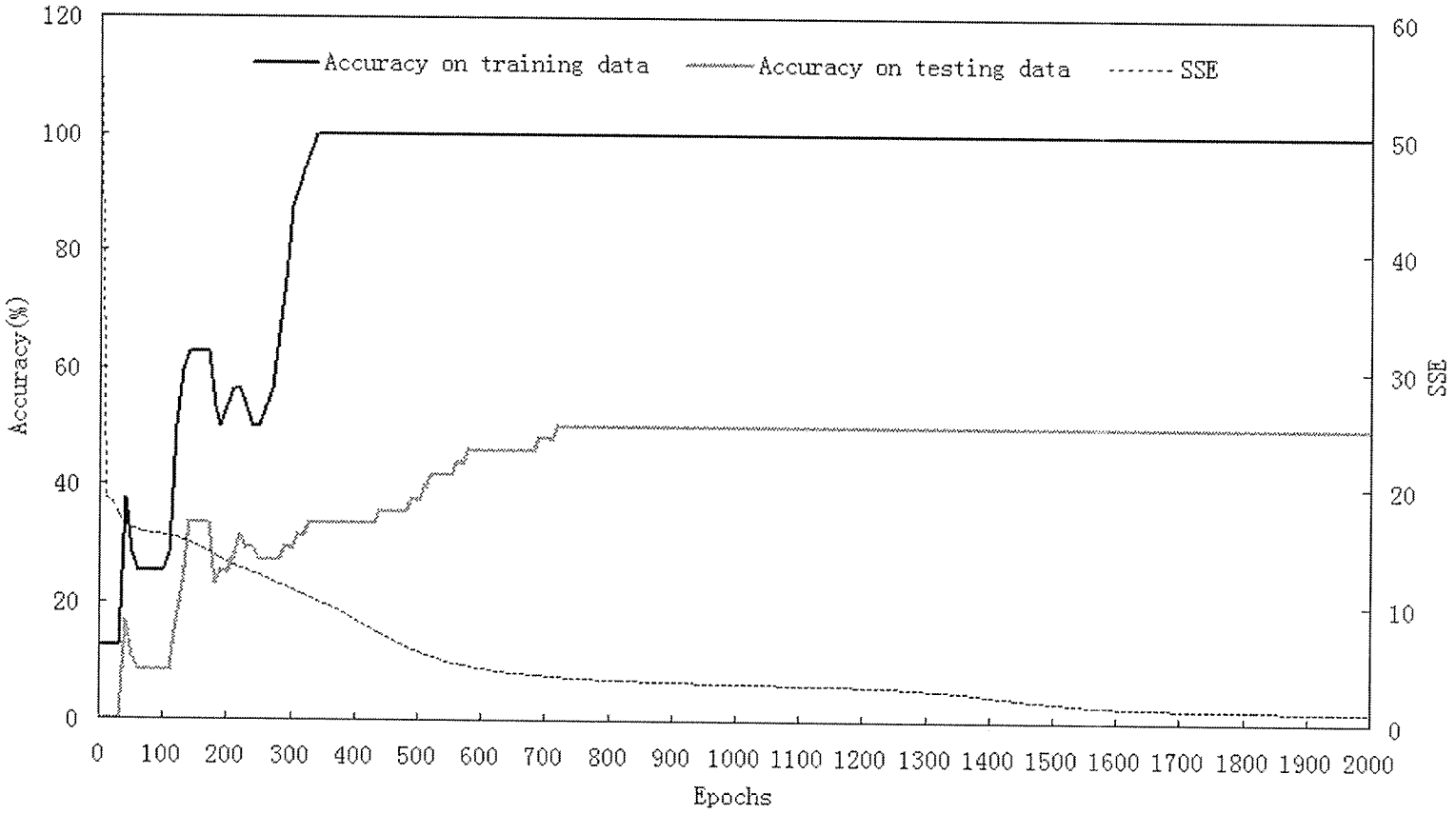

Figure 18. Variation of accuracy and SSE during training

Similarly, for intensity prediction, 3 groups of networks with 10, 20 and 30 hidden units respectively, were experimented for both 8 point and 0-200 intensity scales, and the results are shown in tables 6 and 7. All of these networks achieved 100% accuracy and low SSE (<7.22). The performance of networks on training data is consistent with the PCA, which shows the clusters of training data are evident, and thus the networks can achieve 100% accuracy.

Table 6. Training results of 9 networks using the 8 point odour intensity scale (* the network with the smallest SSE in the group of networks)

Num. of hidden units	Replication	SSE	Accuracy
10	1	2.21	100%
	2	1.91	100%
	3*	1.76	100%
20	1	3.28	100%
	2*	2.01	100%
	3	2.39	100%
30	1	2.34	100%
	2*	2.02	100%
	3	2.03	100%

Table 7. Training results of 9 networks using 0-200 odour intensity scale (* the network with the smallest SSE in the group of networks)

Num. of hidden units	Replication	SSE	Accuracy
10	1	5.91	100%
	2*	4.06	100%
	3	7.22	100%
20	1	3.24	100%
	2	3.23	100%
	3*	1.56	100%
30	1	2.44	100%
	2	1.97	100%
	3*	1.82	100%

5.3 Comparison between neural network prediction and human assessment

Network predictions of odour concentrations are compared with the actual concentrations of n-butanol samples. Predicted odour intensities are compared with those assessed by human panels. Coefficients of determination were calculated as an indicator of network performance.

For concentration prediction, the coefficient of determination (R^2) between predicted and actual n-butanol concentrations was plotted as a function of number of hidden units in figure 19, in which the best network was selected in each network group. A sharp increase in R^2 was observed with the increase in the number of hidden units for less than five hidden units, and little change after 14 hidden units. The highest R2- value achieved was 0.89. Figure 20 shows the comparison between predicted and actual n-butanol concentrations for a 13-14-50 network, which had the lowest SSE and highest R2- value. Points representing samples C, I, D, and K deviate from the best linear fit line (slope = 1) much more than other points. This is consistent with the result of PCA for testing data, in which the clusters of samples C, I, D, and K are overlapped, and hence are difficult to discriminate. This might be attributed to the extremely low concentrations represented by samples C, I, D, and K. The range of concentrations of these four samples (5 to 50 ppm) constituted an insignificant portion of the total range of concentrations tested (5 to 20000 ppm). Furthermore, this should not be a concern practically because the extremely low concentrations of n-butanol represented by samples C, I, D, and K are seldomly encountered in practice.

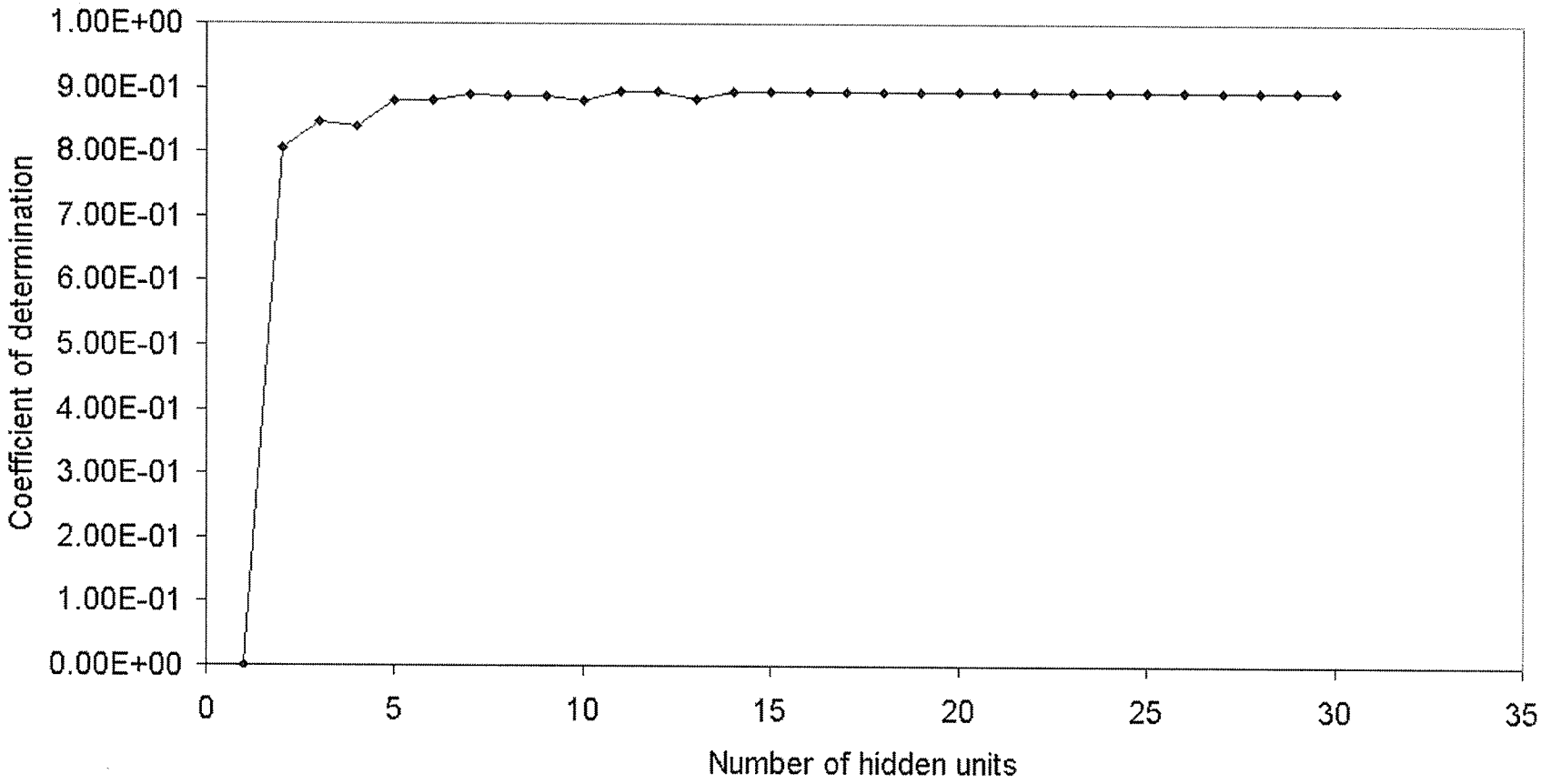

Figure 19. Odour concentration predictions with different number of hidden units

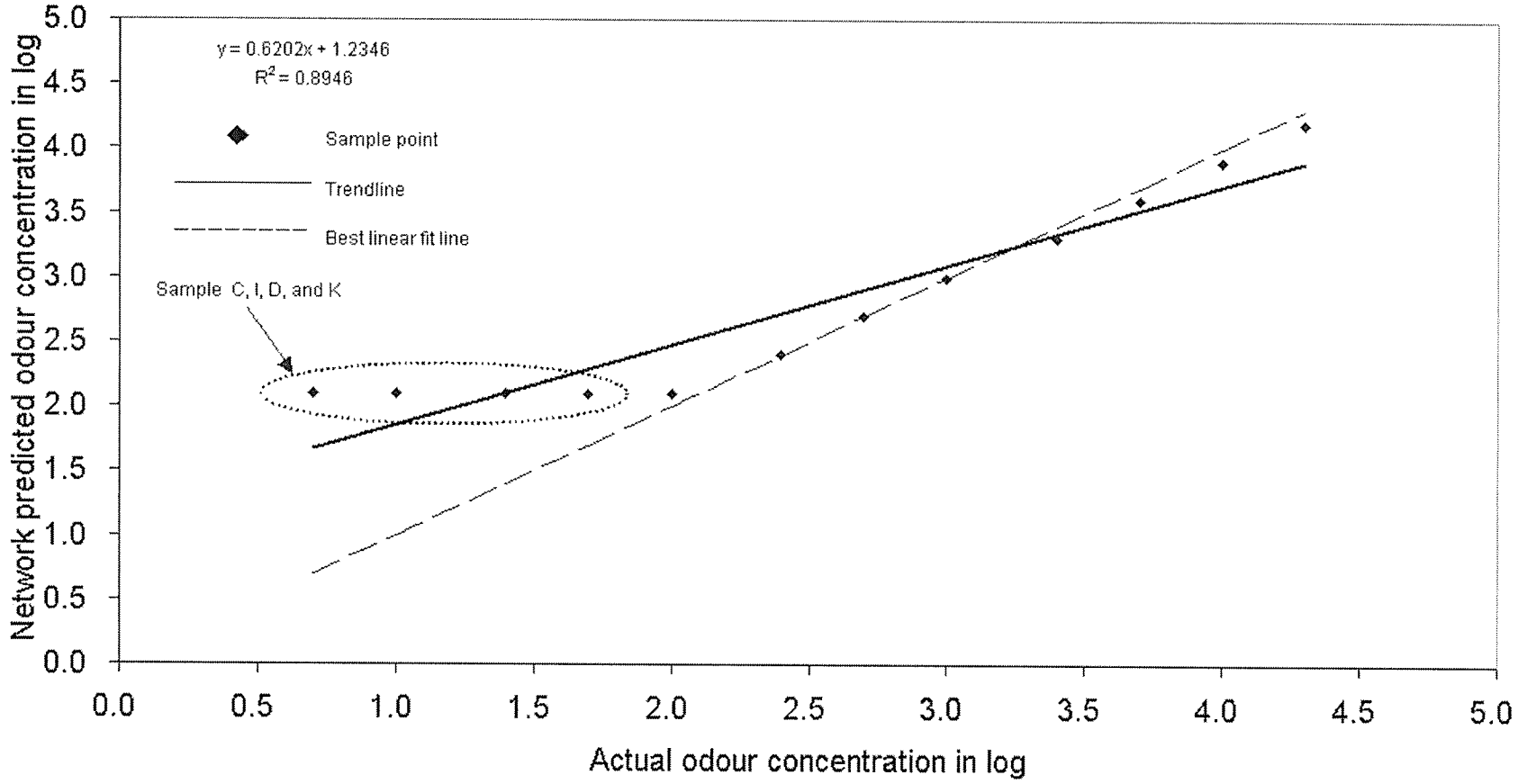

Figure 20. Comparison of network predicted and actual odour concentrations

Figure 21 shows the relationship between network performance and number of hidden units for odour intensity prediction. For the eight-point intensity scale, the three networks had the same R^2 at 0.97. For the 0-200 scale, an increase in R^2 was observed between 10 and 20 hidden units, and no change was observed between 20 and 30 hidden units. The highest R^2 value for 0-200 scale is 0.95. The relationship between the network prediction of odour intensity and human assessment is shown in figures 22 (eight point intensity scale) and 23 (0-200 intensity scale). A network with 10 hidden units was selected to plot figure 22, since the network had the lowest SSE and highest correlation with R^2=0.97. Another network with 20 hidden units, which had the lowest SSE and highest correlation with R^2=0.9461, was selected to plot figure 23. Linear regression equations were presented in figure 22 and 23. Slopes obtained in the regression equations, 0.96 (fig. 22) and 1.17 (fig. 23), were close to 1, which indicates the predicted intensities were very close to those assessed by human panels.

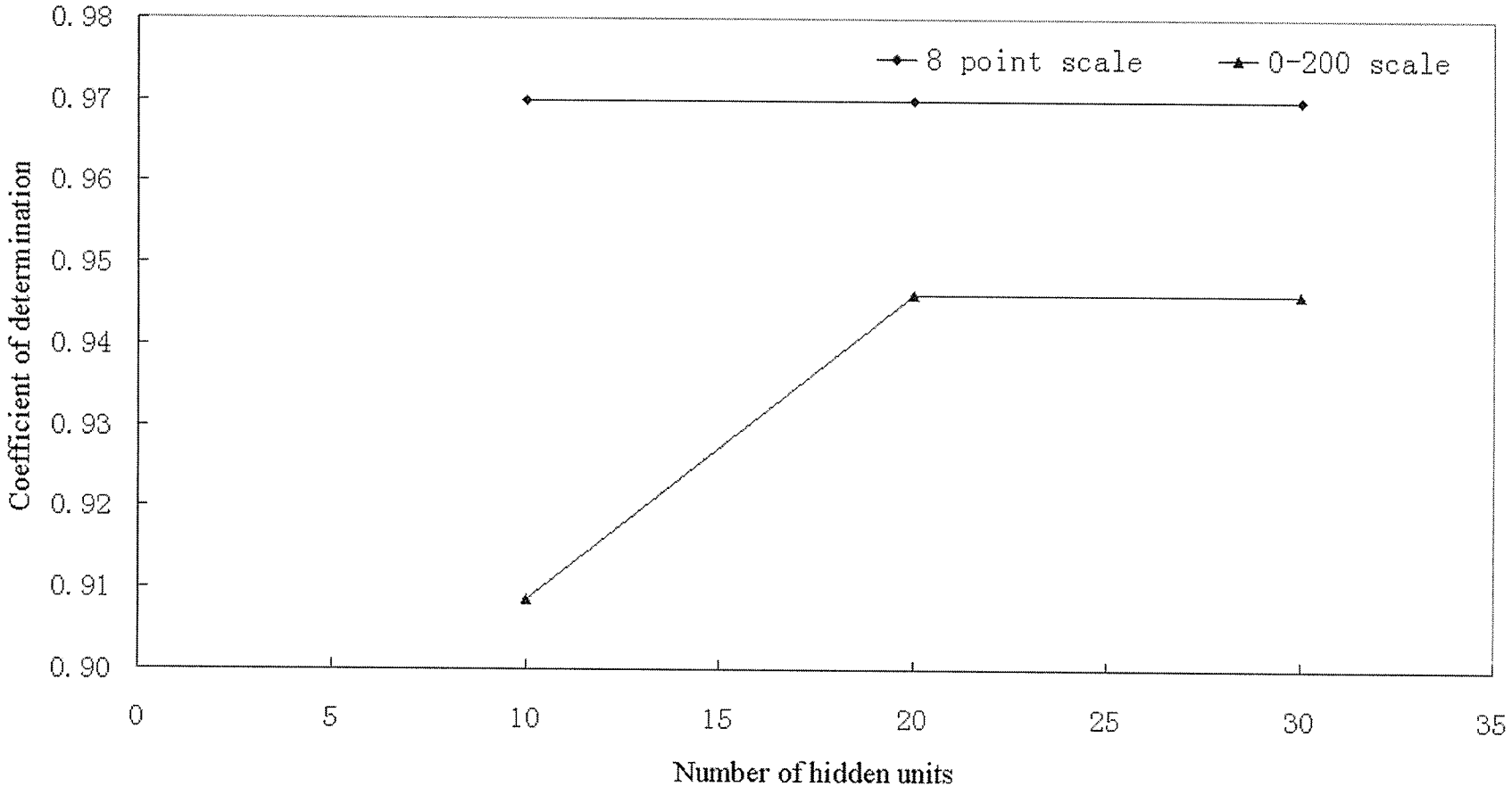

Figure 21. Odour intensity predictions with different number of hidden units

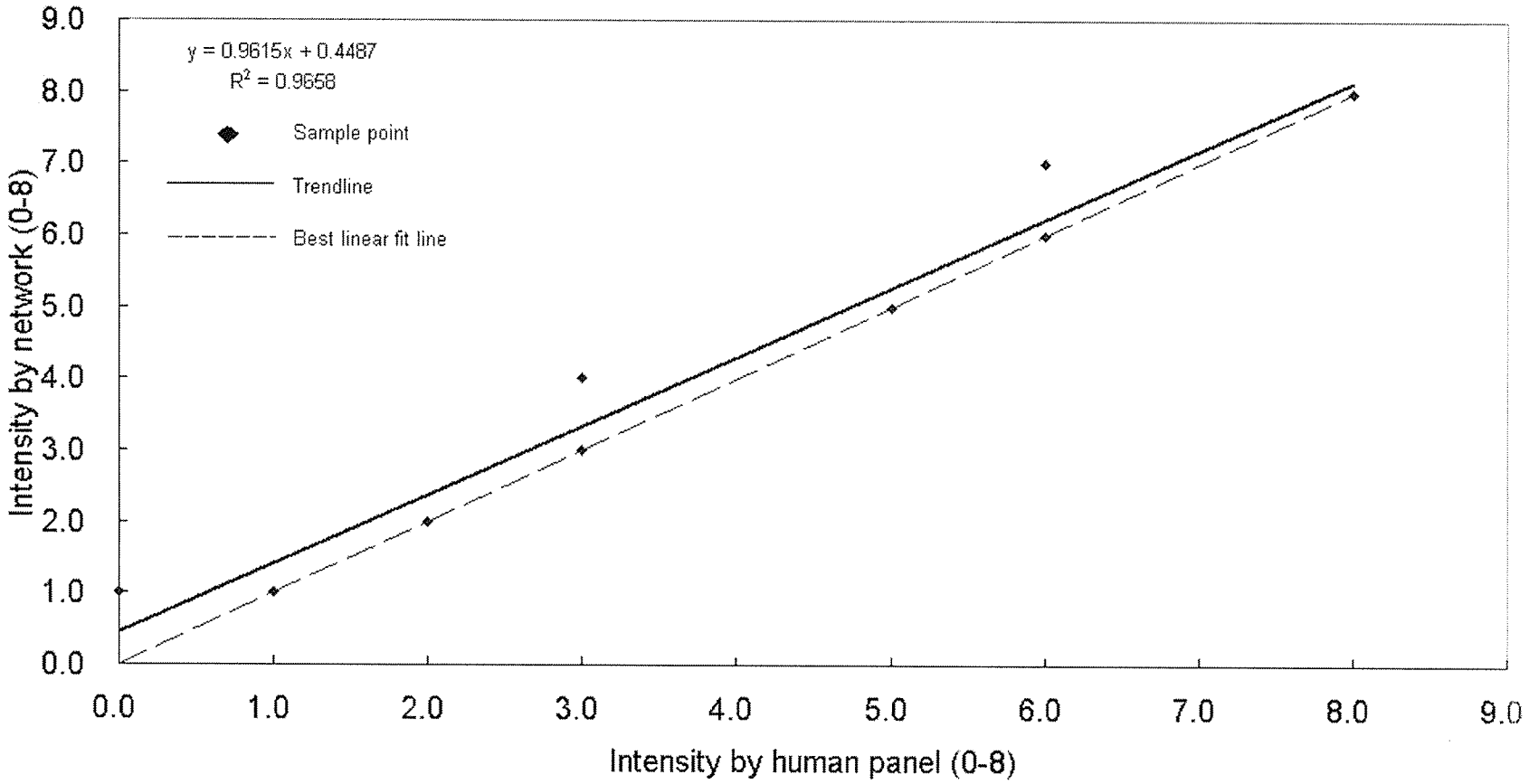

Figure 22. Comparison of network predicted and human assessed odour intensities in the 8-point scale

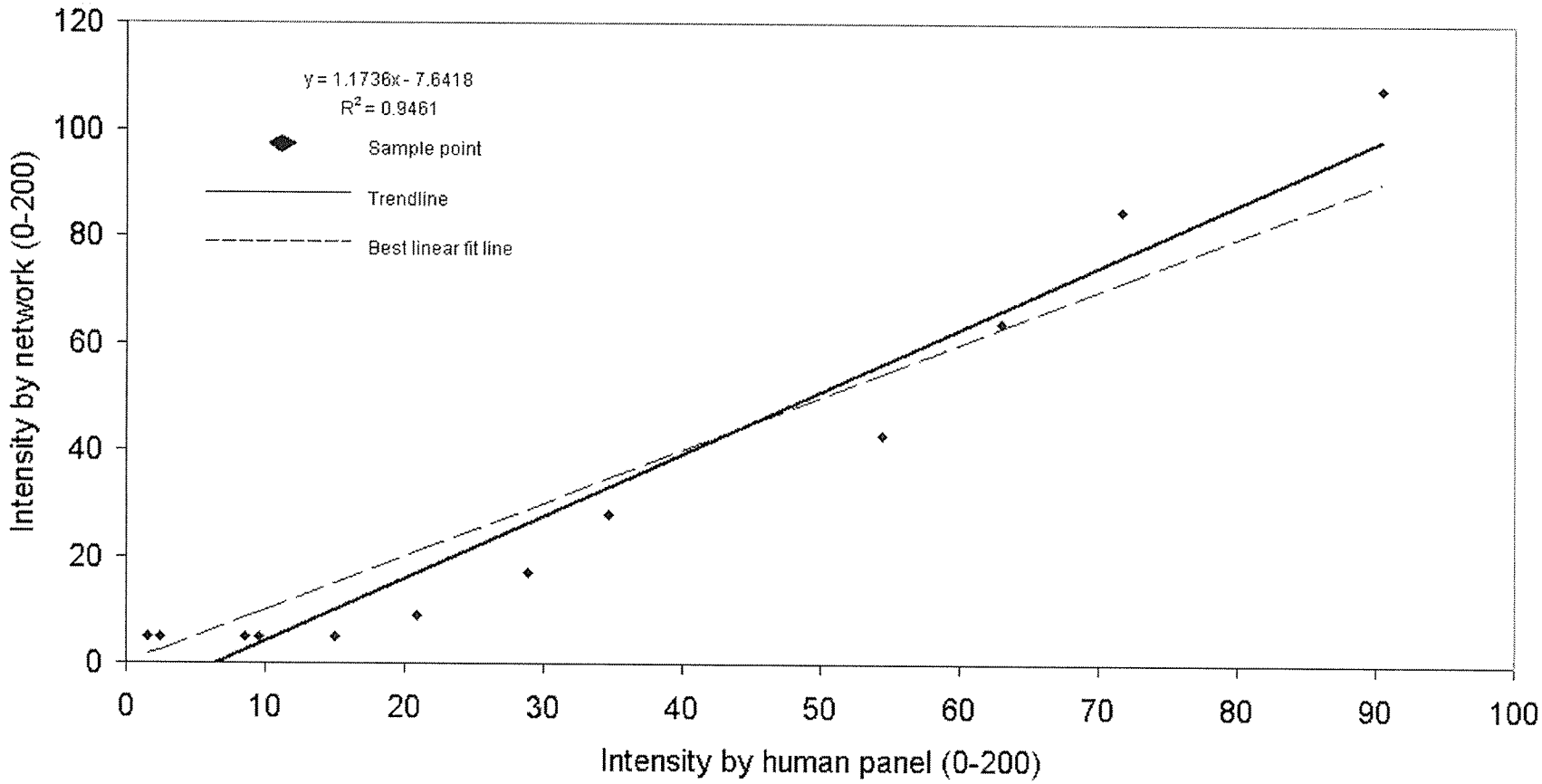

Figure 23. Comparison of network predicted and human assessed odour intensities in the 0-200 scale

6. CONCLUSIONS

1. It is adequate to use artificial neural networks in electronic noses to predict odour intensities and concentrations of n-butanol. High accuracy (100%) was achieved for predicting odour concentrations and intensities during training neural networks. High coefficient of determination ($R^2 = 0.89$) was obtained between network predicted and actual odour concentrations. A high correlation ($R^2 > 0.95$) was obtained between network predicted and human panel assessed odour intensities in both the eight-point and 0-200 intensity scales. These two odour intensity scales have been used as the odour referencing scale for complex odours. Therefore, the results from this study indicated that artificial neural networks have potentials to predict odour intensities and concentrations for complex odours.
2. The number of hidden units affects network performance. With an increase in the number of hidden units, the network performance was improved. However, no change was observed in network performance after a certain number of hidden units.

7. REFERENCES

Ali, Z., James, D., O'Hare, W.T., Rowell, F.J., Scott, S.M. 2003. Radial basis neural network for the classification of fresh edible oils using an electronic nose, Journal of Thermal Analysis and Calorimetry 71:147-154.

Anonymous. 1998. Control of odour emissions from animal operations, report of odour control task force, North Carolina Agricultural Research Service, College of Agriculture and Life Sciences, North Carolina State University, Raleigh, NC.

ASTM E544 1975 (re-approved 1997). Standard practice for referencing suprathreshold odour intensity, Philadelphia, PA.

ASTM 1991. Standard terminology relating to sensory evaluation of materials and products, Annu Book ASTM Stand 1991 15.07:E253-91a

Balakrishnan K, and Honavar V. 1995. Evolutionary design of neural architectures – a preliminary taxonomy and guide to literature, Technical report, AI Research Group, January 1995, CS-TR 95-01.

Baltes, J. 2003. Dr. Jacky Baltes's lecture notes on Machine Learnin, Department of Computer Science, University of Manitoba, Winnipeg, MB, Canada.

Bartlett, P.N., Elliot, J.M., and Gardner, J.W. 1997. Applications of, and developments in, machine olfaction, Ann. Chim. 87:33-44.

Barth, C.L. 1973. Odour sensation theory and phenomena and their effect on olfactory measurements. Transactions of the ASAS 16(2):340-347.

Beets, M.G.J. 1957. Structure and odour, Molecular structure and organoleptic quality, S.C.I. Monog. 1:54-90.

Beets, M.G.J. 1964. A molecular approach to olfaction, In: Ariens, E.J. (Editor) Molecular Pharmacology, Academic Press, New York.

Berna, A.Z., Lammertyn, J., Saevels, S., Natale, C.D., and Nicolai, B.M. 2004. Electronic nose systems to study shelf life and cultivar effect on tomato aroma profile, Sensors and Actuators B 97:324-333.

Berry, M.J.A. and Linoff, G. 1997. Data Mining Techniques, John Wiley & Sons, New York, USA.

Bishop, C.M. 1995. Neural Networks for Pattern Recognition, Oxford University Press, New York.

Bliss, P.J., Schulz, T.J., Senger, T., and Kaye, R.B. 1996. Odour measurement-factors affecting olfactometry panel performance, Wat. Sci. Tech. 34(3-4):549-556.

Blum, A. 1992. Neural Networks in C++, John Wiley & Sons, New York, USA.

Boger, Z. and Guterman, H. 1997. Knowledge extraction from artificial neural network Models, IEEE Systems, Man and Cybernetics Conference, Orlando, FL, USA.

Boilot, P., Hines, E.L., Gongora, M.A., and Folland, R.S. 2002. Electronic noses inter-comparison, data fusion and sensor selection in discrimination of standard fruit solutions, Sensors and Actuators B 88:80-88.

Brennan, B. 1993. Odour nuisance, Water and Waste Treatment 36:30-33.

Broomhead, D.S., and Lowe, D. 1988. Multivariable functional interpolation and adaptive networks, Complex Systems 2:321-355.

Byfield, M.P. and May, I.P. 1996. Olfactory sensor array systems: the electronic nose, GEC J. Res. 13:17-27.

Byun, H.G., Persaud, K.C., Khaffaf, S.M., Hobbs, P.J., and Misselbrook, T.H. 1997. Application of unsupervised clustering methods to the assessment of malodour in agriculture using an array of conducting polymer odour sensors, Computers and Electronics in Agriculture 17:233-247.

Cain W.S. And Moskowitz H.R. 1974. Psychophysical scaling of odours. In: Turk A, Johnston J.W., editors, Human response to environmental odours, Academic Press, New York.

Carey, W.P., Bebe, K.R., and Kowalski, B.R. 1987. Multicomponent analysis using an array of piezoelectric crystal sensors, Anal. Chem. 59:29-34.

Chelani, A.B., Chalapati Rao, C.V., Phadke, K.M., and Hasan, M.Z. 2002. Prediction of sulphur dioxide concentration using artificial neural networks, Environmental Modelling & Software 17:161-168.

Clanton, C.J., Schmidt, D.R., Nicolai, R.E., Goodrich, P.R., Jacobson, L.D., Janni, K.A., Weisberg S. and Buckel, J.A. 1999. Dynamic olfactometer variability in determining odour dilution-to-threshold, Transactions of the ASAE 42(4):1103-1112.

Demuth, H. and Beale, M. 1994. Neural Network Toolbox User's Guide, The MathWorks Inc, USA.

Dickinson, T.A., White, J., Kauer, J.S. and Walt, D.R. 1998. Current trends in "artificial nose" technology, Trends Biotechnol 16:250-258.

Dowdeswell, R.M. and Payne, P.A. 1999. Odour measurement using conducting polymer gas sensors and an artificial neural network decision system, Engineering science and education journal, June.

Duda, R.O., Hart, P.E., and Stork, D.G. 2000. Pattern Classification, second ed., Wiley, New York.

Dutta, R., Kashwan, K.R., Bhuyan, M., Hines, E.L., and Gardner, J.W. 2003a. Electronic nose based tea quality standardization, Neural Networks 16:847-853.

Dutta, R., Hines, E.L., Gardner, J.W., Udrea, D.D., and Boilot, P. 2003b. Non-destructive egg freshness determination: an electronic nose based approach, Measurement Science and Technology (IoP) 14:190-198.

Dutta, R., Hines, E.L., Gardner, J.W., Kashwan, K.R., and Bhuyan, M. 2003c. Tea quality prediction using a tin oxide-based electronic nose: an artificial intelligence approach, Sensors and Actuators B 94:228-237.

Evans, P., Persaud, K.C., McNeish, A.S., Sneath, R.W., Hobson, N., and Magan, N. 2000. Evaluation of a radial basis function neural network for the determination of wheat quality from electronic nose data, Sensors and Actuators B 69:348-358.

Falco, I.D., Della, C.A., Iazzetta, A., Natale, P., and Tarantino E. 1998. Optimizing neural networks for time series prediction, Third World Conference on Soft Computing (WSC3).

Gao, D., Wang, S., and Ji, Y. 2004. An electronic nose and modular radial basis function network classifiers for recognizing multiple fragrant materials, Sensors and Actuators B 97:391-401.

Gardner, J.W. and Bartlett, P.N. 1994. A brief history of electronic noses, Sensors Actuators B 18:211-220.

Gardner, J.W. and Bartlett, P.N. 1999. Electronic Noses – Principles and Applications, Oxford University Press.

Gardner, J.W. 1991. Detection of vapours and odours from a multisensor array using pattern recognition Part 1. Principal component and cluster analysis, Sensors and Actuators B 4:109-115.

Gardner, J.W., Hines, E.L., and Tang, H.C. 1992. Detection of vapors and odors from a multisensor array using pattern recognition techniques, Part 2, Artificial neural networks, Sensors and Actuators B 9:9-15.

Gilbert, A.N. and Wysocki, C.J. 1987. The smell survey, Natl Geogr. 19:514-525.

Goldberg, D.E. 1989. Genetic Algorithms in Search, Optimization and Machine Learning, Reading, MA: Addison-Wesley.

Gostelow, P., Parsons, S.A., and Stuetz, R.M. 2001. Odour measurements for sewage treatment works, Wat. Res. 35(3):579-597.

Harper, R., Bate Smith E.C. and Land D.G. 1968. Odour description and odour classification, J. & A. Churchill Ltd., London.

Hierlemann, A., Weimar, U., Kraus, G., Schweizer-Berberich, M., and Gopel, W. 1995. Polymer-based sensor arrays and multicomponent analysis for the detection of hazardous organic vapours in the environment, Sensor Actuators B:26-27, 126-134.

Ho, C.T., Hwang, H.I., Yu, T.H., and Zhang, J. 1993. An overview of the Maillard reactions related to aroma generation in coffee, Proceedings of 15th International Scientific Colloquium on Coffee II: 519-527.

Hobbs, P.J., Misselbrook, T.M., and Pain, B.F. 1995. Assessment of odours from livestock wastes by a photoionization detector, an electronic nose, olfactometry and gas chromatography-mass spectrometry, J. Agric. Eng. Res. 60:137-144.

Holland, J.H. 1975. Adaptation in Natural and Artificial Systems, Ann Arbor: University of Michigan Press.

Holmberg, M., Davide, F.A.M., Natale, C.D., D'Amico, A., Winquist, F., and Lundstrom, I. 1997. Drift counteraction in odour recognition applications: lifelong calibration method, Sensor and Actuator B:185-194.

Hudon, G. and Guy C. 2000. Measurement of odor intensity by an electronic nose, Journal of the Air & Waste Management Association 50:1750-1758.

Kermani, B.G., Schiffman, S.S., and Nagle, H.T. 1999. Using neural networks and genetic algorithms to enhance performance in an electronic nose, IEEE Transactions on Biomedical Engineering, 36, 4, April.

Kermit, M. and Tomic, O. 2001. Increased discrimination between odor signals by independent component analysis, Proceeding of Artificial Intelligence and Soft Computing (ASC 2001), Cancun, Mexico.

Korenman, Ya. I., and Kalach, A.V. 2003. Application of multi-sensor system for determination of nitroethane in the air, Sensors and Actuators B 88:334-336.

Kress-Rogers, K. (Ed.) 1996. Handbook of Biosensors and Electronic Noses, Medicine, Food and Environment, CRC Press, Boca Raton.

Lavine, B.K. 2000. Encyclopaedia of Analytical Chemistry, John Wiley & Sons, Chichester, USA.

Lucasius, C.B., and Kateman, G. 1993. Understanding and using genetic algorithms, Part 1: Concepts, properties and context, Chemometrics and Intelligent Laboratory Systems 19:1-33.

Mccoy, S.A., Martin, T.P. and Baldwin, J.F. 2003. Learning rules for odour recognition in an electronic nose, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 11(5):517-543.

Misselbrook, T.M., Hobbs, P.J., and Persaud, K.C. 1997. Use of an electronic nose to measure odour concentration following application of cattle slurry to grassland, J. Agric. Eng. Res. 66:213-220.

Mitchell, T.M. 1997. Machine Learning, McGraw-Hill companies, Inc., New York, New York.

Moncrieff, R.W. 1951. The chemical senses, Leonard Hill Ltd., London.

Moody, J.E., and Darken, C.J. 1989. Fast learning in networks of locally-tuned processing units, Neural Computation 1(2):281-294.

Nanto, H., Tsubakino, S., Ikeda, M. and Endo, F. 1995. Identification of aromas from wine using quartz-resonator gas sensors in conjunction with neural-network analysis, Sensors and Actuators B:24-25, 794-796.

Naves, Y.R. 1957. The relation between the stereochemistry and odorous properties of organic substances, Molecular structure and organoleptic quality, 1:38-53.

Odour Control Task Force 1998. Control of odor emissions from animal operations, A report from the Board of Governors of the University of North Carolina, NC State University, USA.

Pearce, T.C., Gardner, J.W., Friel, S., Bartlett, P.N., and Blair, N. 1993. Electronic nose for monitoring the flavour of beers, Analyst 118.

Pearce, T.C. 1997. Computational parallels between the biological olfactory pathway and its analogue 'The Electronic Nose': Part I. Biological olfaction, Biosystems 41: 43-67.

Powell, M.J.D. 1987. Radial basis functions for multivariable interpolation: A review, In Mason, J., & Cox, M. (Eds), Algorithms for approximation, Pages 143-167, Oxford: Clarendon Press.

Preti, G., Gittelman, T.S., Staudte, P.B. and Luitweiler, P. 1993. Letting the nose lead the way – malodorous components in drinking water, Anal. Chem. 65: 699A-702A.

Riskowski, G.L., Chang, A.C., Steinberg, M.P. and Day, D.L. 1991. Methods for evaluating odour from swine manure, Applied engineering in agriculture 7(2):248-253.

Sarkar, Ujjaini and Hobbs, Stephen E. 2002. Odour from municipal solid waste (MSW) landfills: A study on the analysis of perception, Environment International 27:655-662.

Sarle, W.S. 2001. Neural network FAQ, part 3 of 7: generalization, online, Availabe from : ftp://ftp.sas.com/pub/neural/FAQ3.html, Usenet newsgroup comp.ai.neural-nets, Discussion list.

Schulz, T.J. and van Harreveld, A.P. 1996. International moves towards standardisation of odour measurements using olfactometry, Wat. Sci. Tech. 34(3-4):541-547.

Schaller, E., Bosset, J.O., and Escher, F. 1998. Electronic noses and their application to food, Food Science and Technology, 31(4):305-316.

Schaffer, J.D., Whitley, L.D., and Eshelman, L.J. 1992. Combinations of genetic algorithms and neural networks: a survey of the state of the art, In L.D. Whitley and J.D. Schaffer (Eds.), International Workshop on Combinations of Genetic Algorithms and Neural Networks, 1-37, Los Alamitos, CA: IEEE Computer Society Press.

Siedlecki, W., and Sklansky, J. 1988. On automatic feature selection, International Journal of Pattern Recognition and Artificial Intelligence 2(2):197-220.

Sohn, J.H., Smith, R., Yoong, E., Leis, J., and Galvin, G. 2003. Quantification of odours from piggery effluent ponds using an electronic nose and an artificial neural network, Biosystems Engineering 86(4):399-410.

Srivastava, A.K. 2003. Detection of volatile organic compounds (VOCs) using SnO2 gas-sensor array and artificial neural network, Sensors and Actuators B 96:24-37.

Srivastava, A.K. and Levy, D.C. 2002. Gas sensor monitoring of environmental air quality, ChemoSense 4 (3):1-6.

Srivastava, A.K., Shukla, K.K., and Srivastava, S.K. 1998. Exploring neuro-genetic processing of electronic nose data, Microelectronics Journal 29:921-931.

Stevens, S.S. 1957. On the psychophysical law, Psychology Review 61:153-181.

Stoll, M. 1957. Facts old and new concerning relationships between molecular structure and odour, S.C.I. Monograph, 1:1-12.

Stordeur, R.T., Stordeur, C.M., Levine, S.P. and Hoggatt, J.H. 1981. A new microprocessor-controlled dynamic olfactometer, J. Air Pollut 31:377-380.

Stuetz, R.M., Fenner, R.A, and Engin, G. 1999. Assessment of odours from sewage treatment works by an electronic nose, H2S analysis and olfactometry, Wat. Res. 33(2):453-461.

Summer, W. 1971. Odour pollution of air, Leonard Hill Books, London.

Swingler, K. 1996. Applying Neural Networks: A Practical Guide, Academic Press, London.

Yao X. 1993. A review of evolutionary artificial neural networks, International Journal of Intelligent Systems 8:539-567.

York R.K. 2005. Studies on textile stabilization of agricultural malodors for sensory and electronic nose analyses. Ph. D. thesis, University of Manitoba.

Young, P.J. 1984. Odours from effluent and waste treatment, Effluent Water Treatment J. 24:189-195.

Zaromb, S. and Stellar J.R. 1984. Theoretical basis for identification and measurement of air contaminants using an array of sensors having partially overlapping sensitivities, Sensors and Actuators 6:225-243.

Zhang, Q., Feddes, J.J.R., Edeogu, I.K., and Zhou, X.J. 2002. Correlation between odour intensity assessed by human assessors and odour concentration measured with olfactometers, Canadian Biosystems Engineering 44.

Zhou X. 2003. Odour emissions from swine operations in Manitoba, Master thesis, University of Manitoba.

Appendix A: Essential Source Codes of ENBrain

```
import java.io.*;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

public class BPNN
{
        final byte equal='=';
        final byte comma=',';
        final byte well='#';


        int innum=0;      //the number of input units
        int hidnum=0;     // the number of hidden units
        int outnum=0;     // the number of output units.

        String en_code;
        String sample_code;
        float concentration=0.0F;
        float tarinten=0.0F;

        boolean isout=false;        // if write the screen output into a file
        boolean istrain=false;      // if it is in train mode
        boolean istest=false;       // if it is in test mode
        boolean issave=false;       // if save the net into a file
        boolean isload=false;       // if load a net from a file
        boolean stoptraining=false;
        String outfile=new String();          // the name of file storing the screen output
        String loadfile=new String();         // the name of net file which is loaded
        String savefile=new String();         // the name of net file which is saved
        File[] fs=null;                       // file sequence

        int epochs=0;                         // the number of epochs
        float learningrate=0.0F;    // learnnig rate
        float momentum=0.0F;                  // momentum

        boolean concorinten;
        float increment=0.0f;

        float[] inunit;     // the input unit array
        float[] hidunit;    // the hidden unit array
        float[] outunit;    // the output unit array
        float[] tarout;     // the target output unit array
        float[][] hidw;     // the weight between input and hidden layer
        float[][] outw;     // the weight between hidden and output layer
        float[][] oldhidw; // the delta_w(n-1) for hidden_input layer
        float[][] oldoutw; // the delta-w(n-1) for output_hidden layer
        float[] errh;                 // error array for hidden units
        float[] erro;                 // error array for output units


        NetDisplayFrame ndf;
```

```
public BPNN(NetDisplayFrame frame, int iu, int hu, int ou, int ep, float lr, float mt, boolean ci)
{
        ndf=frame;
        innum=iu;
        hidnum=hu;
        outnum=ou;
        epochs=ep;
        learningrate=lr;
        momentum=mt;
        concorinten=ci;
        increment=5.0f/ (float)(outnum);

        inunit=new float[innum];
        hidunit=new float[hidnum];
        outunit=new float[outnum];
        tarout=new float[outnum];
        hidw=new float[hidnum][innum];
        outw=new float[outnum][hidnum];
        oldhidw=new float[hidnum][innum];
        oldoutw=new float[outnum][hidnum];
        errh=new float[hidnum];
        erro=new float[outnum];
}

public BPNN(NetDisplayFrame frame)
{
        ndf=frame;
}
public void SetEpochs(int ep)
{
        epochs=ep;
}
public void SetLearningrate(float lr)
{
        learningrate=lr;
}
public void SetMomentum(float mom)
{
        momentum=mom;
}
public float GetMomentum()
{
        return momentum;
}
public float GetLearningrate()
{
        return learningrate;
}
public int GetEpochs()
{
        return epochs;
}

public File[] GetFiles()
{
```

```
        return fs;
}
public int GetOutputNum()
{
        return outnum;
}
public int GetHiddenNum()
{
        return hidnum;
}
public int GetInputNum()
{
        return innum;
}
public void SetFiles(File[] f)
{
        fs=f;
}

public boolean IsFileSet()
{
        if (fs==null)
                return false;
        return true;
}

public boolean IsConcentration()
{
        return concorinten;
}

public void ReadIns(File fp) throws IOException      // read a training instance from a file
{
        try
        {

                RandomAccessFile in=new RandomAccessFile(fp,"r");
                byte[] encode=new byte[50];
                byte[] samplecode=new byte[50];
                byte[] conc=new byte[50];
                byte[] inten=new byte[50];
                byte[] stream=new byte[1];

                int c=0;
                if(c!=(-1))
                {
                        int i=0;
                        while ( (c=in.read(stream)) !=(-1) && stream[0]!=comma)
                        {

                                encode[i]=stream[0];
                                i++;
                        }
                }
                if(c!=(-1))
                {
```

```
            int i=0;
            while ( (c=in.read(stream)) !=(-1) && stream[0]!=comma)
            {

                        samplecode[i]=stream[0];
                        i++;
            }
}
if(c!=(-1))
{
            int i=0;
            while ( (c=in.read(stream)) !=(-1) && stream[0]!=comma)
            {

                        conc[i]=stream[0];
                        i++;
            }
}
if(c!=(-1))
{
            int i=0;
            while ( (c=in.read(stream)) !=(-1) && stream[0]!=comma)
            {

                        inten[i]=stream[0];
                        i++;
            }
}
en_code=new String(encode).trim();
sample_code=new String(samplecode).trim();
String concs=new String(conc).trim();
String intens=new String(inten).trim();

try
{
            concentration=Float.parseFloat(concs);
}
catch (NumberFormatException e)
{
            System.out.println("Could not get concentration.");
}
try
{
            tarinten=Float.parseFloat(intens);
}
catch (NumberFormatException e)
{
            System.out.println("Could not get intensity.");
}
if (outnum==9)
{
            if (tarinten==4.61F) tarinten=1.0F;
            else if (tarinten==9.25F) tarinten=2.0F;
            else if (tarinten==17.07F) tarinten=3.0F;
            else if (tarinten==28.40F) tarinten=4.0F;
            else if (tarinten==43.37F) tarinten=5.0F;
```

```
                    else if (tarinten==63.97F) tarinten=6.0F;
                    else if (tarinten==84.72F) tarinten=7.0F;
                    else if (tarinten==108.46F) tarinten=8.0F;
                    else tarinten=0.0F;
                }

                inunit[0]=1.0F;
                for (int i=1; i<innum; i++)
                {
                        int j=0;
                        byte[] sensor=new byte[50];
                        while ( (c=in.read(stream)) !=(-1) && stream[0]!=comma)
                        {

                                sensor[j]=stream[0];
                                j++;
                        }
                        String sensdata=new String(sensor).trim();
                        try
                        {
                                inunit[i]=Float.parseFloat(sensdata);
                        }
                        catch (NumberFormatException e)
                        {
                                System.out.println("Could not get sensor data.");
                        }
                }
                in.close();


        }
        catch (FileNotFoundException e)
        {
                System.out.println("File not found.");
        }
}

public void ReadIns(File fp, float[] target) throws IOException
{
        try
        {

                RandomAccessFile in=new RandomAccessFile(fp,"r");
                byte[] encode=new byte[50];
                byte[] samplecode=new byte[50];
                byte[] conc=new byte[50];
                byte[] inten=new byte[50];
                byte[] stream=new byte[1];

                int c=0;
                if(c!=(-1))
                {
                        int i=0;
                        while ( (c=in.read(stream)) !=(-1) && stream[0]!=comma)
                        {
```

```
                    encode[i]=stream[0];
                    i++;
            }
}
if(c!=(-1))
{
            int i=0;
            while ( (c=in.read(stream)) !=(-1) && stream[0]!=comma)
            {

                    samplecode[i]=stream[0];
                    i++;
            }
}
if(c!=(-1))
{
            int i=0;
            while ( (c=in.read(stream)) !=(-1) && stream[0]!=comma)
            {

                    conc[i]=stream[0];
                    i++;
            }
}
if(c!=(-1))
{
            int i=0;
            while ( (c=in.read(stream)) !=(-1) && stream[0]!=comma)
            {

                    inten[i]=stream[0];
                    i++;
            }
}

String concs=new String(conc).trim();
String intens=new String(inten).trim();

try
{
            target[0]=Float.parseFloat(concs);
}
catch (NumberFormatException e)
{
            System.out.println("Could not get concentration.");
}
try
{
            target[1]=Float.parseFloat(intens);
}
catch (NumberFormatException e)
{
            System.out.println("Could not get intensity.");
}
if (outnum==9)
{
```

```
                    target[1]=Math.round(target[1]/11.338);
                }

                inunit[0]=1.0F;
                for (int i=1; i<innum; i++)
                {
                        int j=0;
                        byte[] sensor=new byte[50];
                        while ( (c=in.read(stream)) !=(-1) && stream[0]!=comma)
                        {

                                sensor[j]=stream[0];
                                j++;
                        }
                        String sensdata=new String(sensor).trim();
                        try
                        {
                                inunit[i]=Float.parseFloat(sensdata);
                        }
                        catch (NumberFormatException e)
                        {
                                System.out.println("Could not get sensor data.");
                        }
                }
                in.close();


        }
        catch (FileNotFoundException e)
        {
                System.out.println("File not found.");
        }
}

public void RandomNet()  // randomize a net
{
        for(int i=0; i<hidnum; i++)
                for(int j=0; j<innum; j++)
                {
                        hidw[i][j]=(float)(Math.random()-0.5F) / 50.0F;
                        oldhidw[i][j]=0;
                }
        for (int i=0; i<outnum; i++)
                for(int j=0; j<hidnum; j++)
                {
                        outw[i][j]=(float)(Math.random()-0.5F) / 50.0F;
                        oldoutw[i][j]=0;
                }
}
public void ForwardNet()  // forward the inputs to the net, calculate the outputs of the net
{
        inunit[0]=1.0F;                 // a constant input X0
        for (int i=1; i<hidnum; i++)
        {
                float sum=0.0F;
                for (int j=0; j<innum; j++)
```

```
                        sum+=inunit[j]*hidw[i][j];
                hidunit[i]=1.0F / ( 1.0F + (float)Math.exp( (double)(-sum)) );
        }
        hidunit[0]=1.0F;  // a constant intput X0
        for (int i=0; i<outnum; i++)
        {
                float sum=0.0F;
                for (int j=0; j<hidnum; j++)
                        sum+=hidunit[j]*outw[i][j];
                outunit[i]=1.0F / ( 1.0F + (float)Math.exp( (double)(-sum)) );
        }
}
public float UpdateNet()   // update the weights
{
        for (int i=0; i<outnum; i++)
                tarout[i]=0.1F;
        int idx=0;

        if (concorinten==true)
        {
                idx=Math.round( (float)(Math.log10(concentration))/increment );
        }
        else
        {
                        idx=Math.round(tarinten);
        }

        tarout[idx]=0.9F;               // get the target outputs

        float seq=0.0F;                         // sum of square of the output units and target output
        for (int i=0; i<outnum; i++)            // calculate the error for each output unit
        {
                erro[i]=outunit[i]*(1.0F-outunit[i])*(tarout[i]-outunit[i]);
                seq+=(tarout[i]-outunit[i])*(tarout[i]-outunit[i]);
        }
        for (int i=1; i<hidnum; i++)            // calculate the error for each hidden units
        {
                float sum=0.0F;
                for (int j=0; j<outnum; j++)
                        sum+=outw[j][i]*erro[j];
                errh[i]=hidunit[i]*(1.0F-hidunit[i])*sum;
        }
        for (int i=1; i<hidnum; i++)            // update weights for hidden_input
                for(int j=0; j<innum; j++)
                {
                        float deltaw=learningrate*errh[i]*inunit[j]+momentum*oldhidw[i][j];
                        oldhidw[i][j]=deltaw;
                        hidw[i][j]+=deltaw;
                }
        for (int i=0; i<outnum; i++)            // update weights for ouput_hidden
                for(int j=0; j<hidnum;j++)
                {
                        float deltaw=learningrate*erro[i]*hidunit[j]+momentum*oldoutw[i][j];
                        oldoutw[i][j]=deltaw;
                        outw[i][j]+=deltaw;
                }
```

```
            return seq;         // return the sum of square
        }
        public float NetOutput(float[] buf)  // get the net's prediction
        {
            float max=0.0F;
            int idx=0;
            for (int i=0; i<buf.length; i++)
            {
                if (buf[i]>max)
                {
                    max=buf[i];
                    idx=i;
                }
            }
            float predicted=0.0F;

            if (concorinten==true)
            {
                predicted=(float)Math.pow(10, idx*increment);
            }
            else
            {
                predicted=idx;
            }
            return predicted;
        }

        public void TrainNet() throws IOException  // train a net
        {

            int k=0;

            String snap=new String();           // the screen snap string
            while (k< (epochs+1) && stoptraining==false )
            {
                float sse=0.0F;                 // the sum of square on a whole epoch
                for (int i=0; i<fs.length; i++)     // for each instance, update weights
                {
                    ReadIns(fs[i]);
                    ForwardNet();
                    sse+=UpdateNet();

                }
                if ( (k%10) == 0) // output accuracy, SSE, and MSE for each 10 epochs
                {
                    int cornum=0;
                    for (int i=0; i<fs.length; i++)
                    {
                        ReadIns(fs[i]);
                        ForwardNet();
                        if (concorinten==true)
                        {
                            if ( Math.abs( Math.log10(concentration) -
Math.log10(NetOutput(outunit)) ) <=(increment/2.0f) )
                                cornum++;
                        }
```

```
                                        else
                                        {
                                                if ( Math.abs( tarinten - NetOutput(outunit) ) <=0.5F )
                                                        cornum++;
                                        }

                                }
                                float acc=(float)cornum / (float)fs.length *100.0F; // get the accuracy
                                float mse=sse/(float)fs.length;                     // get the MSE


                                if (concorinten==true)
                                {
                                        snap="Epoch="+k+":      SSE="+sse+",      MSE="+mse+",
Concentration accuracy("+cornum+"/"+fs.length+")="+acc+"%";
                                }
                                else
                                {
                                        snap="Epoch="+k+":      SSE="+sse+",      MSE="+mse+",
Intensity accuracy("+cornum+"/"+fs.length+")="+acc+"%";
                                }
                                ndf.DisplayMessage(snap);

                                if ( isout==true)
                                {
                                        snap+="\r\n";
                                        SaveOutputTo(outfile, snap); // save the screen output into a
                                //file
                                }
                        }
                        k++;
                }

        }
        public void TestNet() throws IOException    // test a net
        {
                float acc=0.0F;
                float sumerr=0.0F;
                int cornum=0;

                String snap=new String();

                for (int i=0; i<fs.length; i++)                 // for each instance, test the net
                {
                        ReadIns(fs[i]);
                        ForwardNet();
                        float nnout=NetOutput(outunit);
                        float err=0.0f;
                        if (concorinten==true)
                        {
                                err=(float)Math.log10(nnout)-(float)Math.log10(concentration);
                        }
                        else
                        {
                                err=nnout-tarinten;
                        }
```

```
                    float abserr=Math.abs(err);
                    if (concorinten==true)
                    {
                            if (abserr<=increment/2.0f)
                                    cornum++;
                    }
                    else
                    {
                            if (abserr <= 0.5F )
                                    cornum++;
                    }
                    sumerr+=abserr;

                    if (concorinten==true)
                    {
                            snap=fs[i].getName()+",          "+en_code+",          "+sample_code+":
                    concentration="+concentration
                                    +"(log10="+Math.log10(concentration)+
                                                                    "),NN
                    concentration="+nnout+"(log10="+Math.log10(nnout)+") ,  error="+err;
                    }
                    else
                    {
                            snap=fs[i].getName()+",          "+en_code+",          "+sample_code+":
                            intensity="+tarinten
                            +" ,  NN intensity="+nnout+" ,  error="+err;
                    }
                    ndf.DisplayMessage(snap);
                    if (isout==true)
                    {
                            snap+="\r\n";
                            SaveOutputTo(outfile, snap);
                    }
            }
            int total=fs.length;
            acc=(float)cornum / (float)total *100.0F;

            float mse=sumerr / (float)total;
            snap="Summary:  accuracy "+cornum+"/"+total+" ="+acc+"%,   mean error="+mse;
            ndf.DisplayMessage(snap);
            if (isout==true)
            {
                    snap+="\r\n";
                    SaveOutputTo(outfile, snap);
            }
    }

    public void NetPredict(File[] testfs) throws IOException        // net prediction
    {
            String snap=new String();

            for (int i=0; i<testfs.length; i++)                // for each instance, test the net
            {
                    ReadIns(testfs[i]);
                    ForwardNet();
                    float nnout=NetOutput(outunit);
```

```
                        if(concorinten==true)
                        {
                                snap=testfs[i].getName()+",  "+en_code+",  "+sample_code+":     NN
concentration="+nnout+" (log10="+
                                        Math.log10(nnout)+")";
                        }
                        else
                        {
                                snap=testfs[i].getName()+",  "+en_code+",  "+sample_code+":     NN
intensity="+nnout;
                        }
                        ndf.DisplayMessage(snap);

                        if (isout==true)
                        {
                                snap+="\r\n";
                                SaveOutputTo(outfile, snap);
                        }
                }
        }

        public void SaveOutputTo(String fp, String con) throws IOException    // save screen output into a
//file
        {
                File f=new File(fp);
                RandomAccessFile fout=new RandomAccessFile(f, "rw");
                fout.seek(f.length());
                fout.writeBytes(con);
                fout.close();
        }

        public void SaveNet(File f) throws IOException        // save a net into a file
        {
                RandomAccessFile fout=new RandomAccessFile(f, "rw");
                fout.writeBoolean(concorinten);
                fout.writeInt(innum);
                fout.writeInt(hidnum);
                fout.writeInt(outnum);
                fout.writeInt(epochs);
                fout.writeFloat(learningrate);
                fout.writeFloat(momentum);
                for (int i=0; i<hidnum; i++)
                        for (int j=0; j<innum; j++)
                                fout.writeFloat(hidw[i][j]);
                for (int i=0; i<outnum; i++)
                        for (int j=0; j<hidnum; j++)
                                fout.writeFloat(outw[i][j]);
                fout.close();
        }
        public void LoadNet(File f) throws IOException        // load a net from a file
        {
                try
                {
                        RandomAccessFile fin=new RandomAccessFile(f,"r");
                        concorinten=fin.readBoolean();
```

```
                innum=fin.readInt();
                hidnum=fin.readInt();
                outnum=fin.readInt();
                epochs=fin.readInt();
                learningrate=fin.readFloat();
                momentum=fin.readFloat();
                increment=5.0f/ (float)(outnum);

                inunit=new float[innum];
                hidunit=new float[hidnum];
                outunit=new float[outnum];
                tarout=new float[outnum];
                hidw=new float[hidnum][innum];
                outw=new float[outnum][hidnum];
                oldhidw=new float[hidnum][innum];
                oldoutw=new float[outnum][hidnum];
                errh=new float[hidnum];
                erro=new float[outnum];

                for(int i=0; i<hidnum; i++)
                        for (int j=0; j<innum; j++)
                                hidw[i][j]=fin.readFloat();
                for (int i=0; i<outnum; i++)
                        for (int j=0; j<hidnum; j++)
                                outw[i][j]=fin.readFloat();
                fin.close();
        }
        catch (FileNotFoundException e)
        {
                isload=false;
                System.out.println("File not found.");
        }
}

public void EarlyStopping(File[] tf) throws IOException //Training a net
                        //with considering the accuracy on a validation data set
{
        int k=0;

        String snap=new String();               // the screen snap string
        while (k< (epochs+1) && stoptraining==false )
        {
                float sse=0.0F;                         // the sum of square on a whole epoch
                for (int i=0; i<fs.length; i++)         // for each instance, update weights
                {
                        ReadIns(fs[i]);
                        ForwardNet();
                        sse+=UpdateNet();

                }
                if ( (k%10) == 0) // output accuracy, SSE, and MSE for each 10 epochs
                {
                        int cornum=0;
                        for (int i=0; i<fs.length; i++)
                        {
                                ReadIns(fs[i]);
```

```
                                        ForwardNet();
                                        if (concorinten==true)
                                        {
                                                if ( Math.abs( Math.log10(concentration) -
Math.log10(NetOutput(outunit)) ) <=(increment/2.0f) )
                                                        cornum++;
                                        }
                                        else
                                        {
                                                if ( Math.abs( tarinten - NetOutput(outunit) ) <=0.5F )
                                                        cornum++;
                                        }

                                }
                                float acc=(float)cornum / (float)fs.length *100.0F; // get the accuracy
                                float mse=sse/(float)fs.length;                         // get the MSE

                                float[] tar=new float[2];
                                int testcornum=0;
                                for (int i=0; i<tf.length; i++)
                                {
                                        ReadIns(tf[i], tar);
                                        ForwardNet();
                                        if (concorinten==true)
                                        {
                                                if (Math.abs(Math.log10(tar[0]) -
Math.log10(NetOutput(outunit)) ) <= increment )
                                                        testcornum++;
                                        }
                                        else
                                        {
                                                if ( Math.abs( tar[1]- NetOutput(outunit) ) <=0.5F )
                                                        testcornum++;
                                        }

                                }
                                float testacc=(float)testcornum / (float)tf.length *100.0F; // calculate
//the accuracy on a validation data set

                                if (concorinten==true)
                                {
                                        snap="Epoch:"+k+", SSE:"+sse+", MSE:"+mse+",
Concentration accuracy("+cornum+"/"+fs.length+")%:"+acc+", Test accuracy
("+testcornum+"/"+tf.length+")%:"+testacc+",";
                                }
                                else
                                {
                                        snap="Epoch:"+k+", SSE:"+sse+", MSE:"+mse+", Intensity
accuracy("+cornum+"/"+fs.length+")%:"+acc+", Test accuracy
("+testcornum+"/"+tf.length+")%:"+testacc+",";
                                }
                                ndf.DisplayMessage(snap);

                                if ( isout==true)
                                {
                                        snap+="\r\n";
```

```
                                        SaveOutputTo(outfile, snap); // save the screen output into a
//file
                                    }
                                }
                                k++;
                            }
                        }
                    }
```

Appendix B: Original Sensor Responses of Training Data

Concentration (*ppm*)	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5	Sensor 6	Sensor 7	Sensor 8	Sensor 9	Sensor 10	Sensor 11	Sensor 12
120	0.181483	-0.090749	-0.171884	-0.066736	-0.072727	-0.014739	0.235294	0.18133	0.153946	0.177755	0.1875	0.257068
120	0.184828	-0.092128	-0.178426	-0.06455	-0.073814	-0.014164	0.232484	0.17861	0.150694	0.17517	0.185499	0.253517
120	0.134945	-0.078534	-0.182366	-0.056308	-0.060937	-0.012493	0.228213	0.174406	0.14841	0.168967	0.180446	0.249513
120	0.18594	-0.094878	-0.185078	-0.06596	-0.074898	-0.014739	0.233861	0.178977	0.150531	0.175873	0.185378	0.255361
240	0.159768	-0.152473	-0.309237	-0.108672	-0.117861	-0.022714	0.254586	0.189189	0.162863	0.188347	0.207754	0.274113
240	0.167163	-0.1576	-0.319751	-0.112299	-0.121359	-0.023282	0.260063	0.193697	0.166185	0.19258	0.2121	0.279702
240	0.168057	-0.159391	-0.325294	-0.112834	-0.121892	-0.023295	0.260013	0.19315	0.164796	0.192398	0.211716	0.279549
240	0.141404	-0.14934	-0.332388	-0.107065	-0.115003	-0.022147	0.255726	0.188298	0.161954	0.186469	0.208033	0.27379
480	0.201328	-0.305498	-0.554545	-0.221263	-0.239875	-0.044886	0.305485	0.21895	0.186473	0.224848	0.261454	0.320429
480	0.20071	-0.314182	-0.567584	-0.226537	-0.245459	-0.045455	0.309114	0.221201	0.188062	0.228159	0.264726	0.324185
480	0.205091	-0.320764	-0.579976	-0.231267	-0.251556	-0.046023	0.313144	0.224612	0.190903	0.231724	0.268376	0.328707
480	0.195138	-0.301638	-0.57574	-0.21926	-0.235879	-0.044293	0.310182	0.221434	0.188807	0.226676	0.265727	0.324333
960	0.322513	-0.56341	-0.849626	-0.425176	-0.454026	-0.088687	0.375551	0.265791	0.218609	0.282631	0.340342	0.385041
960	0.325902	-0.576245	-0.863966	-0.43544	-0.465455	-0.089205	0.376937	0.267208	0.21825	0.284228	0.341772	0.386452
960	0.333093	-0.589027	-0.877844	-0.444924	-0.47535	-0.090341	0.382022	0.271413	0.221653	0.289209	0.347313	0.391707
960	0.311305	-0.566772	-0.876139	-0.429658	-0.456681	-0.089205	0.379603	0.267496	0.219504	0.283892	0.344008	0.387549
1940	0.515493	-1.017408	-1.264722	-0.830323	-0.856117	-0.197727	0.472739	0.338538	0.262073	0.371457	0.455171	0.471182
1940	0.513875	-1.016407	-1.261473	-0.82806	-0.854545	-0.196134	0.473578	0.338983	0.262692	0.372483	0.454449	0.471783
1940	0.519003	-1.029909	-1.274144	-0.839835	-0.867016	-0.200228	0.474539	0.341093	0.263529	0.374916	0.456811	0.473905
1940	0.495023	-0.997837	-1.264137	-0.811915	-0.837386	-0.193182	0.47146	0.336771	0.261108	0.368332	0.452888	0.469802
3880	0.64108	-1.455379	-1.654952	-1.295538	-1.370967	-0.4	0.56928	0.417145	0.308113	0.463185	0.568306	0.553045
3880	0.643674	-1.46118	-1.661003	-1.300981	-1.377168	-0.401364	0.569883	0.417668	0.308462	0.464694	0.568406	0.554038
3880	0.646594	-1.485189	-1.678034	-1.321823	-1.398635	-0.4124	0.573811	0.420904	0.311095	0.469181	0.572987	0.557653
3880	0.628583	-1.44495	-1.663778	-1.286888	-1.35274	-0.398522	0.570654	0.41756	0.308511	0.462831	0.569151	0.553817
7750	0.730564	-1.966953	-2.064644	-1.723109	-1.874247	-0.717045	0.661013	0.498269	0.356314	0.555233	0.673789	0.632425
7750	0.732329	-1.988017	-2.079858	-1.741155	-1.89488	-0.727686	0.663339	0.500931	0.358215	0.5584	0.676088	0.63497
7750	0.732127	-1.997453	-2.085126	-1.749442	-1.903764	-0.731097	0.664273	0.502461	0.358974	0.560224	0.677395	0.636219
7750	0.729291	-1.968729	-2.081236	-1.726238	-1.876347	-0.722317	0.663156	0.499814	0.357097	0.556261	0.675823	0.634091
15550	0.797784	-2.498684	-2.48123	-2.292524	-2.336498	-1.101283	0.743883	0.580336	0.407479	0.641792	0.764501	0.708854
15550	0.795795	-2.492841	-2.473664	-2.287756	-2.336552	-1.094723	0.743439	0.580333	0.407748	0.641844	0.763818	0.708918
15550	0.794775	-2.500205	-2.471034	-2.292827	-2.32846	-1.094066	0.742909	0.580267	0.407764	0.642128	0.763426	0.708863
15550	0.79749	-2.480159	-2.479023	-2.280436	-2.339451	-1.095501	0.743123	0.579573	0.406008	0.640091	0.763557	0.707653

Appendix C: Original Sensor Responses (Relative Changes in Electrical Resistance) of Testing Data

Concentration (*ppm*)	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5	Sensor 6	Sensor 7	Sensor 8	Sensor 9	Sensor 10	Sensor 11	Sensor 12
5	0.184144	-0.019988	-0.040041	-0.016546	-0.017606	-0.005129	0.206484	0.166132	0.135925	0.157769	0.157913	0.230416
5	0.160402	0.008117	0.013283	-0.006041	-0.006332	-0.00234	0.192883	0.156306	0.125681	0.144559	0.145671	0.217158
5	0.181915	-0.020432	-0.043686	-0.013048	-0.018129	-0.005659	0.205975	0.166015	0.135152	0.157734	0.157695	0.230267
5	0.178005	-0.020349	-0.045321	-0.013971	-0.018097	-0.005093	0.206961	0.165957	0.135213	0.158322	0.158003	0.231094
10	0.15011	-0.011856	-0.024929	-0.010735	-0.011654	-0.004538	0.197017	0.160113	0.130407	0.150708	0.150068	0.221862
10	0.148457	-0.012807	-0.028436	-0.010707	-0.012289	-0.003402	0.201923	0.163489	0.13344	0.154053	0.15361	0.226598
10	0.15625	0.008161	0.012821	-0.006604	-0.006875	-0.002836	0.19317	0.15696	0.126643	0.145296	0.14617	0.217458
10	0.144242	-0.012768	-0.030303	-0.01123	-0.011543	-0.003401	0.203033	0.164258	0.134419	0.154698	0.154657	0.228149
25	0.144095	-0.01844	-0.040207	-0.014092	-0.015658	-0.005108	0.202292	0.163013	0.133846	0.152941	0.154474	0.226611
25	0.137491	-0.018868	-0.043503	-0.014602	-0.016154	-0.005108	0.205691	0.165315	0.136555	0.156166	0.157269	0.230596
25	0.156992	-0.015259	-0.032032	-0.012115	-0.012169	-0.003973	0.19963	0.160099	0.131132	0.149206	0.151967	0.22291
25	0.129416	-0.019864	-0.045479	-0.015127	-0.016129	-0.00454	0.204138	0.163617	0.134376	0.154567	0.156038	0.22825
50	0.14086	-0.030867	-0.070048	-0.022863	-0.024633	-0.006385	0.209288	0.166078	0.138008	0.156764	0.16117	0.232686
50	0.13306	-0.03125	-0.073228	-0.022814	-0.025105	-0.006243	0.210006	0.166197	0.138019	0.157578	0.161465	0.233591
50	0.123618	-0.031679	-0.076613	-0.0239	-0.025078	-0.005675	0.210494	0.166374	0.138162	0.157701	0.16175	0.233954
50	0.159473	-0.027793	-0.065796	-0.020925	-0.022222	-0.005675	0.206762	0.163591	0.135343	0.153235	0.159153	0.229031
100	0.144132	-0.057219	-0.133755	-0.041462	-0.044094	-0.009075	0.224046	0.172792	0.146013	0.165325	0.17477	0.245063
100	0.134448	-0.061235	-0.14484	-0.043573	-0.047195	-0.010789	0.224861	0.173469	0.145594	0.166333	0.175534	0.245981
100	0.124226	-0.060542	-0.145821	-0.043549	-0.046073	-0.010221	0.226223	0.173882	0.146599	0.166778	0.176231	0.246684
100	0.163557	-0.055525	-0.132801	-0.0413	-0.042834	-0.010216	0.221352	0.170628	0.143407	0.161969	0.173175	0.241762
250	0.162125	-0.159164	-0.346645	-0.114145	-0.122438	-0.024404	0.266762	0.195381	0.166773	0.192694	0.218155	0.282616
250	0.151886	-0.146667	-0.32016	-0.105837	-0.113326	-0.022134	0.264715	0.194161	0.16656	0.191923	0.214431	0.282015
250	0.135925	-0.149015	-0.336331	-0.10687	-0.11484	-0.022727	0.263859	0.193446	0.166187	0.191909	0.214213	0.281159
250	0.18674	-0.143013	-0.323674	-0.104018	-0.111052	-0.022134	0.263142	0.192758	0.16565	0.189511	0.214893	0.279297
500	0.205296	-0.288841	-0.55167	-0.211245	-0.226772	-0.0437	0.313481	0.22183	0.18854	0.226206	0.267165	0.325348
500	0.202206	-0.303514	-0.580102	-0.221374	-0.237546	-0.045997	0.318644	0.225545	0.190732	0.230641	0.27168	0.331083
500	0.184202	-0.297398	-0.574216	-0.216894	-0.232059	-0.044318	0.312325	0.221595	0.18816	0.226984	0.266417	0.325266
500	0.233184	-0.286255	-0.561208	-0.21044	-0.225568	-0.044861	0.314356	0.222222	0.189467	0.22655	0.269479	0.326059
1000	0.336515	-0.577046	-0.898603	-0.440327	-0.466213	-0.093697	0.391723	0.274558	0.223752	0.290312	0.355321	0.39754
1000	0.325893	-0.572487	-0.896296	-0.436344	-0.462343	-0.092561	0.388101	0.272183	0.221754	0.288577	0.351186	0.394982
1000	0.314655	-0.589012	-0.904605	-0.449402	-0.476987	-0.094886	0.390479	0.274815	0.224392	0.291653	0.354912	0.397349
1000	0.350248	-0.560451	-0.896507	-0.426616	-0.451579	-0.091373	0.387307	0.270952	0.221259	0.285714	0.352741	0.393416
2500	0.537733	-1.124175	-1.434465	-0.958086	-0.976427	-0.250851	0.515427	0.367804	0.279152	0.402492	0.502547	0.506216
2500	0.545639	-1.105678	-1.417294	-0.962241	-0.984165	-0.252129	0.514434	0.367751	0.279449	0.403356	0.499719	0.50526
2500	0.543824	-1.117144	-1.428968	-0.973586	-0.995607	-0.254401	0.515581	0.369979	0.280735	0.40602	0.500795	0.507312
2500	0.53516	-1.109047	-1.419825	-0.966107	-0.98801	-0.24929	0.508863	0.36518	0.277244	0.402083	0.495052	0.501207
5000	0.65473	-1.603808	-1.818791	-1.44319	-1.514643	-0.49773	0.610287	0.446833	0.32421	0.493532	0.612377	0.585497
5000	0.667467	-1.628417	-1.82345	-1.467	-1.541793	-0.509081	0.61237	0.450854	0.327249	0.498716	0.614155	0.588407
5000	0.670152	-1.644094	-1.836484	-1.480937	-1.557422	-0.515616	0.613904	0.453085	0.328649	0.501771	0.615751	0.590488
5000	0.670256	-1.646202	-1.838702	-1.483272	-1.557713	-0.513913	0.610423	0.451155	0.327559	0.500525	0.613713	0.587563
10000	0.748298	-2.130933	-2.236274	-1.883386	-2.035364	-0.854793	0.701031	0.53174	0.375309	0.587489	0.714995	0.665955
10000	0.755896	-2.149794	-2.23503	-1.912155	-2.055598	-0.864305	0.701577	0.533822	0.377443	0.590814	0.715366	0.667629
10000	0.758176	-2.162929	-2.243242	-1.924964	-2.070265	-0.868892	0.700983	0.534284	0.377117	0.591823	0.715213	0.667656
10000	0.759784	-2.18563	-2.260238	-1.945973	-2.087385	-0.879132	0.701759	0.535909	0.378923	0.59423	0.717278	0.668491
20000	0.812937	-2.622949	-2.62094	-2.449989	-2.440144	-1.227805	0.77411	0.610065	0.426277	0.668585	0.79371	0.736506
20000	0.815491	-2.633955	-2.607628	-2.458904	-2.450139	-1.229841	0.772925	0.609794	0.426555	0.669082	0.792687	0.736202
20000	0.816929	-2.648644	-2.613933	-2.47142	-2.462714	-1.233707	0.773042	0.610821	0.427383	0.670486	0.793008	0.736901
20000	0.81912	-2.674408	-2.634555	-2.495909	-2.486381	-1.245031	0.773551	0.612321	0.428986	0.672608	0.79474	0.737706

Appendix D: Input Data Files to ENBrain

The data files are included in the attached CD.

The data file for one n-butanol sample:

F1_10,1,120,4.61,0.181483,-0.090749,-0.171884,-0.066736,-0.072727,-

0.014739,0.235294,0.18133,0.153946,0.177755,0.1875,0.257068

In the above data file, *F1_10* is EN_code, *1* is sample code, *120* is the concentration, *4.61* is the intensity, and the followed 12 numbers are sensor responses. There are 80 data files used for training and testing ENBrain.