UNIVERSITY OF BAHRAIN

College of Information Technology



Electronic Nose Based on an Artificial Neural Network with Multimedia Applications

A Thesis Submitted in Partial Fulfilment of the

Requirements for the Master's Degree in Information Technology

Submitted by

Shaima Ebraheem Hijris

University number:19990190

Supervised by

Dr. Yousif Abdul Latif Nasrulla Al-Bastaki

(Associate Professor)

University of Bahrain

Kingdom of Bahrain

May 2015

DECLARATION

I declare that this work is the result of my own investigation and that it has not already been accepted in substance for any degree, nor is it currently submitted for any degree.

Signed: ---------- Date: 26/11/2015

Shaima Ebrahim Hijris (Candidate)

The Thesis defense committee considers the Thesis titled:

Electronic Nose Based on an Artificial Neural Network with Multimedia Applications

Submitted by Shaima Ebrahim Hijris to the College of Information Technology is satisfactory and acceptable for the Master of Science in Information Technology.

Thesis Defense Committee:

Dr. Yousif Al-Bastaki
(Supervisor)

Department of Computer Science
University of Bahrain

Dr. Syed Afaq Husain
(Member)

King Faisal University
Saudi Arabia

Dr. Ali Khan
(Member)

Department of Computer Science
University of Bahrain

Abstract

This study proposes a prototype system for an electronic nose handheld odour meter (OMX-GR sensor) to recognise odours of three types of meat and whether it is good or bad. An artificial neural network (ANN) is the core of the system using software in C# with Visual Studio 2010 to analyse the data and recognise the odours. The ANN uses a backpropagation multi-layer algorithm with certain parameters for momentum and learning rates with supervised learning and the sigmoid activation function. The data input to the ANN software must be prepared in a special format.

The ANN is needed here in order to decide what is the best architecture selection to be applied, after many training structures the winner structure is Consequently ratio of 2:6:1 with a validation of the least mean of sum of square error at 0.03, 1.72729 training error, 0.77849 validation, and 0.03% testing. For such results the 2:6:1 structure considered as the best ANN architecture for the problem solution which had the less validation result.

In the future, mimic smell sense development will be applied and integrated into computer and smart devices. Accordingly, the smell sense is believed to a new element that to be added to major elements in multimedia.

Acknowledgments

First, I would like to thank God for helping me and supporting the completion of my thesis and giving me the energy, ability, and determination to work hard.

I want to thank my father, Ebraheem Jumaan Hijres, and my mother, Zamzam Khalifa Hijris, for supporting me, encouraging me, believing in me, and doing whatever they could to make me comfortable to finish this thesis.

I would like to thank my husband, Nawaf Alghanem, for supporting me always and assisting me to finish this learning journey, and I appreciate his bearing of responsibility and tolerance.

I would like to thank my supervisor, Dr. Yousif Abdul Latif ALbastaki, for his support throughout my entire journey and for encouragement to choose the subject of my thesis.

I would like to thank my family and friends for all the support.

Table of Contents

Topics	Page No.
Abstract	II
Acknowledgments..	III
Table of Contents	IV
Appendix Table	VII
List of Tables	VIII
List of Figures	IX
Abbreviation Table	XII
Chapter 1 **Introduction**	1
1.1 Introduction.	2
1.2 Statement of the Problem.	7
1.3 Question of the Thesis.	7
1.4 Objective of the Thesis.	8
1.5 The Value of the Study.	9
1.6 The Limitation of the Study.	10
1.7 Organization of the Thesis.	11
1.8 Definition of the Study.	12
Chapter 2 **Literature Review**	14
2.1 Studies of Odour sensing development.	15
2.2 Studies of odour-sensing with multimedia .	24

Chapter 3 **Electronic Nose and ANN**	32
3.1. Electronic Nose.	33
3.1.1. The Olfactory System.	33
3.1.2. Smell Classification.	34
3.1.3. Smell Sensors.	36
3.1.3.1 semiconductors sensors.	37
3.1.3.2 Figaro sensors.	37
3.1.4 Electronics Sensors.	38
3.1.5 Data Processing Module.	39
3.2. Artificial Neural Network.	39
3.2.1 Neural Network Architecture.	41
3.2.2 Type of Learning.	44
3.2.3 Activation Function.	44
3.2.4 Backpropagation.	48
3.2.4.1 Other Major Components of the Artificial Neural Network.	52
3.2.5 Training Modes.	55
3.2.6 Least Mean Sum of Squares Error Function (SSE).	55
3.2.7 The Advantages of Using ANN.	56
Chapter 4 **Analysis and Design of a Prototype System**	57
4.1 The Proposed System.	59
4.2 Odour sensing system (e-nose).	59
4.3 Data Collection.	61

4.3.1 Data Description.	63
4.4 Data Preparation.	65
4.4.1 ID or Classification Input.	66
4.4.2 Strength Odour Input.	69
4.4.3 The Balance between Bad and Good Meat through Both Inputs.	71
4.4.3.1 The Difference between the Conditions of Chicken.	71
4.4.3.2 The Difference between the Conditions of Beef.	71
4.4.3.3The Difference between the Conditions of Mutton.	72
4.5 Artificial Neural Network structure.	73
4.5.1 Designing ANN Software.	74
4.5.2 Importing Training Data Sets.	76
4.5.3 The Flow Code of ANN.	81
4.5.4 Training Phase.	84
4.5.5 Performance Report.	85
4.6 Results.	86
4.7 Multimedia Elements.	89
Chapter 5 **Conclusion**	90
5.1 Discussion.	91
5.2 Conclusions.	94
5.3 Future Work.	95
References.	96

Appendix Table

Appendix 1: Code of network training.	
Appendix 2: Code of network testing.	
Appendix 3: Code of network validation.	
Appendix 4: Code of network query.	
Appendix 5: Code of learning graph.	
Appendix 6: Code of import wizard .	
Appendix 7: Code of meat decision based on network output.	
Appendix 8: Code of activation network.	
Appendix 9: Code of activation layers.	
Appendix 10: Code of activation neurons.	
Appendix 11: Code of sigmoid function.	
Appendix 12: Code of backpropagation learning algorithm.	

List of Tables

No	Topics	Page No.
1	Table (1): Linnaean And Zwaardemaker Smell Classifications (Castro, Mandal, Ajemba, & Istihad, 2003).	35
2	Table (2): LOVELL Smell Classifications (Castro, Mandal, Ajemba, & Istihad, 2003).	35
3	Table (3): Typical Material Used As Sensors (Castro, Mandal, Ajemba, & Istihad, 2003).	36
4	Table (4): List Of The Sensors Proposed (Castro, Mandal, Ajemba, & Istihad, 2003).	38
5	Table (5): Backpropagation Training Algorithm Notations And Working (Fausett, 1994).	51
6	Table (6): The Range Of The Classification Data For Each Type And State Of Meat.	66
7	Table (7): The Range Of The Strength Data For Each Type Of Meat.	69
8	Table (8): The Results For Training, Validation, And Testing Of ANN.	88

List of Figures

No	Figure	Page No.
1	Figure (1):Human Olfactory System (Castro, Mandal, Ajemba, & Istihad, 2003)	3
2	Figure (2):The electronic nose system which combine of ANN.	5
3	Figure (3):The Scope Of The Work.	11
4	Figure (4):The Process Of The Olfactory System (Al-Bastaki, 2009).	34
5	Figure (5):The Scope Of The Work (Fausett, 1994).	40
6	Figure (6):Simple ANN (Fausett, 1994).	41
7	Figure (7):Single-Layer Perceptrons (Feedforward Networks) (Fausett, 1994).	42
8	Figure (8):A Multilayer Neural Network (Fausett, 1994).	43
9	Figure (9):Activation Function (Sibi, Jones, & Siddharth, 2013).	45
10	Figure (10):Identity Function (Fausett, 1994).	45
11	Figure (11):Binary Threshold (Chakraborty, 2010).	46
12	Figure (12):Bipolar Threshold (Chakraborty, 2010).	46
13	Figure (13):Piecewise Linear (Chakraborty, 2010).	47
14	Figure (14):Binary Sigmoid Function (Fausett, 1994).	47
15	Figure (15):Bipolar Sigmoid Function (Fausett, 1994).	48
16	Figure (16):Backpropagation Neural Network (Fausett, 1994).	50
17	Figure (17):Add Bias In Neural Network.	53
18	Figure (18):The Back-Propagation Neural Network Epoch (Carney & Cunningham, 1998)	53
19	Figure (19):Handheld Odour Meter (OMX-GR Sensor)/Display Panel.	60
20	Figure (20):Select Between The Measurement Mode.	61
21	Figure (21):Creating a Data File.	61
22	Figure (22):Initialising the Data File to Transfer the	62

	Data.	
23	Figure (23):The File Data.	63
24	Figure (24):Example of the Modified Data Used in Coding.	64
25	Figure (25):Example of the Sampling Process.	65
26	Figure (26):The Results of Chicken Classification for Good and Bad Chicken.	67
27	Figure (27):The Results of Beef Classification for good and Bad Beef.	68
28	Figure (28):The Results of Mutton Classification for Good and Bad Mutton.	68
29	Figure (29):Comparison of Chicken Odour Strengths between Good and Bad Chicken.	69
30	Figure (30):Comparison of Beef Odour Strengths between Good and Bad Beef.	70
31	Figure (31):Comparison of Mutton Odour strengths between Good and Bad mutton.	70
32	Figure (32):Comparison of the Two Inputs between Good and Bad Chicken.	71
33	Figure (33):Comparison of the Two Inputs between Good and Bad Beef.	72
34	Figure (34):Comparison of the Two Inputs between Good and Bad Mutton.	72
35	Figure (35):Flow Diagram Showing The ANN Black Box System.	75
36	Figure (36):The Proposed Program Electronic Nose.	76
37	Figure (37):Choose Run Wizard to Import Data.	76
38	Figure (38):Import Data Wizard Window.	77

39	Figure (39):Selection of the Data Location.	77
40	Figure (40):Select Input.	78
41	Figure (41):Select Output.	78
42	Figure (42):Shuffle Icon in ANN.	79
43	Figure (43):Data Sample Ratio.	79
44	Figure (44):Create New Network.	80
45	Figure (45):Create and Set the Parameters of the Network.	80
46	Figure (46):Setting the Network Trainer Parameters.	81
47	Figure (47):Sinapse.Databases.	82
48	Figure (48):GUI Code Items.	82
49	Figure (49):Wizard Control.	83
50	Figure (50):AForge.NET.	83
51	Figure (51):AForge.NET and AForge.Neuro.	84
52	Figure (52):Training Graph.	85
53	Figure (53):Performance Report for ANN system.	86
54	Figure (54):The Best ANN Architecture is 2:6:1.	87
55	Figure (55):The Multimedia Elements Added into the Output of the ANN System.	89

Abbreviation Table

Word	Abbreviation
Artificial neural network	ANN
Electronic Nose	e-nose
Handheld odour meter (OMX-GR sensor).	OMX-GR sensor
Backpropagation	BP
self-organizing maps	SOM
learning vector quantizers	LVQ
Volatile organic compound	VOC
Static headspace extraction	SHS
Dynamic headspace extraction	DHS
Solid-Phase Micro Extraction	SPME
Metal Oxide Semiconductor	MOS
Metal Oxide Semiconductor Field-Effect Transistors	MOSFET
Conducting Polymer sensors	CP
quartz crystal microbalance	QCM
surface acoustic wave	SAW
principal components analysis	PCA
linear discriminate analysis	LDA
Near Infrared	NIR
Multi Layered Perception	MLP
Bulk Acoustic Wave	BAW
Surface Acoustic Wave	SAW
Radial Basis Function	RBF
Direct current	DC
Taguchi Gas Sensor	TGS
multilayer perceptrons	MLP

Chapter 1: Introduction

1.1 Introduction.

1.2 Statement of the Problem.

1.3 Question of the Thesis.

1.4 Objective of the Thesis.

1.5 The Value of the Study.

1.6 The Limitation of the Study.

1.7 Organization of the Thesis.

1.8 Definition of the Study.

1.1 Introduction

Over decades, many researchers proposed an Electronic Nose System had applied many different methods and tools in order to invent a devise to simulate the olfactory human system. Ultimately the invention of this smell sense simulator can be a great part in helping to improve quality of human life as in medical, security and environmental in the future.

Therefore, today technology is racing dreams. As a result of communication technology, the world has become a small place; people can communicate with each other using techniques based on vision and sound in real time. On the other hand, the uncommon use of the sense of smell as an element in communication could be a technology that is coming soon. Envisage your friend travelling to another country who wants you to live in the moment with him by going to a fine restaurant and allowing you to see and smell the food while he records it on various applications used for communication, such as Skype or Viber. You may want purchase perfume from Amazon or eBay or another online store; the perfect scenario would be that you could test the smell before buying it. To illustrate, imagine that you could smell the aroma on the photos you can browse online on Instagram, Facebook, Google, and WhatsApp. This technology deserves to be one of the dream future technologies that might or could win a world class prize as being an state of the art technology.

Therefore, the odour-sensing electronic nose EN technology could become a very significant tool. In 1982, the concept of an artificial nose system was introduced (Schaller, Bosset, & Escher, 1998); in the same year, the first gas multisensory array was invented (Wilson & Baietto, 2009), which applies to many sectors such as food, industry, perfume, market, agricultural, and others.

In order to understand the concept of odour sensing, the mechanism of the human olfactory system must be understood. It consists of olfactory receptors(olfactory Epithelium), which receive the chemical substance, an olfactory bulb, which produces signals to combine with the substance of the odour, and a mechanical recognition pattern in the olfactory cortex, which processes the information and performs a classification and recognition model of any detected odour. After the odour identification, it will transmit the information to the cerebral cortex, limbic, and hippocampus. The hippocampus is the main component that responds to the relationship between the sense of smell and the mind regarding memories Figure(1).

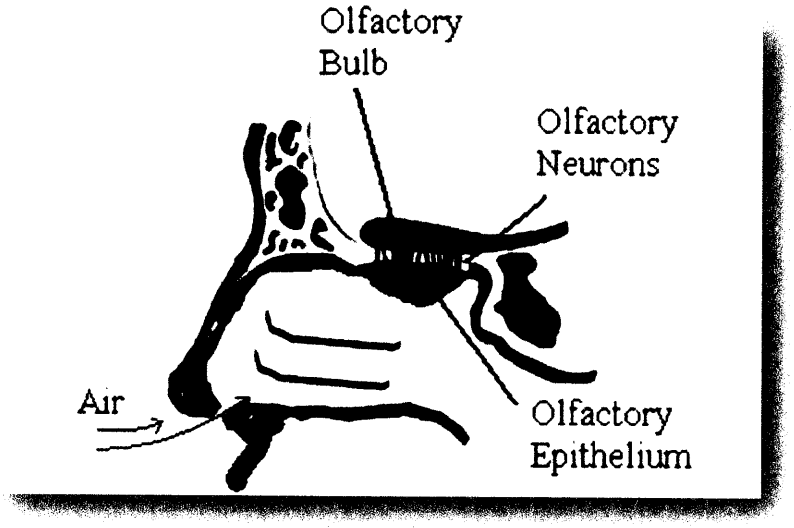


Figure (1)

Human Olfactory System (Castro, Mandal, Ajemba, & Istihad, 2003)

Odour sensing simulates the human smell senses, which can be represented in an array sensor; the sensor captures the smells that consist of the molecules based on the chemical substance. Each odour has its own fingerprint; in other words, every odour is formed of unique chemical elements, even if there are two odours in the same chemical substance

and even though they may have different percentages of concentration for each element; therefore, this will cause different smells (Al-Bastaki, An Artificial Neural Networks-Based on-Line Monitoring Odor Sensing System, 2009).

The human olfactory sense is not that simple. The immature understanding of smell sense is the major reason that it has been ignored. When we compare the sense of smell with an eye recognition or sound recognition systems or tools, the reason this sense is ignored is that the human olfactory system contains several million receptors of approximately 1,000 different types. These receptors together can detect up to 5,000 different smells with 98% accuracy (Julian W. Gardner, 1994).

Human smell detection is based on a combinational scheme, where a different number and type of smell receptors are activated in the existence of a particular smell. Small changes depend on the chemical construction of odours, which activate a different combination of receptors. In addition, even the same chemical structure with a different percentage can result in a different smell, unlike the human visual system, which has three types of cone cells (to detect three primary colours: red, green, and blue), which makes vision detection easier (Stuetz, Egnin, & Fenner, 1998; Castro, Mandal, Ajemba, & Istihad, 2003).

The efficient classification of odours is considered the core of any odour-sensing system Figure (2). On the other hand Artificial neural network (ANN) refers to machine learning and computational models.

Moreover, it can represent biological nerves of which the main unit is a neuron. All neurons are connected with each other through the network, feeding their knowledge and their data with the values that come from inputs. Three main layers comprise the system. The first layer has neurons, which receive data and send it to the second layer, which it is

called the computational layer, to classify the input, and the last layer presents the output data (Hoskins & Himmelblau, 1992; Keller, Kouzes, & Kangas, 1994; Bhadeshia, Dimitriu, Forsik, Pak, & Ryu, 2009).

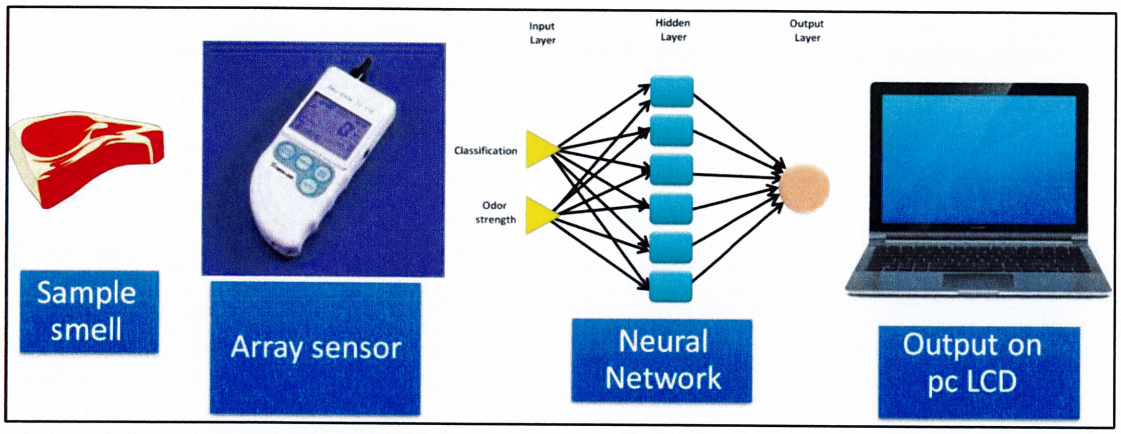


Figure (2)

The electronic nose system which combine of ANN

Odour sensing is associated with ANN to create a system that handles the expectations of the technology, according to Schaller this is the best method to solve problems when there is no inadequate information or data. In this case, human or expert knowledge must be converted into professional systems. This system will collect the information and match it with the expected output of various experimental models. In addition, the learning methods needs are different depending on the problem features (Schaller, Bosset, & Escher, 1998; Wasserman, 1993).

The combination of odour sensing and ANN exploration has increased over the last years (Macías, Agudo, & Manso, 2013), (Sibi, Jones, & Siddharth, 2013), (Rafiq, Bugmann, & Easterbrook, 2001). Moreover, the multimedia component could gain this system an upward trend. Additionally, multimedia is increasingly becoming a major factor in many system implementations that serve many sectors, such as in education, industries, medical, mathematical and scientific research,

disabilities, entertainment, and food (Castro, Mandal, Ajemba, & Istihad, 2003).

Multimedia is user-friendly and makes any system easy to use and learn. For example, instead of just listening to a presentation, you can hear, read, listen, and view it. It can be displayed, recorded, accessed by information content processing devices, and played to ensure the best efficient performance (Carraro, et al., 2007). Multimedia elements can be an integration of text, graphics, pictures, video, animation, audio, and any other media where every type of information can be represented, stored, transmitted, and processed digitally. Multimedia is a great tool that can be added to any system to perfectly present the output or the information, and can have added interactive and integrated factors to offer a simple feedback, especially for non-expert users. Flexibility factors can easily change in different media to fit many purposes. In particular odour element could be the new brilliant element which can enrich any system and provide the unlimited real life experiment.

The integration of the e-nose based on ANN and multimedia elements will create a powerful mechanism to develop smell detection technology. Multimedia is now extensively used in industry, consumer electronics environments, medical, food, and perfume, where individuals can work and communicate through multi-sensory interfaces. Unfortunately, smell detection is not part of today's multimedia systems. However, in the latest research, some of the researchers have given attention to the combination of the e-nose and multimedia (Castro, Mandal, Ajemba, & Istihad, 2003).

The proposed odour recognition will create a suitable environment that will produce a result similar to the real world with respect to the multimedia element, which could be a beneficial contribution to recent studies in the same area of concern.

1.2 Statement of the Problem

Sense of smell is very important function that assists humans to realise their environment. Nevertheless, many factors could hamper the smell sense as getting down with flu, imbalance in the vital functions and many other factors.

Under these circumstances, when the smell sense in a human could not detect the odours correctly it could cause a series of problems and may cause death. For example, in food factories the damage food produces bad smell, in some certain situations this bad smell could not be reorganised by the normal smell function of a human that can cause severe food poisoning or else if there is a precarious gas leak, since its possible for humans to not smell some kind of gases which can cause dangerous cases suffocation or even death. According, to many studies done that proven to have a great potential in using EN technology in many sectors to improve quality of life and services offered.

Therefore, this research study examines how to mimic the olfactory system through the electronic nose devise integration with ANN which can grant the system the ability to solve the computational problem in the same way the biological brain works, and to investigate the usage of the odour element as a new part of multimedia to increase performance and cover a wide range of sectors.

1.3 Question of the Thesis

Many studies examine the improvement of the electronic nose EN through merging it with various recognition pattern systems and has got diverse results. For this reason, two questions came up as a result of this research which is listed as follow:

First, the possibility of increasing the output performance of EN by merging it with other recognition pattern systems as a Artificial Neural Network ANN which considered as a powerful tool. ANN can achieve the increase in applications of the complicated computational system, which offers quality, efficiency and accuracy.

Secondly, many elements are composed of many multimedia applications like video, sound, animation, text and graphics. The question that should be asked here, if those results in the study with the odour sense can be added or integrated with other major multimedia elements. Certainly multimedia is increasingly becoming a major factor in many system implementations that serve many sectors (education, industries, medical, mathematical and scientific research, disabilities, entertainment, and food processing). Nevertheless, there are limitations to this tool with the odour sensing area, even though using multimedia tools in odour sensing will result in great benefits in several ways. The system proposed in this thesis is about designing a prototype of an odour-sensing device based on an ANN integrated with some elements of multimedia tools. After the implementing phase, the system must works efficiently at the final test stage and then will be compared with other similar research with the same aim but with different scenarios.

1.4 Objective of the Thesis

The main objectives of this research study are to develop the odour-sensing system to ensure the capability and improve the performance of the continuing evolution of the odour-sensing journey. To achieve the main objective, this research study will tackle the following:

- Investigate the development of odour-sensing system.
- To detect the smell of the specimens proposed in this research.

- To correctly discriminate between a different odour categories and between similar odours in the case of good or bad meat types.
- To use a fine learning recognition pattern and technique, such as ANN.
- To examine a practical transfer function.
- To add a portable system.
- To produce a good productivity system.
- To use array sensors to characterise the following factors:
    - Detect changes in meat smells.
    - Quick real-time recognition.
    - Give an accurate repeated resolution.
    - Reasonable performance.

1.5 The Significance of the Study

The study of this subject could inspire individuals to consider the possibility of controlling smells within display devises or computers. We could interact with static or dynamic images or several tools in any application or system. This subject could initiate future innovations. Here we are not referring only to this specific study but to all the studies in the same field.

The combination of an odour-sensing system and an ANN is a great challenge, and the multimedia component has many limitations. Furthermore, this study will add multimedia elements to the odour-sensing results to utilise the benefits that multimedia elements could add to any system. Moreover, it could achieve additional benefits for other studies within the same scenario as well as be considered a beneficial contribution to this area of study.

In addition, this proposed system could increase some properties in similar systems. The utilization could maximise the odour-sensing system and migrate it to an acceptable level of recognition for a mass of odours with high accuracy. In addition, it could increase performance that it achieves when the proposed prototype works as expected with high speed performance, minimum errors, reasonable numbers of detected smells, and correct results in a short period of time with less training time even for inexperienced users. Therefore, this research study is considered as an motivational start point for further future studies.

1.6 The Limitations of the Study

The study will focus on three main components, the odour-sensing system, the ANN, and expose question of adding the odour sensing as a new multimedia elements. The testing area will be for three types of meats (chicken, beef, and mutton), using the available odour sensor (OMX-GR sensor).

Second, the learning technique ANN, which can simulate the human olfactory system, will use backpropagation in this study. It will ensure accuracy and performance, and will use a definite parameter for supervised learning with backpropagation ANN and have a sigmoid activation function.

The third component is the multimedia elements, such as sounds and images, which can be integrated with the entire odour system in a particular sector, which can increase the benefits of the system (Persaud & Dodd, 1982) Figure (3).

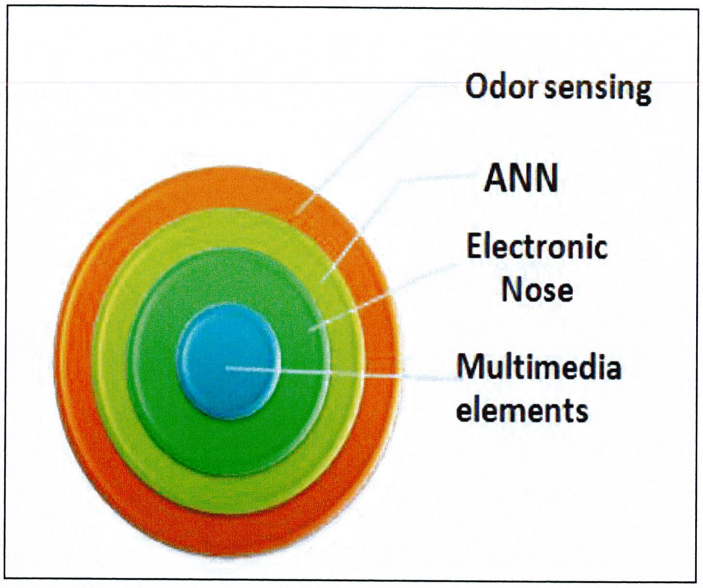


Figure (3)
The scope of the work

1.7 Organization of the Thesis

This thesis is organised into five chapters. Chapter 1 is an introductory chapter of the thesis in which the problem is defined and the main objectives of research are identified.

Chapter 2 presents the literature review, which contains some important studies similar to the subject and some varied methods.

Chapter 3 covers an outline of important information for e-nose and ANN.

Chapter 4 contains the instructions, includes the analysis and design of a prototype system, presents the electronic sensor that has been used, displays the data collection, preparations, and ANN software as well as the results of the proposed system.

Chapter 5 is the conclusion of the thesis including the discussion, conclusion, and future work recommendations.

1.8 Definition used in the Study

An ANN "is an information processing paradigm that was inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. Which have been used to analyse complex data and for pattern recognition, are showing promising results in chemical vapour recognition" (Karlik & Bastaki, 2004).

Odour sensing (e-nose) "is a device detecting odours which is designed to resemble the ability of the human nose, usually applied to the robot. The process of identification of the electronic nose will run into a problem when the gas which is detected has the same chemical element" (Helmy, Rivai, & Djoko, 2012).

Odour sensing (e-nose) "electronic-nose devices usually is performed on simple to complex mixtures of volatilized organic compounds derived from living tissues or from non-living processed products derived from plant or animal cells. The most common purpose of such analyses with e-nose instruments is to identify the source (plant, animal, or derived product) that produced the unique mixture of organic compounds present in the sample analyse, not the individual compounds present in the sample mixture. A second common purpose for performing e-nose analyses is to assess one or more chemical, biological or physical characteristics about the source that released the sample analyses" (Wilson, 2013).

Multimedia "is defined as the combining of different media types such as sound, animation, text, graphics and video for the presentation of information by making use of computers" by Bornman & Solms. (1993), (Güngören, 2013, p.14). According to Mayer. (2001), "multimedia is the presentation of materials in which text and images are used. Moreover,

Chapter 2: Literature Review

2.1 Studies of Odour sensing development.

2.2 Studies of odour-sensing with multimedia .

In this chapter, several studies are summarised that reviews similar researches and studies done on an e-nose. The chapter is divided into two parts. The first part briefly focuses on odour-sensing research and its development. The second part considers the combination of the odour-sensing system with multimedia elements.

2.1 Studies of Odour sensing development

Odour sensing, in some studies defined as e-nose, has been developed to try to attain certain goals, such as to obtain the best performance, real-time results, and precisely simulate the smell function of humans. Many studies have been done on odour sensing; most of them integrate with ANN to provide complex data processing.

Al-Bastaki (2009): proposed a prototype of a low cost odour-sensing system that can detect a large number of smells when compared with other industrial devices. It can discriminate between 20 different smells using a handheld odour meter (OMX-GR sensor) based on a neural network.

The main part of building an odour sensing system are the sensing system tools, which these tools contains a number of array sensors or a single sensor along with an automated pattern recognition system. As previously stated, each odour has its own signature; therefore, there could be a database of odours trained in the system via a recognition system. This study used several common household chemicals, and each chemical has its own designed sensor.

The role of a neural network is analysing and processing the received information with good detection and least selective sensors, which reduces the expense. The definition of an ANN is based on the study "An Artificial Neural Network (ANN)" and is an information

processing paradigm that was inspired by the way biological nervous systems, such as the brain, process information (Al-Bastaki, 2009).

An Artificial neural network works by neurons, which 1) receive inputs or raw information, 2) multiply the inputs' individual weights, 3) sum the weights of all the inputs, 4) and then transfer them through the transfer function. The function could be linear or sigmoid; in this study, the sigmoid function was used. The ANN has interconnected neurons as a network that is comprised of three layers. The first layer is an input layer, which has sensors for each neuron. The hidden layer is in the middle and does the classification (one layer with a variable number of neurons), and the output layer has one neuron for each input sample, which has a previous prediction.

In practice, this system can be trained to simulate real values. The parameters were chosen from a well-known training set to minimise errors. There are many benefits to using an ANN in the prototype, such as speed of performance, identification time equal to the response time, using propagation, and many array sensors can grant high real-time odour identification.

Backpropagation trained, Kohonen or Self-Organizing Maps (SOMs), Learning Vector Quantizers (LVQs), feed-forward networks, hamming networks, Boltzmann machines, and Hopfield networks are all types of ANN detections used in odour sensing. A multi-layered neural network with backpropagation delta rule learning is the method type of the ANN that is used in this prototype.

The proposed prototype trained with 20 different odours taken from common household perfumes, using a handheld odour meter (OMX-GR sensor), which can acquire data by real-time sampling and memory sampling data.

The system displays three forms for the user. The first one allows the user to choose between the real-time sampling option or memory sampling option, each choice will lead to another dialog box. If the user chooses the first option, it will ask the user to pick the desire file, and if the user chooses the second option, it will open the third form.

This process for training the system requires creating a database by asking the user to enter some values, such as a file name, then creating four new files containing the weights, values, and errors, and the output file generation containing the results of the testing session. The program will ask the user to enter the desired function as L for learning, O for obtaining the output, or 1 to resume from old weights. If L is chosen, then there should be 30 numbers entered referring to the odour features, followed by 20 output units (depending on the number of the case). After the system searches for the file and reads it, the user enters the momentum rate value 0.9, followed the learning rate 0.7, then enters the maximum number, then inputs the number of hidden layers and layers units for each layer. Finally, the program asks the user whether there is a need to create an error file before starting the learning session. When the learning session starts and the user selects the testing option, the system can work with multiple learning tasks, and then enter some values to complete the session.

The proposed odour-sensing system based on an ANN completes the process within seconds, sensors signals are digitised, and the ANN is implemented in software to identify chemical odours, and could be a great online odour recognition system in the future (Al-Bastaki, 2009).

Wilson (2013): tracked the growth of odour-sensing systems in the agriculture and forestry fields, which have recorded a huge improvement and increased profit in several ways. The e-nose could identify species of plants and animals, could check the correctness of a health test depending

on government regulations, could assure the healthy environment of fruit growth without any toxic substances, and could check the production of the woody plants in the forestry industry. In this specific industry, the e-nose could be used to detect organic and nonorganic materials to recognise a mixtures of gas. The electronic tongue sensor technique optimises the concentration of lipids on the membrane; the latest version of e-tongue uses multisensory arrays for liquid processing and recognition patterns including an ANN.

Electro conductive hydrogels are another instrument that could combine with the e-nose to detect chemical species, such as DNA fragments, enzymes, etc. Furthermore, it could detect plant diseases using Near Infrared (NIR) integrated with the e-nose. This tools also can improve gas sensor sensitivity through using an e-nose made of metal oxide nanowires and using of 1-D metal oxide for chemical sensing applications.

A recent e-nose application in this industry is using e-nose as a monitoring tool to detect any changes in Volatile organic compound VOC to assure the safety and value of the production of consumable plants. The great benefits offered from using the e-nose in agricultural and forestry industries is fast real-time recognition, accurate and repeated resolution, portability and inexpensiveness compared with complex analytical instruments. Still, the e-nose is a promising solution for many gas recognition problems in this industry (Wilson, 2013).

Chen ,et al. (2013): use of an odour-sensing system is in clinical diagnosis; a recent study concluded the high potential of the e-nose to detect early symptoms of diseases. The potential is due to the real-time detection, non-invasiveness, and accurate early diagnostic results, which could save many lives. The same study demonstrated many types of array sensors, which define the e-nose with different signal processing and

pattern-recognition systems depending on the purpose of the e-nose. Each sensor could generate a characteristic fingerprint when the interaction occurs between the VOC. The unique fingerprint for each odour could be known by the recognition system.

The e-nose is composed of three main systems. The first is a sample delivery system that will introduce odours to the detection system via filtering and improve the odour signal to give the most precise results. There are many techniques used to extract from VOC, and each technique depends on the type of sample used and the required sensitivity. Static Headspace Extraction (SHS) is the technique most used because it only requires minimal sample training and is inexpensive, simple, and rapid. Dynamic Headspace Extraction (DHS) and Solid-Phase Micro Extraction (SPME) are other techniques used.

The second system is a detection system, which is comprised of various sensor arrays from the first sensor array, which uses a Metal Oxide Semiconductor (MOS). Moreover, there are many sensors, such as Metal Oxide Semiconductor Field-Effect Transistors (MOSFET), Conducting Polymer sensors (CP), optical sensors, piezoelectric sensors, such as Quartz Crystal Microbalance (QCM), and Surface Acoustic Wave (SAW). Each sensor has positive and negative sides, and every researcher must try to optimise these sensors to reduce the error while increasing the accuracy of identifying the samples and must choose the suitable sensor for the proper purpose.

The last system is a data computing system, which begins when the sample is accepted and stored in the computer. Accordingly, this stage is divided into four stages as follows: 1) signal pre-processing is extracting descriptive parameters from the sensor array, 2) dimensionality reduction is to avoid any problems from the high dimensional database, 3) the prediction stage can classify unknown odour samples from the previously

learned database, and 4) finally, the validation is choosing the parameter model, which reduces the error via the previously trained model.

The unsupervised method of learning to discriminate between unknown odours without corresponding knowledge uses Principal Components Analysis (PCA). The supervised method of learning unknown odour classifications is done according to 87 preceding knowledge using Linear Discriminate Analysis (LDA). These are both types of pattern recognition methods.

Many applications of the e-nose are used in clinical diagnoses like cancer, respiratory diseases, such as asthma and tuberculosis, and urinary tract infection detection. All researchers agree that the high accuracy of early detection gives devices high potential in the clinical field; nevertheless, some challenges need to be resolved, for example, the temperature, humidity, sensitivity, and sample interference with other odours (Chen, Wang, & Choi, 2013).

Karlik (2004): found that in clinical diagnosis regarding diabetic patients the odour-sensing system could be used to detect different odours and classify them. In other words, it can examine the breath of diabetic patients and diagnose the disease.

The e-nose in this system used a handheld odour (OMX-GR) sensor, which can assist in obtaining the data, and the role of the ANN was to classify and obtain the state of the patients.

The ANN includes a Multi Layered Perception (MLP) trained system with the backpropagation technique, which will provide a real-time diagnosis when the identification time is equal to the response time. Further, the parallel process can increase the speed of the learning phase and the diagnosis time.

The system will transfer the information between computers or several areas, such as hospitals or medical clinics, helping to share the data, using a friendly interface system, which is implemented via a Visual Basic program that can provide portability (Karlik. B, 2004).

Srinonchat, J. (2013): Besides the previous study, another study used the e-nose to monitor air pollution and alarms for the industrial sector, based on four sensors (TGS2620, TGS2620, TGS2442, and TGS832). They are considered VOC sensors that are comprised of metal oxide SnO_2, which allows the oxygen molecular creates electronic that block the flow of electrons and cause high resistance. The resistance and flow current assist in converting them into digital data, and then transfer it to data acquisition. The results show that these sensors can classify and be sensitive to different gases, such as methanol, n-propanol, hexane, and dichloromethane (Srinonchat, 2013).

Macías and .et al (2013): In addition, regarding the uses of the e-nose in aroma detection, one study used a type of e-nose to detect odour with lower cost based on an mbed microcontroller, which is defined as series of ARM microcontrollers to develop any smart device with the aim of using the microcontroller to allow portability, simple, fast control of the device, and high performance as well as the ability to use it without expert knowledge. The microcontroller can include Ethernet connections, many interfaces, and flash memory, and it is a dual in-line package with a built-in USB flash programmer; therefore, it will not need updating, and the external programmer cannot remove the mbed. Moreover, the programming concept of the mbed is based on cloud computing, which allows hosting via the mbed.org developer website to reduce overhead. The proposed e-nose is comprised of mbed microcontrollers, four Figaro gas sensors for the sensor chamber, which have load resistors, a temperature and humidity sensor, two small pumps, and electro valves.

The measurement techniques used in the proposed e-nose include injection, which mixes the sample with fresh air, and cleaning to avoid interference with the next sample. The e-nose procedure operates in two ways: the measurement mode and analysis mode. The first mode uses a file on a computer to train the neural network, and the second method uses a trained neural network to take new measures. The testing phase faces some problems like reproducibility and repeatability.When conducting the testing on some wine samples, in three synthetic samples of aqueous ethanol solutions at 10%, 12%, 14% V/V with twenty more reads, a total of 60 prototypes were classified in the three groups. Hidden layer perceptrons and support vector machines (SVM) are used to solve the classification problem. Accurate classification used to compare both SVM and ANN, SVM which only can take more than three hours but ANN can take up to four days to generate the results. Generally the ANN gets better results at 100% for alc 10 and alc 14, with 99.5% for the alc 12 class, (Macías, Agudo, & Manso, 2013).

Emmanuelle and .et al (1998): a food application was summarised in a review article that used an e-nose system comprised of sensors, air conditioning, electronic pumps, and much more. The main characteristics of this system were to work a long time with good repeatability defined as the ability to obtain the same sample on the same array within a short time period. Reproducibility defines the ability of the sensors to produce the same results for the same sample. This reviewed all developed sensors and showed the advantages and disadvantages for each one. For instance, the Metal Oxide Semiconductor sensor (MOS) relies on conductivity changes and the film technique depends on the size. The thin film has a faster response and higher sensitivity with difficulty in reproducibility. In contrast, the thick film Conducts Organic Polymer sensors (CP), is based on resistance changes, has a more complex

operation, is not widely used because of the possibility of creating a fine batch-to-batch reproducibility, and has a high expense in manufacturing. Piezoelectric crystal sensors, which are based on changing the mass (measures the resonance frequency change), can be either Bulk Acoustic Wave (BAW) and Surface Acoustic Wave (SAW) sensors. The SAW sensors are more sensitive than the BAW sensors and the difficulty to integrate both sensors can cause complex electronics, which causes more sensitivity. Metal Oxide Semiconductor Field-Effect Transistor sensors (MOSFET) similar to MOS sensors have low sensitivity to moisture and require competent manufacturing to obtain great quality. The e-nose uses three main statistical analysis techniques, namely graphical analyses, which are the simplest, multivariate analyses, and network analyses, which contains the Radial Basis Function (RBF) and an ANN, which is considered a great tool to classify unknown samples from preceding references. Many researchers and experiments test many types of foods using a special e-nose applications, for example, meat, grains, coffee, beer, mushrooms, cheese, sugar, blueberries, and beverages, such as orange juice and cola, with many kinds of sensors, almost all results are positive and obtain great recognition samples. The conclusion is that e-nose or odour-sensing systems could not yet be used as a complete replacement of the human nose function. Further, the huge difficulty to combine types of sensors into one instrument will produce an extremely complex device; therefore, the best solution for now is to create an e-nose device for one purpose and for certain applications (Schaller, Bosset, & Escher, 1998).

Degenhardt, and .et al (2012): Addition, one study examined the possibility of using e-nose technology to detect the stink bug from a damaged cotton bolls and to discriminate between VOC from different species of stink bugs. The proposed e-nose includes 32 sensors. The

resistance for each sensor measures and represents a smell print, then collects the signals and analyses them. The training of the database before determination is necessary, and then checking the accuracy of the training database is necessary. The prototype recorded great results when analysing deferential damage and an undamaged cotton boll at 100%; however, differentiating between types of stink bugs was a weakness in this study. The author suggested to resolve this problem by optimising the sensors. The e-nose technology has a promising future in many ways as well as in detecting the family of stink bugs feeding on cotton bolls (Degenhardt, Greene, & Khalilian, 2012).

2.2 Studies of odour-sensing with multimedia

All above studies have attempted to discover the perfect odour-sensing device that can mimic the exact function of the human nose; moreover, in this study, we focus on the possibility of examining odour-sensing devices with multimedia components. There has been several studies on the topic (four or five researchers have considered the same matter, depending on our search methods).

Castro, and .et al (2003): One of the interesting studies done by a group of qualified doctors focused on an odour-sensing system using multimedia as a primary element for lower cost conditions. The study presented some statistics and odour-sensing history. The first smell detection instrument was created in 1961 for agricultural purposes. In the late 1980s, the first intelligent e-nose was invented. There are a number of ways to classify smell into different categories. The Linnaeus classification in 1752 used seven groups based on their appeal. Zwaardemakers continued the work of Linnaeus in 1895 to conclude there are nine groups. In 1975, Lovell proposed eight groups based on the

smell source. The best standard classification is immature, using multimedia applications, the discrimination of smell is between 50-200 groups.

Smell sensors are developed over the life of the odour-sensing system and are considered the heart of odour sensing. To the authors' knowledge, there has been no development of smell sensors targeting multimedia applications. The materials used to build any type of sensor consist of metals, semiconductors, ionic compounds, and organic compounds. The Metal Oxide Semiconductor (MOS) is the most used sensor due to its lower cost, high productivity, and low power consumption. Most systems are designed to detect ranges of smells using an array of microsensors.

Next is a brief review of important facts necessary to produce an efficient odour-sensing system for a multimedia application. The proposed system aims to identify and classify well-known smells in food, beverages, plants, perfumes, and essential oils. The proposed e-nose comprises four main parts: delivery system, sensor chamber, sensor electronics, and a data processing module.

The smell delivery system brings the smell into the sensor chamber manually or automatically, which confirms the humidity, temperature, volume instability, and speed of the sample. To ensure repeatability, there is a small suction fan pumping clean air to remove any trace of the previous odour.

The sensor array used is the Figaro sensor because of the low cost and sensitivity to volatile vapours from many organic compounds found in nature and non-natural odours, such as cigarette smoke, diesel, and gasoline. The detection of the above odours could be a good background to enrich the multimedia application. The types of sensors used were the TGS 826, TGS 825, TGS 822, TGS 880, TGS 883T, TGS 2602, TGS

2610, TGS 2201; each one can determine a type of smell. The proposed sensor electronics measure the change of the conductance of the MOS sensors via changes in the resistance caused by the constant voltage, which heats the circuit and causes detection of a wide range of smells.

The data processing model used the neural network paradigm that offers a high accuracy in classifying smells, which is implemented directly in the hardware to increase data processing. The output presents a look-up table number with a range from 0 to 100.

The main factors for a good e-nose with low cost are the detection time, power consumption, and portability. For multimedia aims, the sensors must have high detection of many natural and non-natural odours and be able to classify them into groups. There is no need to highly discriminate between similar odours, but unlike odours can be easily distinguished.

Furthermore, the response of the odour-sensing system generally takes a long time (between two to ten minutes) depending on the type of the sensor and factors, such as the temperature and humidity. For example, a MOS sensor can take 30 to 120 seconds to detect a smell and three minutes for the cleaning period before it can receive the next sample.

Some factors that consume the power of the device are the automated delivery system pump, opening and closing electromechanical valves, which takes 50% of the power consumption, and finally high temperature operation.

The portability of the odour-sensing system is required and depends on the size of the sensor. However, the smaller device used increases the cost; this technology is not yet mature. The low cost of the system was obtained by selecting eight off-the-shelf sensors.

The e-nose in this study attempts to create an odour-sensing system that works along with multimedia applications, which could be used in any field where it is necessary to identify a wide range of smells.

The proposed prototype for the odour-sensing system with multimedia is difficult because such systems are not yet mature, as the olfactory function in the human olfactory system is not completely understood yet. There are almost 10,000 detectable odours, which increase the complexity of the system when compared to the vision system.

The proposed odour detection system is comprised of a small DC motor-driven suction fan to create an air current inside the sensor chamber and create signals in the off state of the fan to stabilise the sensors. The array sensor used is made of a semiconductor and metal oxide based on sensors from Figaro. Consequently, they are sensitive to the volatile organic vapours from various substances, such as food, toxic gases, and organic solvents. Moreover, each sensor could recognise a large number of odours; however, the temperature and humidity are not included in the design; therefore, it will reduce the response time, especially as the design proposed is to be portable and low cost. The output is a result of the sensors module process, which depends on a lookup table that represents the output in two forms: it first displays the odour name in text and second in an electronic code representing the odour sample in line with the transmission through the network.

The researchers wish to continue developing their proposed design and to have the system detect a combination of smells using the ANN or even a fuzzy logic algorithm and to detect a wider range of odours, while reducing the size of the system, which will have a robust multimedia interface (Castro, Mandal, Ajemba, & Istihad, 2003).

Al-Bastaki (2011): in a very remarkable study the authors proposed an odour-sensing device that can classify 20 odours using a handheld odour meter (OMG-GR sensor) based on a backpropagation neural network, which can be used in any multimedia application. In this research, they used the system to improve online shopping in the perfume e-market. As previously mentioned, there are few applications integrated with odour-sensing devices as a result of the complexity of designing a simulation of the olfactory system that can identify a wide range of smells. On the other hand, there are many advantages to developing an odour-sensing device, such as to reduce the cost of production in various industries and to prevent any risks from many gases and substances. The ANN is considered an incredible information-processing paradigm to train the system to learn to identify odours and chemicals in parallel processing to increase speed performance by engaging repetitive processes from previously trained data using a training algorithm. When the identification time is the same as the response time, odour identification can be conducted in real time.

Most odour-sensing systems that could be considered an e-nose consist of two main elements: the sensing system and the pattern recognition system. The sensing system could consist of an array sensor, where each sensor identifies a specific property of a chemical substance of an odour or could use one sensor to create an array of interpretations for each odour, or it could use both.

The sensor produces a unique signature for each odour, which will create a database labelled with the properties for the specific odours to train the recognition pattern in order to complete the automated identification. In their prototype, they used array sensors and the backpropagation neural network for classification and recognition to create a multilayer feed-forward network trained with the

backpropagation algorithm, which is integrated with online shopping systems to recognise odours from common household odorants. The system uses real-time sampling data or memory sampling data to obtain data from the sensor. Three forms are contained in the system, and the ANN tests the data and classifies them. Moreover, the system asks the users to enter information into the learning session state, and they can obtain the output. The authors suggested that the limitation and the future work were to improve the identification of the odours and cover larger samples of them, respectively, which requires a great number of sensor elements. The proposed system supports an online recognition system, which could be used in the future for a regeneration system. In addition, this will improve the response time, which is a vital property of multimedia applications. Furthermore, the study intended to examine the usability in integration of the olfactory data and the multisensory interfaces in some applications. The study did not examine the e-market only; it could reach beyond many other sectors needed. The most important factors in building a multimedia system are to respect security and safety (Al-Bastaki, 2011).

To summarise from the previous studies, the odour-sensing system continues to evolve and requires improvement to the system. Generally, authors have agreed that to produce an efficient system several factors are required, which include the following:

- Detection of a large number of odours.
- Speedy performance.
- Real-time smell identification.
- Using array sensors grants very accurate results and high real-time odour identification. Many common sensors, such as metal oxide semiconductor (MOS), feature speed, high responsiveness, productivity, low cost, low power consumption, and high

sensitivity. In addition, Figaro sensors are widely used because of their low cost, sensitivity to volatile vapours from many VOCs in nature and non-natural odours.

- Low cost device.
- An efficient learning technique that supports the final product, such as backpropagation.
- Portability.
- Fine discrimination between similar odours.
- The odour-sensing device could be used in many different sectors; therefore, selecting the type of sensors depends on the functionality of the system. Such systems could include the following:
    - food industry,
    - packaging material,
    - medical diagnostics,
    - environmental monitoring,
    - perfume and aroma industry,
    - control of beverages (e.g., wine and beer),
    - assessment of car interiors,
    - biology science (identify the species of plants and animals), and agriculture and forestry.

- To create a device that simulates the human olfactory system, different sensors must be combined. Each has its own characteristics and recognition criteria to identify a wide range of smells; however, this scenario is difficult with complex issues and a high cost to implement. Therefore, the best technique at the current time is to produce a system with specific regions or specialties.
- The goals in most studies are to create a professional device to replace the human smell functionality with a robust the multimedia interface.

- Despite the major role of using multimedia elements integrated with the odour-sensing system in many industries, this study could be a good contribution in this specific era.

Chapter 3: Electronic Nose and ANN

3.1. Electronic Nose.

3.1.1. The Olfactory System.

3.1.2. Smell Classification.

3.1.3. Smell Sensors.

3.1.3.1 semiconductors sensors.

3.1.3.2 Figaro sensors.

3.1.4 Electronics Sensors.

3.1.5 Data Processing Module.

3.2. Artificial Neural Network.

3.2.1 Neural Network Architecture.

3.2.2 Type of Learning.

3.2.3 Activation Function.

3.2.4 Backpropagation.

3.2.4.1 Other Major Components of the Artificial Neural Network.

3.2.5 Training Modes.

3.2.6 Least Mean Sum of Squares Error Function (SSE).

3.2.7 The Advantages of Using ANN.

3.1 Electronic Nose

The electronic nose (e-nose) or artificial nose has been continuously developed through several decades for detecting and classifying odours or gases. The following two major components comprise the e-nose:

1. The sensing system: It is a sensor, which could be a single sensing element or an array of several sensing elements; each element will measure a specific odour attribute. Moreover, the device can be a combination of both elements.

2. The automated pattern recognition system: This component of the e-nose will organise and recognise each odour, and assign each odour its own recognition key. Further, each odour has a different chemical mixture. Even when there are two odours that have the same chemical substance, the saturation will be different. Because each odour is represented as a fingerprint, which is differentiated from other odours, it can automate a special pattern for each precise odour (Al-Bastaki, 2009).

During the training process, when the odour is presented to the sensing system, the ANN has the biggest role to recognise and analyse the broad range of different odours. The ANN will be discussed in Section 3.2.4.

3.1.1 The Olfactory System

To understand how the e-nose works, the biological olfactory system process must be understood. When any odour or smell is presented to the sensing element in the human nose Fig3.1, which consists of many receptors or chemical sensors to produce signals, the signals or odour information will start processing in both the olfactory bulb and cortex of the brain. The olfactory cortex is responsible for the

recognition and classification of odours. When it identifies the odour, the information will transfer to the hippocampus, limbic system, and the cerebral cortex. In addition, the cerebral cortex is connected with the conscious to determine the odour and control the reaction to the specific odour. In addition, the hippocampus is responsible for evoking memories. Many times when smelling a certain food we can remember a specific place or a sensation that happened during childhood (for example), and some smells can evoke happy or sad memories (Al-Bastaki, 2009) Figure (4).

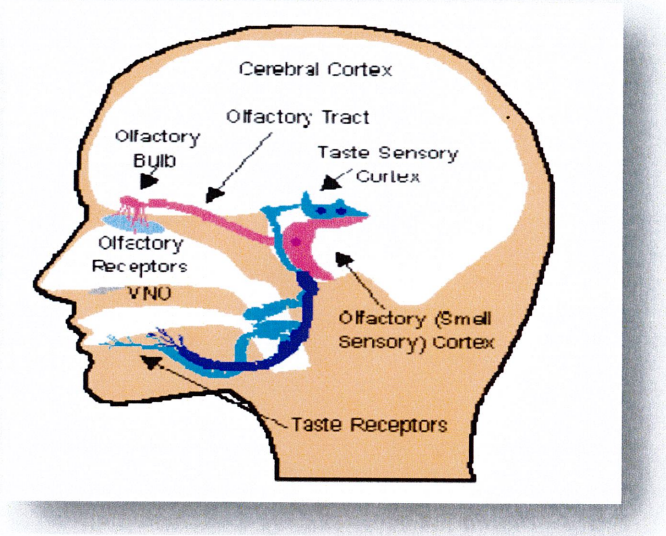


Figure (4)
The process of the olfactory system (Al-Bastaki, 2009)

3.1.2 Smell Classification

In this section, we will discuss some of the famous classifications of smells that have been proposed by several scientists. The classification of smells will have different perceptions.

In 1752, the first smell classification was proposed by Linnaeus. He classified smells into seven groups, as shown in Table (1) based on their demand.

Table (1)

Linnaean and Zwaaredmaker Smell Classifications

(Castro, Mandal, Ajemba, & Istihad, 2003)

LINNAEAN AND ZWAARDEMAKER CLASSES	EXAMPLES
Aromatic	Camphor, spices, anise, citron, almond
Fragrant	Flowers, vanilla, balsam
Ambrosiac	Musk, amber
Alliaceous	Onion, garlic, acetylene, iodine
Hircine	Goaty smells, cheese, sweat, chestnuts
Foul	Narcotics, and some bugs
Nauseous	Carrion, carrion flower, faeces
Ethereal	The fruity ethers of perfumes, beeswax, ether
Empyreumatic	Roasted coffee, tobacco smoke, xylol, naphthalene

In 1895, Zwaardemaker's classification was extended from Linnaeus's classification by adding two more groups shown in Table (1). In 1923, the Lovell classification was based on the smell source and contained eight groups; it satisfied several services, such as agriculturalists, consumers, and naturalists, Table (2).

Table (2)

LOVELL Smell Classifications

(Castro, Mandal, Ajemba, & Istihad, 2003)

LOVELL CLASSES	EXAMPLES
Sweet Flowers	Honey, apricot
Fruit	Apple, pear
Aromatic or Spicy	Cloves, ginger
Musk	Musk mallow
Onion and Alliaceous	Onion, garlic
Rank	Goat smell
Foul	Bugs, opium.
Nauseous	Urine, putrid fish

There are some conflict regarding the best and accurate smells classification, also the developed wheel will continue to find the best classification of smells (Castro, Mandal, Ajemba, & Istihad, 2003).

3.1.3. Smell Sensors

The smell sensors are continuously developed to detect odours as accurately as possible. Smell sensors are generally built based on the recognition smell requirements. In other words, sensor properties and features are based on the purposes of the detection. The manufacture of sensors includes many materials, such as semiconductors, metals, ionic compounds, and organic compounds, such as enzymes and polymers. Table (3) shows in detail many materials that are used to create a variety of types of smell sensors (Castro, Mandal, Ajemba, & Istihad, 2003).

Table (3)

Typical Material Used as Sensors

(Castro, Mandal, Ajemba, & Istihad, 2003)

TYPE OF MATERIAL	EXAMPLES
Metals	Pt, Pd, Ni, Ag, Au, In, Ga, Rh, Sb
Semiconductors	Si, GaAs, InP, $NiFe_2O_4$
Electronic conductors	SnO_2, TiO_2, Ta_2O_5
Mixed conductors	Ga_2O_3, WO_3, IrO_x
Ionic conductors	ZrO_2, LaF_3, CeO_2, CaF_2, Na_2CO_3
Crystals	PbPc, $LuPc_2$, $(PcAlF)_n$
Polymers	Phthalocyanines, polydiacetylenes

Each material is used for a specific detecting class or group of odours. Some sensors will be briefly illustrated in following subtitles.

3.1.3.1 semiconductors sensors:

For example, P-type semiconductors are used for oxidising gases, and semiconductors and metal sensors are used for the applications of gas sensors. Moreover, some semiconductors can detect methane and hydrogen. The semiconductor sensors has a very effective feature in gas molecules arrangement of the surface of these sensors, mesostructure, heterostructure, and Schottky diode. The metal-oxide semiconductor (MOS) is a low cost sensor; therefore, it is more popular.

3.1.3.2 Figaro sensors:

Figaro sensors also are low cost. This sensor will detect various kinds of gases, such as odourless toxic gases, combustible gases, and organic compounds, such as solvents, foods, alcohols, and a number of toxic gases in nature as well as non-natural gases like gasoline and cigarette smoke. Figaro sensors include large numbers of this type like TGS 826 and TGS 825 sensors, which sense sulphurous smells. The TGS 822 will react with alcohols and wood fermentation. The TGS 2602 recognises VOCs, and contains a wide range of smells that were introduced in the sensors mentioned previously. The TGS 2610 is sensitive to hydrocarbon substances, such as butanol and propanol. The TGS 880 and TGS 883T both are sensitive to smells of cooking food produced from water vapour. The TGS 2201 is affected by the smell of the street and bars, such as toxic gases from cigarette smoke and gasoline. Table (4) displays the Figaro sensors (Castro, Mandal, Ajemba, & Istihad, 2003).

Table (4)

List of the Sensors Proposed

(Castro, Mandal, Ajemba, & Istihad, 2003)

SENSOR	TARGETED GASES	SMELLS LIKE
TGS 826	Ammonia (NH3)	Old rotten urine and fish (sharp, penetrating, and irritating smell)
TGS 825	Hydrogen sulfide (H2S)	Rotten-egg, sulfurous smells
TGS 822	Alcohol, Xylene, Toluene, other volatile org. vapors	Aromatic solvents, wood fermentation, alcoholic beverages
TGS 880	Volatile vapors from food	Food (while cooking)
TGS 883T	Water vapors from food	Soups, coffee
TGS 2602	Responsive to a wide range of gases: ammonia, hydrogen sulfide, ethanol, toluene	A wide range of smell: from rotten fish to wood fermentattion
TGS 2610	General hydrocarbons	Smell of ripening fruits
TGS 2201	Gasoline and diesel exhaust	Irritating and suffocating smells

3.1.4 Electronics Sensors

The way an electronic sensor (e-nose) works is that it will measure the change of the values of different parameters in each smell exposed to it. In addition, sensors will respond to different properties depending on the nature of the sensor selected, such as the conductivity and capacitance. The electricity voltage is split between standard resistance and the changing resistance (Castro, Mandal, Ajemba, & Istihad, 2003).

3.1.5 Data Processing Module

To discriminate between a wide range of smells, there are some techniques used to process the smells, such as the statistical pattern analysis technique, for instance, the discriminate function analysis, principal component analysis, and canonical correlation. The most successful technique used is the ANN with backpropagation that will be illustrated in the next section. Other types of ANNs include the radial basis function and Kohonen self-organizing map. Adaptive techniques are also used, such as genetic algorithms and adaptive resonance theory (Castro, Mandal, Ajemba, & Istihad, 2003).

3.2. Artificial Neural Network

The first model of an ANN was built by Warren S. McCulloch and Walter Pitts in 1943. They constructed a simple model, which had one cell that received inputs and produced an output (Shiffman, 2012).

The ANN is defined as a paradigm that simulates the biological human brain functions, such as thinking, learning, remembering, and more. Cells in the brain are considered the main element that generates the capability to process the incoming information that is received from human senses in order to have appropriate actions and output. These cells are called neurons, which construct the neural network, as interconnected neurons shape the network by working together to solve problems (Djarfour, Aifa, Baddari, Mihobi, & Farahtia, 2008).

Each neuron connects to the other neurons by a connection link that is related to a weight; the weight represents the strength of the value or the data used to activate the function to calculate the output for next layer (Djarfour, Aifa, Baddari, Mihobi, & Farahtia, 2008) Fig 3.2.

The neural network comprises many neurons that are interconnected with each other, formed in the following three layers:

- The input layer receives the raw data and transmits to the next layer.
- The hidden layer facilitates the entire process, extracts the data, and transfers it to the output layer.
- The output layer will produce the results (Hussain, 2009) Fig3.3.

For further illustration, in the Fausett example there are three inputs x_1, x_2, and x_3 (input layer) received to one neuron Y (hidden layer), and an associated link between values with weights w_1, w_2, and w_3. The activation is represented by x_1, x_2, and x_3. The mathematical way to solve the problem will first find the output of Y (Y_in) through the sum the product of the total weight and the input activation as shown in Eq. 1 and Figure (5) (Fausett, 1994).

$$Y\_in = w_1x_1 + w_2x_2 + w_3x_3 \qquad (1)$$

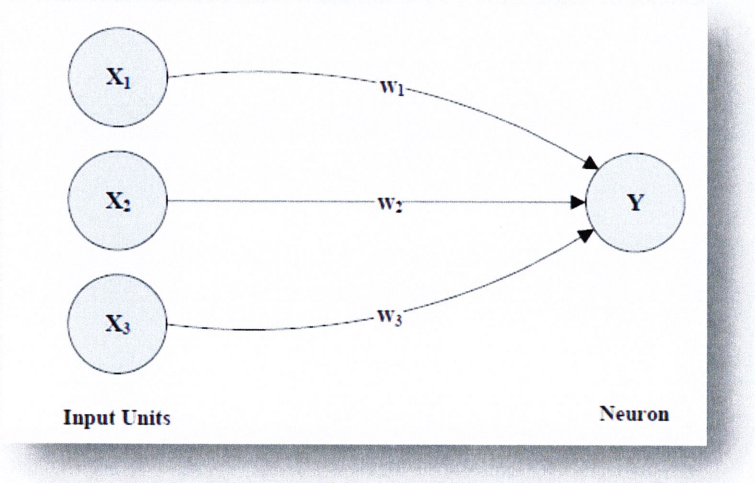


Figure (5)

The Scope of the Work (Fausett, 1994)

The second step is to use an activation function, which will be illustrated in Section 3.2.3. In the simple Fausett model of an ANN, the Y

sends the output after the process between the input layer and hidden layer, which considers the input to the next layer: Z_1 and Z_2 (output layer), which is comprised of two neurons with the related weights v_1 and v_2. (Fausett, 1994) Figure (6).

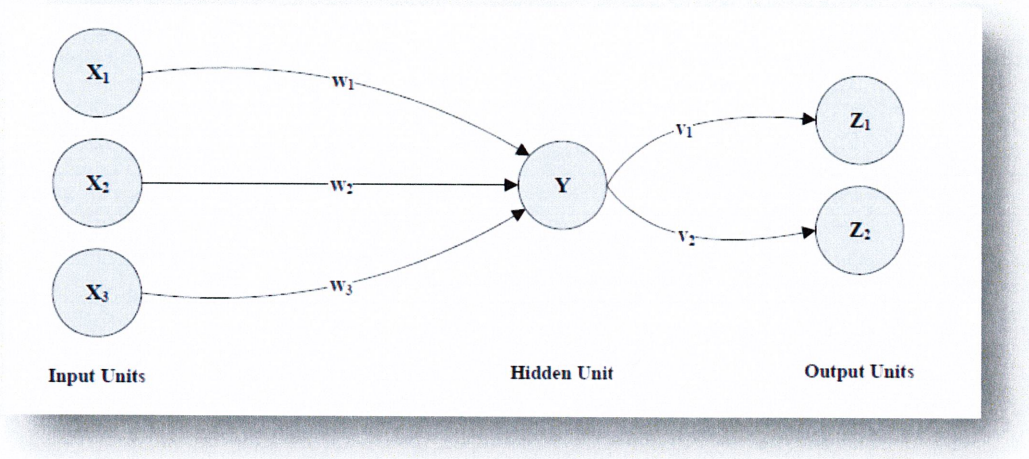


Figure (6)

Simple ANN (Fausett, 1994)

The problem that must be solved to control the nature of the ANN is the number of neurons in each layer and the number of layers, the initial weight with learning rate and momentum, and the architecture of the ANN. These subjects will be illustrated briefly in the next sections.

3.2.1 Neural Network Architecture

The way that neurons arrange (nerve cells) and exchange the information represented by electronic pulses called action potentials, is reflected in the arrangement of the three layers: the input layer, hidden layer, and output layer in the network and the number of the neurons in each layer. This is called the neural network architecture (Smith, 1997).

In 1999, Haykin divided network architectures into three groups as follows:

1. Single-layer perceptrons (feedforward networks): The single layer of perceptrons is the first and simplest class of neural network; it contains two layers in its feedforward network:

   - First layer (input layer) has stick function and connections weight.
   - Second layer (output layer): has one layer of the connection weight directly from input to output layers, for this reason it is indicated as the simplest class.

Moreover, it is a supervised learning model Figure (7) (Fausett, 1994; He, 1999).

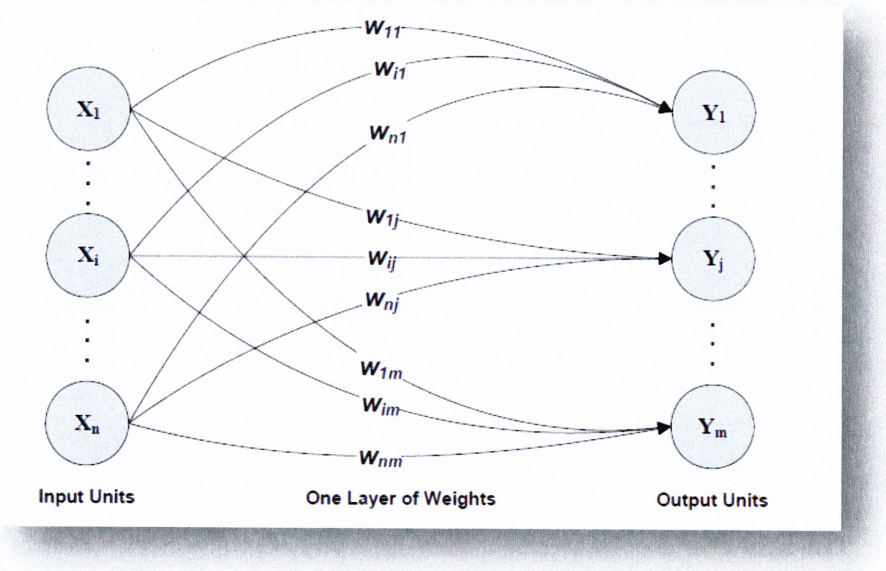


Figure (7)

Single-layer perceptrons (feedforward networks) (Fausett, 1994)

2. Multi-layer perceptrons (feedforward networks): The multi-layer comprises three layers: input layers, a hidden layer, which is linked to the input and the output layer. There can be more than one hidden layer. The

hidden and output layers do the computational processes. The characteristics of this class are as follows:

- This model uses a nonlinear activation function.
- Contains one or more hidden layer.
- Includes a high connectivity, through the weight (synapses) of the network.

This class of network always uses supervised learning. The error backpropagation algorithm is the more popular type used, which will be exhibited later (He, 1999) Figure (8).

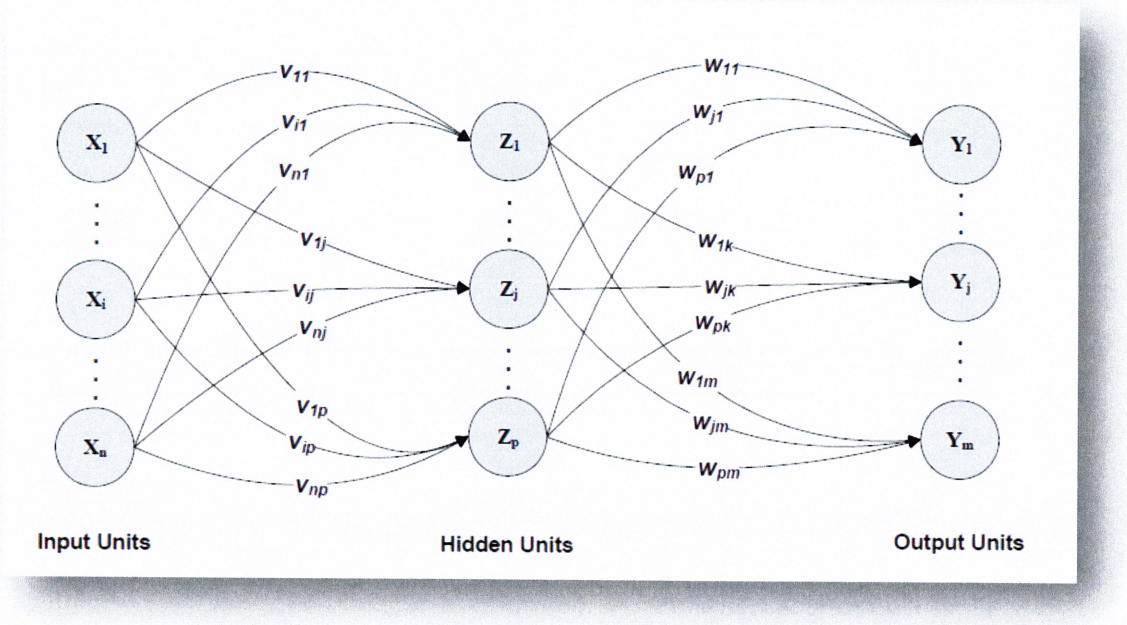


Figure (8)

A Multilayer Neural Network (Fausett, 1994)

3. Recurrent networks: Recurrent networks have one or more feedback loops that are used in a stable system required to use unsupervised learning, for example. The feedback behaviour included in this model is as follows:

- Local feedback in the single neuron.
- Global feedback for the entire network.

3.2.2 Type of Learning

To select the perfect type of learning, it will depend on the complexity of the problem. This section will display the types of learning.

Supervised learning has a supervisor or a teacher. For example, in our subject, the teacher will expose a number of odours to the network and will know the answer; the network processes this and provides a list of a guesses. Afterward, the teacher will provide the correct answers, and the network will compare between the desired answers and its guesses to correct the errors. This learning method has a serial of training with adjusted weights (Shiffman, 2012).

Unsupervised learning is used in a set of data without knowing the actual answer. Clustering is the method of learning. It will group the similar sets of data into groups without any training set or desired goal (Shiffman, 2012).

Reinforcement learning is based on observation. After a time of learning, the network will be able to make the right decision. For instance, when the network needs to take an action to choose between two options, one of them it will get a negative inspection and the other a positive inspection, and it adjusts the weights (fixes the error) to obtain the right answer the next time (Shiffman, 2012).

3.2.3 Activation Function

The activation function is a mathematical method to transform the input set to the output. There are many types of functions that will be presented, such as the linear function, piecewise linear function, threshold function, tangent hyperbolic function, and sigmoid function (Chakraborty, 2010) Figure (9).

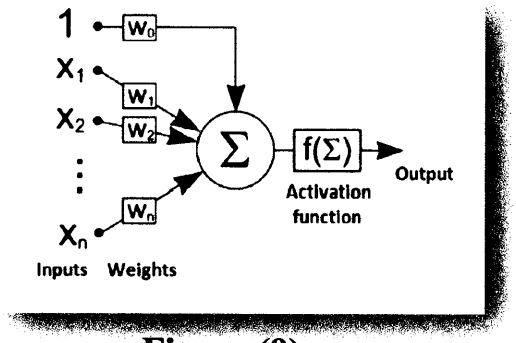


Figure (9)

Activation Function (Sibi, Jones, & Siddharth, 2013)

1. Identity function:

It will return the same values used. The mathematical equation used Eq. 2; Figure (10):

$$f(x)=x, \text{ for all } x \quad (2)$$

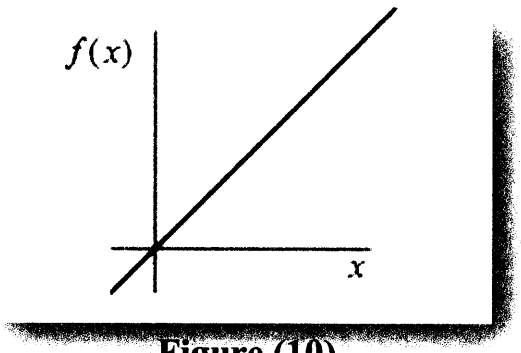


Figure (10)

Identity Function (Fausett, 1994)

2. Threshold function:

Also called binary function or Heaviside function, their job is to transfer the input continuously to the output in two manners:

1.Binary threshold:

In this case will generate a binary output:

1. When the sum of all the weight it is positive, it will be one.

2. When the sum of all the weight it is negative, it will be zero Eq. 3; Figure (11).

$$Y = f(I) = \begin{cases} 1 \text{ if } I \geq 0 \\ 0 \text{ if } I < 0 \end{cases} \qquad (3)$$

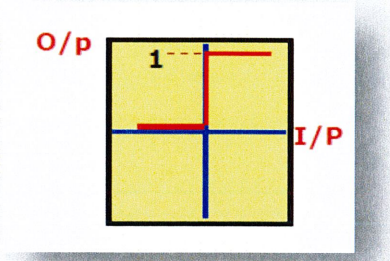


Figure (11)

Binary threshold (Chakraborty, 2010)

Bipolar threshold: In this case, it will generate an output as follows:

- When the sum of all the weight is positive, it will be one.
- When the sum of all the weight is negative, it will be -1 ; Eq. 4; Figure (12).

$$Y = f(I) = \begin{cases} 1 \text{ if } I \geq 0 \\ -1 \text{ if } I < 0 \end{cases} \qquad (4)$$

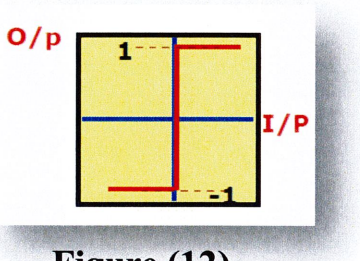


Figure (12)

Bipolar Threshold (Chakraborty, 2010)

Piecewise linear function or saturating linear function: It can have either a binary or bipolar output range from 0 to 1, and for the symmetric function, the output range is from -1 to 1 , Figure (13), (Sibi, Jones, & Siddharth, 2013).

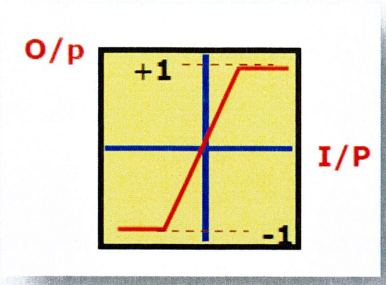


Figure (13)

Piecewise linear (Chakraborty, 2010)

Sigmoid function is defined as nonlinear with an S shape curved function, the most used of these functions are the logistics and hyperbolic tangent functions. These functions are used due to the decrease in the computational load of the training phase in the bond between the value and their derivative at a certain point (Chakraborty, 2010; Fausett, 1994); the two classifications are shown below.

Binary sigmoid function or logistic function: When the output range is between 0 and 1 and the steepness parameter σ=1 and σ=3 ; Figure (14), Eq. (5).

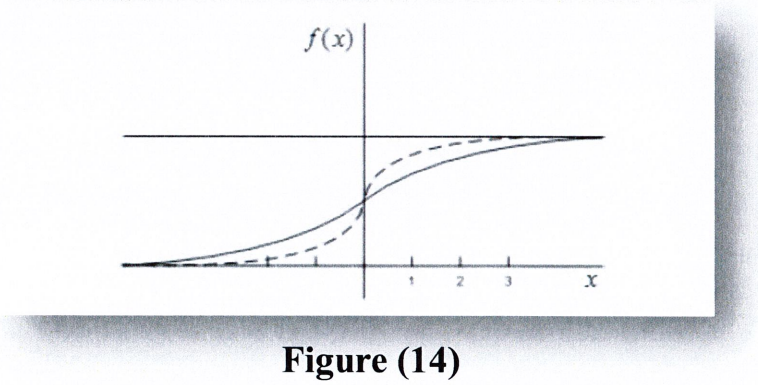


Figure (14)

Binary sigmoid function (Fausett, 1994)

$$f(x) = \frac{1}{1+\exp(-\sigma x)} \qquad (5)$$

Bipolar sigmoid function: When the output range is between 1 and -1, and the parameter σ=1 ; Figure (15), Eq. 6.

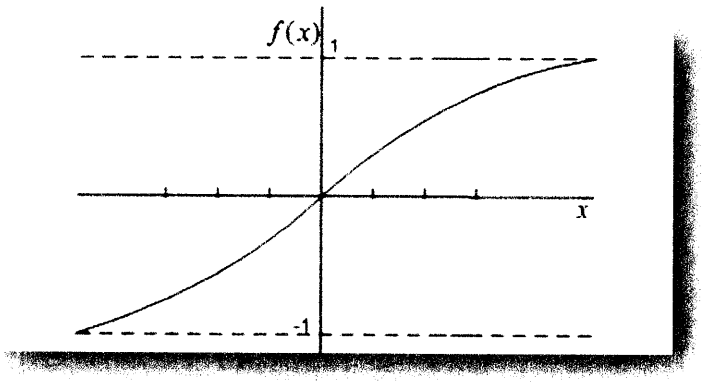


Figure (15)

bipolar sigmoid function (Fausett, 1994)

$$g(x) = 2f(x) - 1 = \frac{2}{1+\exp(-\sigma x)} - 1 \tag{6}$$

$$g(x) = \frac{1 - \exp(-\sigma x)}{1 + \exp(-\sigma x)}$$

In this research the Binary sigmoid function or logistic function will be used in the proposed system.

3.2.4 Backpropagation

Backpropagation is used to solve complex problems, when the XOR (Exclusive OR) has limitations. Backpropagation has two main principles: hidden layers and a gradient. It will require learning and training of multilayer perceptrons (MLP) (Haykin, 1994).

The training of the backpropagation is classified into three levels depending on the following (Fausett, 1994):

- Feedforward of the input training pattern:

The feedforward stage starts with the input layer in which the input receives input signals and transfers them to the hidden layer, where it will compute the activation function. Then, it will send the results of the activation function to the output layer, which will compute the activation to generate the output signal to the outside world (Fausett, 1994).

- Calculation and backpropagation of error:

In this stage, the error will calculate through taking the output signal that comes from the output layer and comparing it to the desire output to resolve the error. The amount of error will control the δ factor, which is considered a key element to propagate the error backward from the output layer to hidden layer. The weight updates between the output layer and hidden layer, after the hidden layer computes the error factor δ. The error factor is used only to update the weight between the hidden layer and input layer but will not propagate the error to the input layer.

- Adjustment of weights:

After stage two, when all the error factors are computed, in the third and final stage, the weights are adjusted from the hidden to the output layer and from the input to the hidden layer. The error factor and the activation of hidden and input layers controls the adjustment of all the weight processes (Fausett, 1994).

For example, the backpropagation work model is shown in Figure (16), where the input units (*X*), hidden units (*Z*), and output units (*Y*) are illustrated. The bias represents W_{0k} and V_{0j}, which are valued at 1 for the unit Y_k and Z_j (Fausett, 1994).

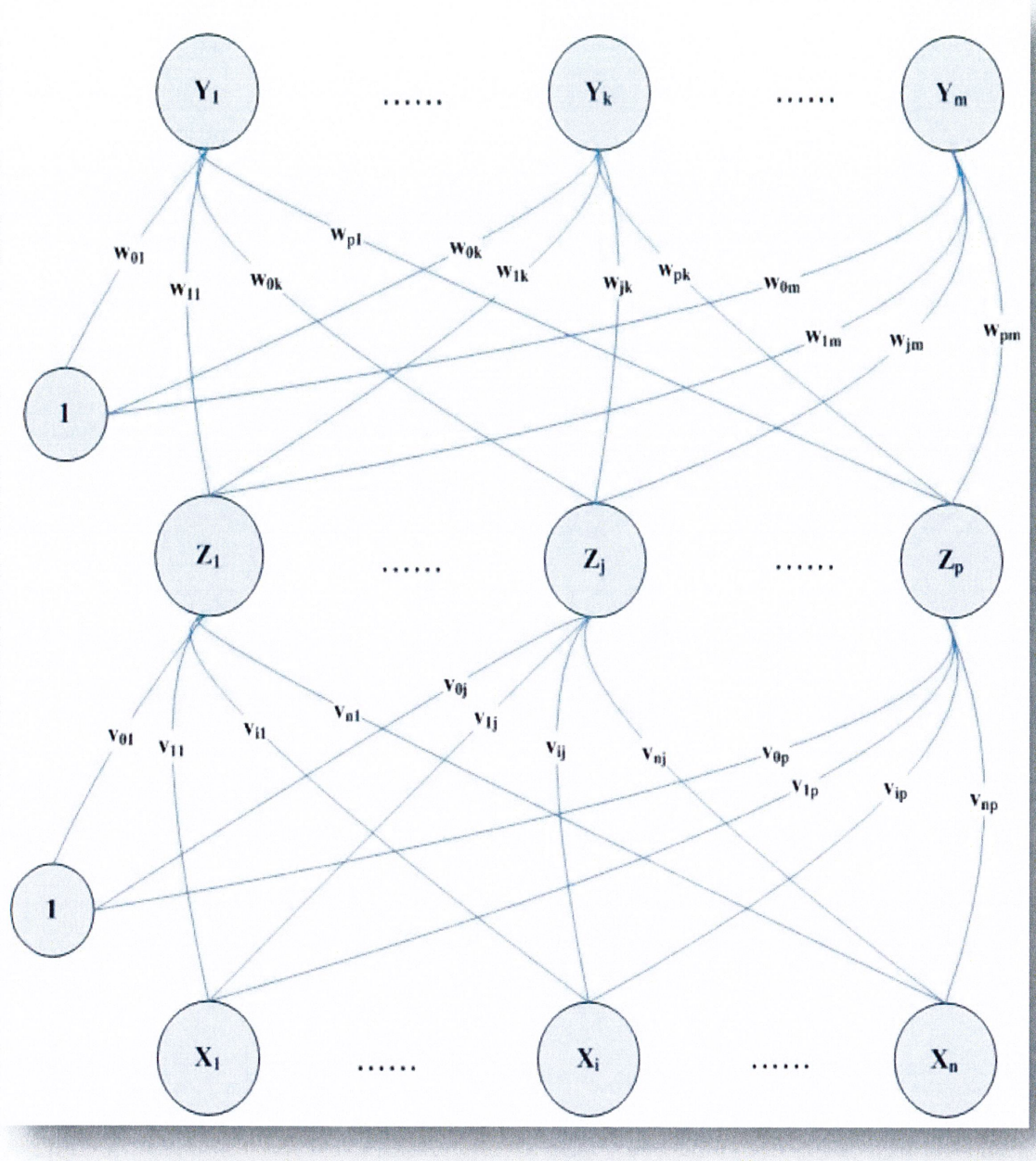


Figure (16)

Backpropagation Neural Network (Fausett, 1994)

In this research the Backpropagation will be used in the proposed system as a type of ANN.

Table (5)

Backpropagation Training Algorithm Notations and Working

Notations	
X	The input training vector $\mathbf{x} = \{x_1, \ldots, x_i, \ldots, x_n\}$
T	The target output vector $\mathbf{t} = \{t_1, \ldots, t_k, \ldots, t_m\}$
δ_k	Error backpropagated from output unit to hidden unit.
δ_j	Error backpropagated from hidden unit to input unit.
α	Learning rate.
$\mathbf{X_i}$	Input unit i.
v_{0j}	Bias on hidden unit j.
$\mathbf{Z_j}$	Hidden unit j.
w_{0k}	Bias on output unit k.
$\mathbf{Y_k}$	Output unit k.
Working	
Steps	**Instructions**
0	Initialize weight (Set to small random values)
1	While stopping condition is false, do steps 2-9
2	For each training pair, do steps 3-8
3	Each input layer (X_i, $i = 1, \ldots, n$) receives input signal x_i and broadcasts these signals to hidden units above.
4	Each hidden units (Z_j, $j = 1, \ldots, p$) sums its weighted input signals, $z\_in_j = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$, applies its activation function to compute the output signal, $z_j = f(z\_in_j)$, Send output signal to above output layer
5	Each output unit (Y_k, $k = 1, \ldots, m$) sums its weighted input signal, $y\_in_k = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$, applies its activation function to compute the output signal, $y_k = f(y\_in_k)$
6	Each output unit (Y_k, $k = 1, \ldots, m$) receives a target pattern corresponding to the input training pattern, computes its error information term, $\delta_k = (t_k - y_k) f'(y\_in_k)$, calculates its weight correction term (used to update w_{jk} later), $\Delta w_{jk} = \alpha \delta_k z_j$, calculates its bias correction term (used to update w_{0k} later), $\Delta w_{0k} = \alpha \delta_k$, and send δ_k to units in the layer below,

7	Each hidden layer (Z_j, $j = 1,\ldots\ldots, p$) sums its delta input from the units in above layer. $\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk}$. calculates error information by multiplying with the derivative of activation function. $\delta_j = \delta\_in_j f'(z\_in_j)$. calculates its weight correction term (used to update v_{ij} later). $\Delta v_{ij} = \alpha \delta_j x_i$. calculates its bias correction term (used to update v_{0j} later). $\Delta v_{0j} = \alpha \delta_j$
8	Each output unit (Y_k, $k = 1,\ldots\ldots, m$) updates its bias and weights ($j = 0,\ldots\ldots, p$): $w_{jk}(new) = w_{jk}(old) + \Delta w_{jk}$. Each hidden unit (Z_j, $j = 1,\ldots\ldots, p$) updates its bias and weights ($i = 0,\ldots\ldots, n$): $v_{ij}(new) = v_{ij}(old) + \Delta v_{ij}$
9	Test stopping condition.

3.2.4.1 Other Major Components of the Artificial Neural Network

Weights are the value or the strength of the input signals; moreover, they play a vital role in the learning phase and the output signals. Accordingly, setting the weights is called training and is a vital matter. Weights will update between any two layers by taking the derivative from the upper unit and the activation from the lower unit. Networks must select an initial weight or bias between (-0.5) and (+0.5) or (-1) and (+1) randomly to protect the derivative of the activation from not being able to detect zero (Fausett, 1994).

Another component is the bias that is linked in each layer; the bias is just another neuron added to the neural network to help it in the learning process. In addition, it has a constant value (usually +1). Bias is very common to ensure balance in the network and allow the output to obtain a zero value when needed. The bias is shown in the yellow circles in Figure (17) (Beale, Hagan, & Demuth, 2015).

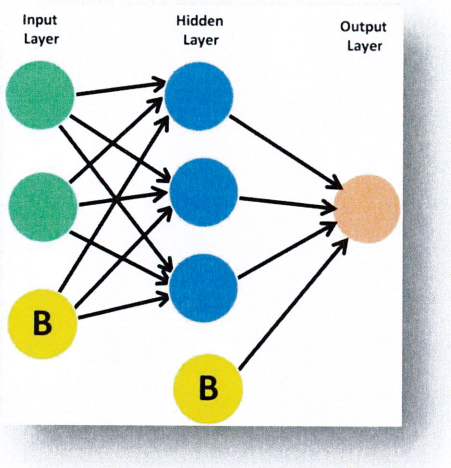


Figure (17)

Add Bias in Neural Network

The epoch is another component that is defined as the training iteration or the repeated updating of the weight. Therefore, it is the number of cycles for training a set of data associated with a certain weight. There is no standard number of epochs. This will depend on the type of the problem to solve, the number of hidden layers and their neurons, and the size of data set (Carney & Cunningham, 1998) Figure (18).

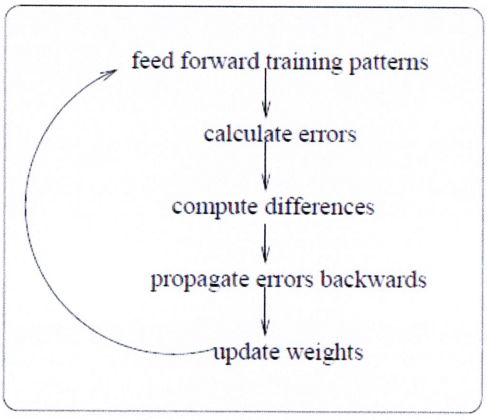


Figure (18)

The Backpropagation Neural Network Epoch

(Carney & Cunningham, 1998)

To minimise the error, the gradient descent component is the error function that adjusts the delta error before implementing it to the new weights that need updated; the update of the weights requires the gradient of error. See equation 7.

$$\Delta \text{Wij} = n(\partial \text{E}/\ \partial \text{Wij}) \qquad (7)$$

Δ Wij = Weight updates of the i^{th} neuron and j^{th} neuron.

n = Learning rate.

$(\partial \text{E}/\ \partial \text{Wij})$= The error gradient.

If the error gradient is positive, it indicates an increasing error; if negative, it indicates a decreasing error (Chakraborty, 2010).

There are two main factors for optimising the gradient descent: the learning rate and the momentum.

The learning rate is defined as the quantity of the weight that changes through the training phase. If the learning rate is slower, it will take more time; in contrast, if it is fast, it could skip a good solution. Therefore, small values will give accurate results. The value of the learning rate could be constant or variable, between 0 and 0.5 (Rojas, 1996).

The momentum is a technique used to decrease the altering rate of the weight or the entire training process, which could produce unacceptable or inaccurate data (Sutskever, Martens, Dahl, & Hinton, 2013).

The initial values of the learning rate, and momentum are selected by the user based on the problem.

3.2.5 Training Modes

The benefit of the training mode is to reduce the rate of error. The training mode is divided into two modes, the batch and pattern modes. When the average error is calculated with the adjusted weight based on the entire (batch) set of data, that is, when one epoch is accomplished, it is a batch mode. In the pattern mode, the error average is calculated over each pattern existing in the network. Generally, when comparing between these two modes, in order to select the best mode for solving the precise problem the following factors apply:

- Batch mode involves fewer weights to update; therefore, it is faster to learn than the pattern mode.
- Batch mode provides results that are more accurate.
- Batch mode is more sensitive for initial weights than pattern mode (Rafiq, Bugmann, & Easterbrook, 2001).
- The Backpropagation used in this studies is based on the batch mode because it adjusts the weights after each one iteration (Mas & Flores, 2008).

3.2.6 Least Mean Sum of Squares Error Function (SSE)

This function improves the performance and speed of the training. The error is calculated by taking the difference between the current and the desired output. In addition, this function calculates the least mean square error of the desired and the current output and sums the values; the error will propagate back from the output layer to the previous layer to modify the weights before the next cycle of learning (Rojas, 1996).

3.2.7 The Advantages of Using ANN

In addition to the ability to obtain important data from rough, complex data, there are additional advantages as follows:

- Adaptive learning: the capability of learning based on the raw data in the training phase or a new experience.
- Self-organization: through the learning phase, the data creates and organises information automatically.
- Real-time operation: the parallel process or computation via a special construct designed to assist this property (Rajthilak & Sasikumar, 2008).

Chapter 4: Analysis and Design of a Prototype System

4.1 The Proposed System.

4.2 Odour sensing system (e-nose).

4.3 Data Collection.

4.3.1 Data Description.

4.4 Data Preparation.

4.4.1 ID or Classification Input.

4.4.2 Strength Odour Input.

4.4.3 The Balance between Bad and Good Meat through Both Inputs.

4.4.3.1 The Difference between the Conditions of Chicken.

4.4.3.2 The Difference between the Conditions of Beef.

4.4.3.3The Difference between the Conditions of Mutton.

4.5 Artificial Neural Network structure.

4.5.1 Designing ANN Software.

4.5.2 Importing Training Data Sets.

4.5.3 The Flow Code of ANN.

4.5.4 Training Phase.

4.5.5 Performance Report.

4.6 Results.

4.7 Multimedia Elements.

4.1 The Proposed System

The system in this proposal will focus on the e-nose that imitates the function sense of smell which has been largely ignored. The first component of system proposed includes a manual chemical odour-sensing device (handheld odour meter [OMX-GR sensor]), the role of this sensor is to catch the odours with their different chemical natures and with different concentrations. This is an array sensor that includes automated pattern recognition. The second component is the ANN, which is a vital component of the system to identify, classify, and recognise the variety of odours. The proposed ANN uses the network training multilayer with backpropagation error algorithm. This prototype system was trained on three types of meat: beef, mutton, and chicken. The system will also measure the quality of the meat (whether it is edible or has gone bad. Therefore, there will be six classifications indicates to three type of meat and for each type it will be two condition the bad and the good condition.

In this chapter, we will discuss the details of the proposed prototype, which contains two main segments: the odour-sensing device and the ANN. Then, there will be a discussion on the analyse and results.

4.2 Odour sensing system (e-nose)

The chemical odour sensor used in this study is the handheld odour meter (OMX-GR sensor; Figure (19)). This sensor contains some of features and specifications are detailed below:

- It includes two semiconductor gas sensors.
- It has the ability to detect a variety of odours from combustible gases.

- It has the ability to measure continuous sampling with a built-in air pump.
- The range of detection is between 0.1 ppm to 200 ppm for ethanol.
- The measures display on the LCD of the sensor. The strength of an odour is from 0 to 999 and the classification is from 0 to 89.
- The response time is consider good, for instance, ethanol at 100 ppm is within 20 seconds, methyl mercaptane at 0.16 ppm is within 50 seconds, and xylene at 46 ppm is within 50 seconds.
- The function can be in real-time sampling or memory sampling.
- The temperature has a small effect on the sensor. In addition, humidity has an effect on the sensor; lower humidity could slightly increase the measurement of the strength and classification values, but high humidity may decrease the values of both. For our study sample, the temperature was 22°C, and the humidity was 54% approximately.

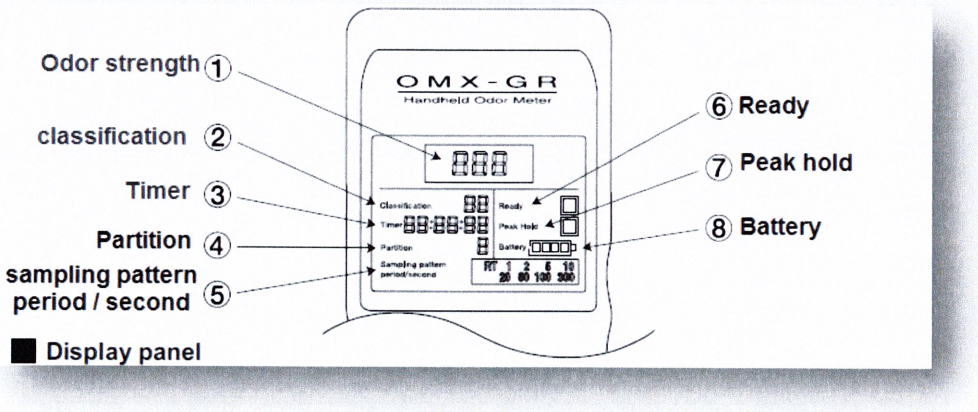


Figure (19)

Handheld Odour Meter (OMX-GR sensor) Display Panel

4.3 Data Collection

The sensor used in this study provides two ways for measuring functions that can be selected by the end user, after connecting the sensor to a computer through the computer's RS232C terminal (9 pinD-SUB type) with the cable supplied with the OMX-GR. Moreover, we used an extra USB cable to transfer the data sample to the laptop, after installation of the software SDDF.EXE. The form allow the user to choose between the measurement modes: real-time sampling or memory sampling Figure (20). Each one will shift to another dialog box. When the user chooses the first option, it will ask the user to choose the desired file, and if the user chooses the second option, it will open the third form.

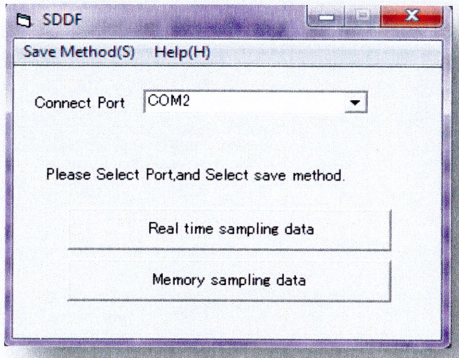


Figure (20)

Selection between the Measurement Modes

The real-time sampling data option will open a dialog box to create a data file with the extension .txt, where it will be saved Figure (21).

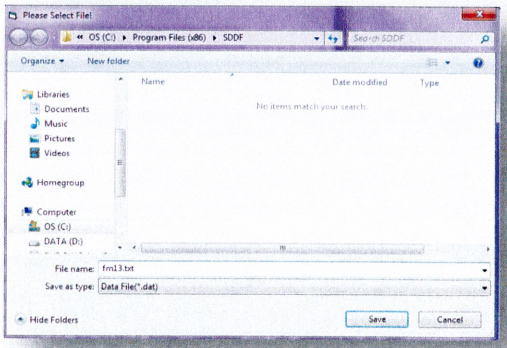


Figure (21)

Creating a Data File

Then, the program will start the process for real-time sampling. After pressing start, the data will begin to initiate and transfer to the data file Figure (22).

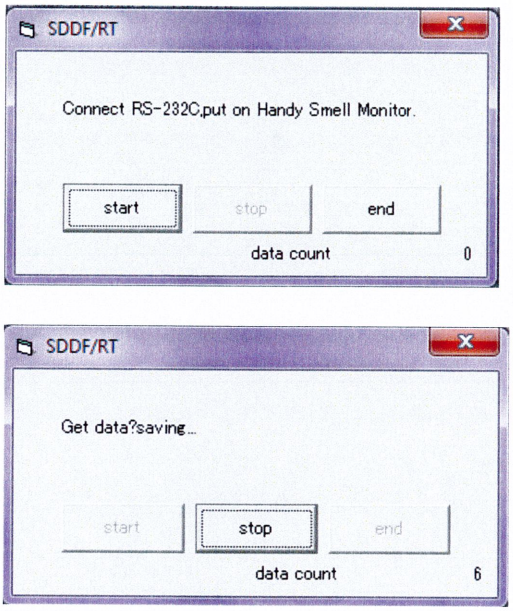


Figure (22)

Initialising the Data File to Transfer the Data

The process for transforming the data will create a database automatically. The file data contain the time and date of the sample. Further, the two main values are the ID of the odour classification with a two-digit numeric value from 0 to 89, and the current strength of the odour with a three-digit numeric value from 0 to 999; however, it can indicate more than 999 Figure (23). Measuring data will be continuous until the user presses stop to end the measuring process. In addition, the user can pause and resume the measuring in the same data file, which will automatically provide the resume time and date.

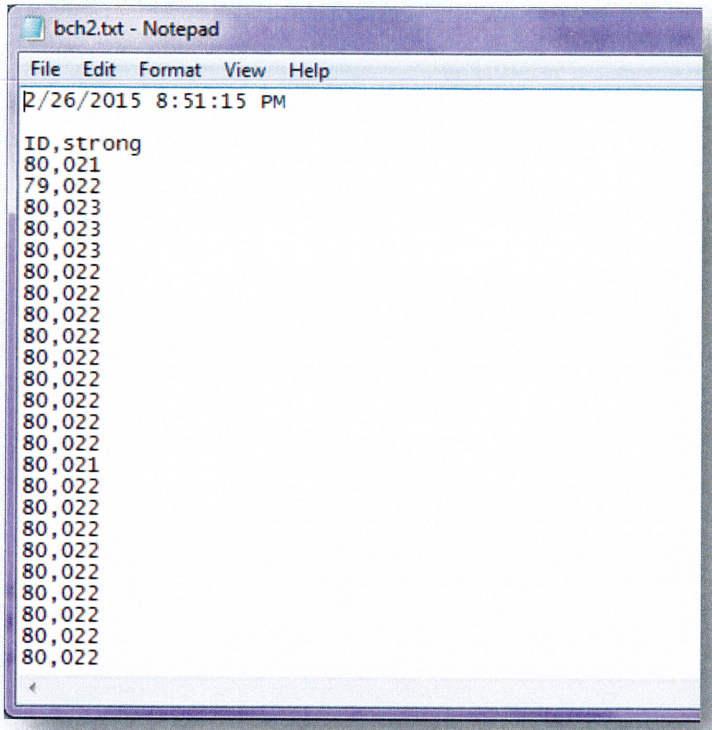


Figure (23)

The File Data

The file data is used as a training session through the proposed odour-sensing device, which will digitise the data based on the software implementation in order to identify the odour samples.

4.3.1 Data Description

The study will investigate the discrimination between two conditions of the same type of food, good food and bad or damaged food. The meat categories that will be examined in our prototype include three types of meat: beef, mutton, and chicken. For each type of meat there will be two conditions: fresh or good and bad or damaged. Each state will include approximately 2356 units of data or more; therefore, each type can produce 4712 data points or more. This implies the total number of data points that will be used to train the proposed system is approximately 14136.

The data must be prepared by modifying the data because the ANN only accepts certain forms of data. The modified data is categorised into

six files from the total actual file numbers estimated at 210 .txt files. The first two digits indicate the classification ID, and the next three digits specify to the odour strength, the comma will separate the previous data from the result of good or bad food. See the example of a modified file in Figure (24). The one signifies good food; in contrast, zero indicates damaged food.

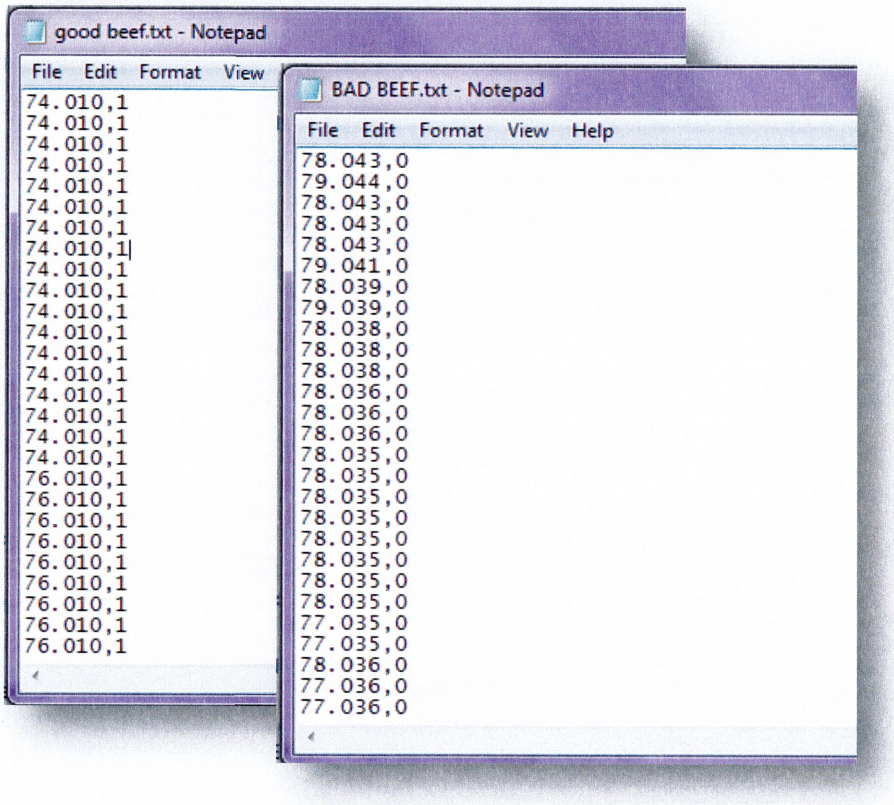


Figure (24)

Example of the Modified Data Used in Coding

Photos of the sampling process with different types of food are shown in Figure (25).

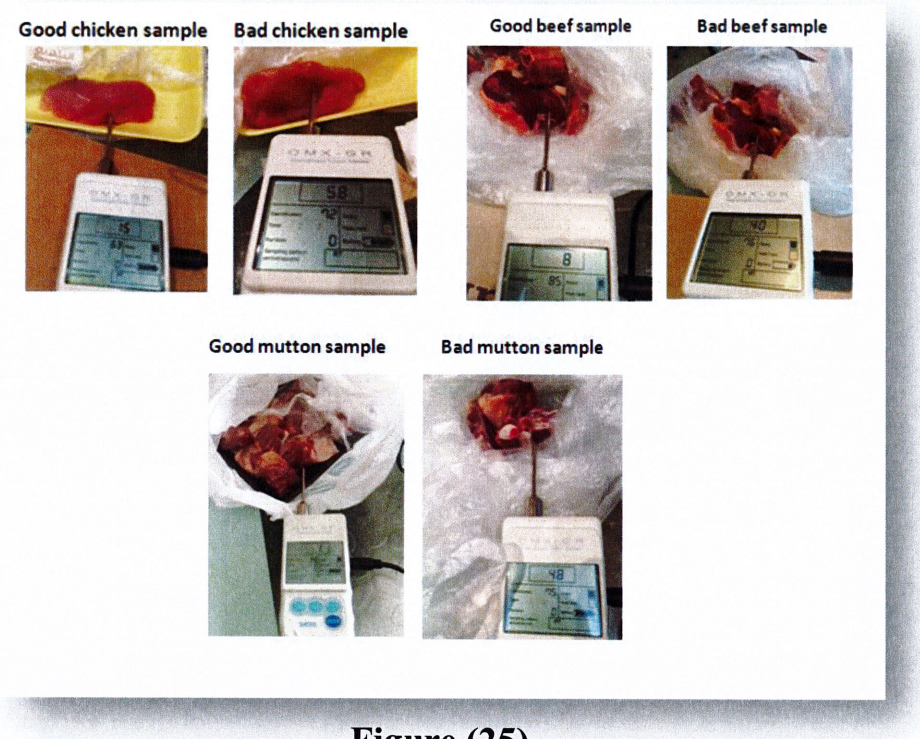


Figure (25)

Example of the Sampling Process

4.4 Data Preparation

The neural network requires a specific format to process data. Raw data can include correct and erroneous data. To ensure accuracy, it is necessary to ignore some errors, which can be judged through the number of lower frequency numbers and the incomplete portions of the individual data points. For example, each data set has two inputs: the ID number (classification number) and the strength of the odour; therefore, if any part is missing, it will be considered an error. In other words, it selects the top quality data.

In this stage, we will prepare the raw data before in putting it into the neural system to ensure the highest performance. The next section will demonstrate the preparation of both inputs and then combine them together to present the final difference between good and bad meat.

4.4.1 ID or Classification Input

The ID or classification input is displayed as two-digit data. After the final data is obtained, a range is chosen to discriminate between good food and bad food, which is based on the frequency of the highest reading or measurement, while ignoring the error measurements. As can be expected, it can be difficult to have the optimal atmosphere and pure odours to measure and separate an odour from other odours. However, to get more accurate and valid data as possible the different type of meat samples has been purchased from couple of well-known butcher shops. In addition, several readings was obtained for the raw data with same type of meat among different sample source obtained. The range of the data is introduced and summarised in Table (6).

Table (6)

The Range of Data Classification for Each Type and State of Meat

Type of Meat	State	Range
Chicken	Good	60-65
	Bad	70-75
Beef	Good	75
	Bad	80
Mutton	Good	70-75
	Bad	80-90

The difference between each type of food with different states is presented in the three graphs that follow.

The different states of chicken can be recognised in the graph, as good chicken, which is represented by a blue line, is lower than the bad smell of chicken, which is represented by the red line. This result is logical, as when the duration of time increases, the smell of the sample results in higher values and increases Figure (26).

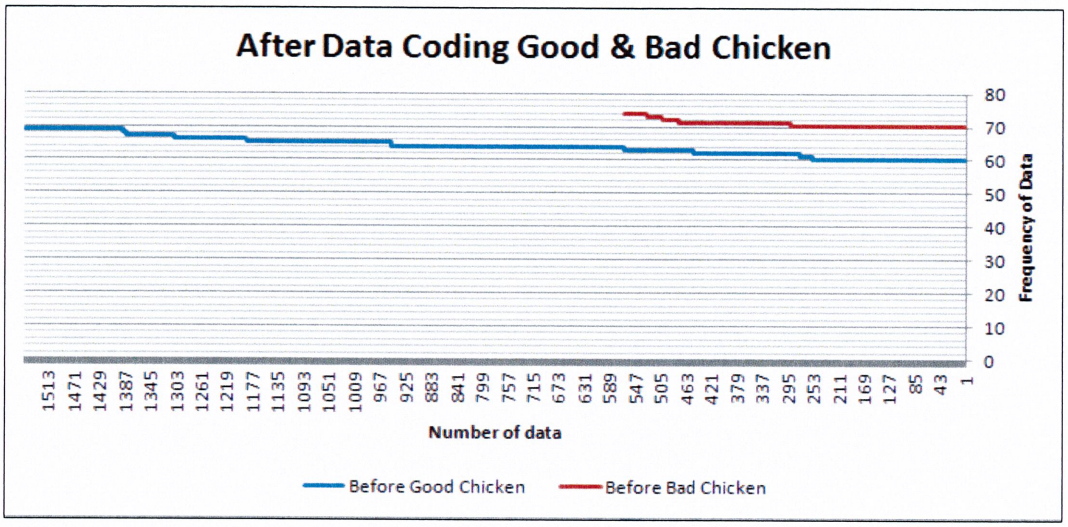


Figure (26)

The Results of Chicken Classification for Good and Bad Chicken

The different states of beef can be recognised in the graph, as good beef, which is represented by a blue line, is lower than the bad smell of beef, which is represented by the red line. Additionally, both the blue and red lines increased over time Figure (27).

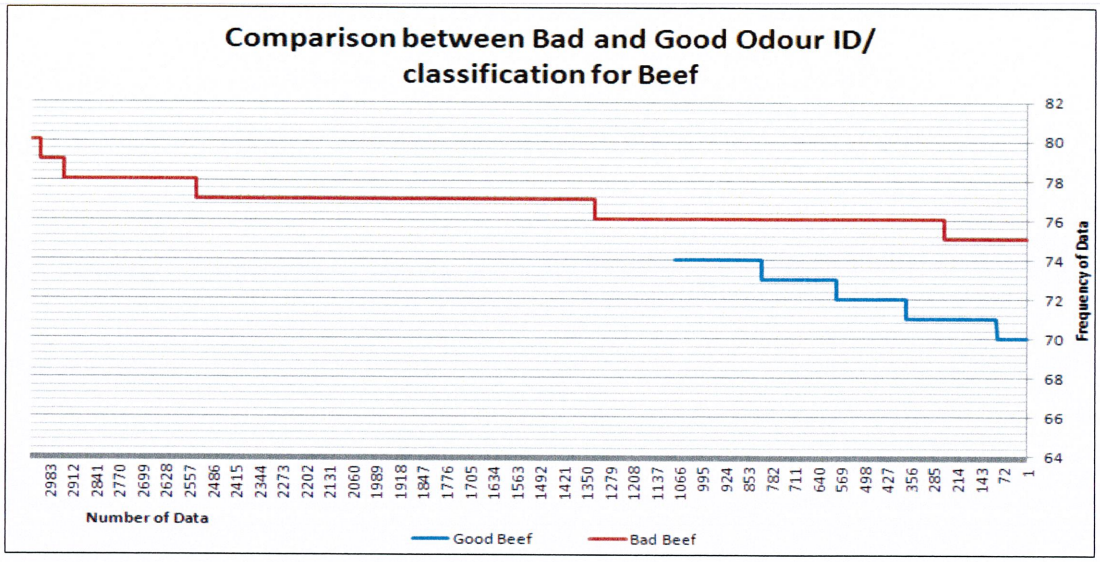


Figure (27)

The Results of Beef Classification for good and Bad

The different states of mutton can be recognised in the graph, as good mutton, which is represented by a blue line, is lower than the bad smell of mutton, which is represented by the red line. Naturally both lines increase gradually Figure (28).

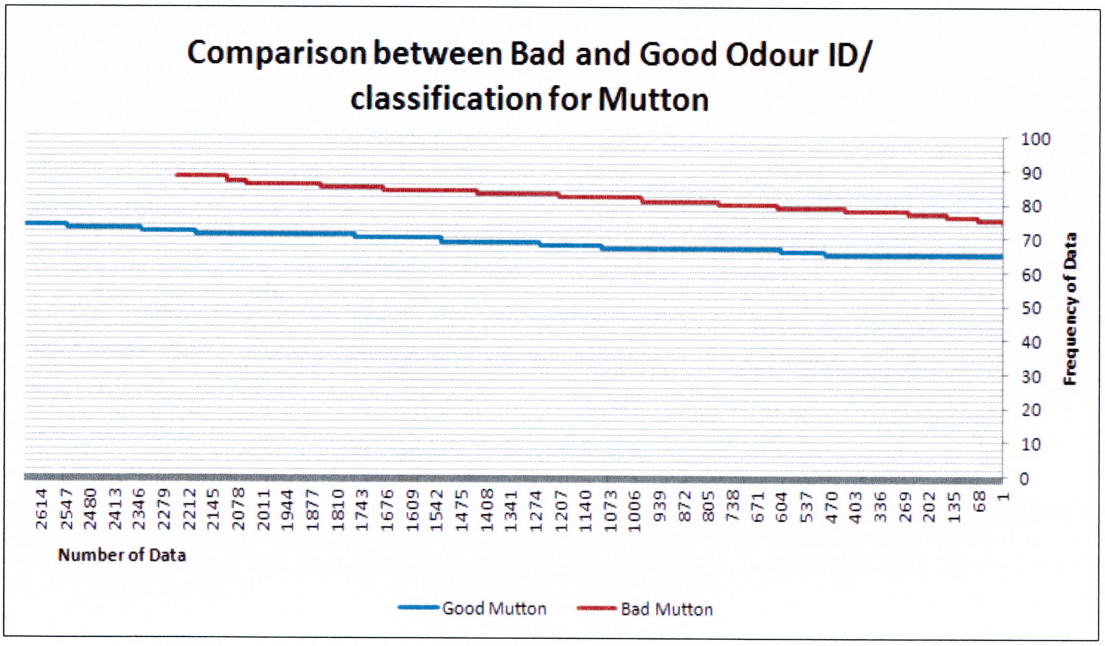


Figure (28)

The Results of Mutton Classification for Good and Bad Mutton

4.4.2 Strength Odour Input

The strength of the odour was measured and is summarised in Table (7).

Table (7)

The Range of the Strength Data for Each Type of Meat

Type of Meat	State	Range
Chicken	Good	1-20
	Bad	21-99
Beef	Good	1-15
	Bad	16-86
Mutton	Good	1-14
	Bad	15-62

The strength input is divided into three graphs that illustrate the obvious difference between the good and bad meats. The chicken strength input demonstrates a change from the good chicken at 1 to 20 to the bad range at 21 to 99; this is shown in the graph in Figure (29).

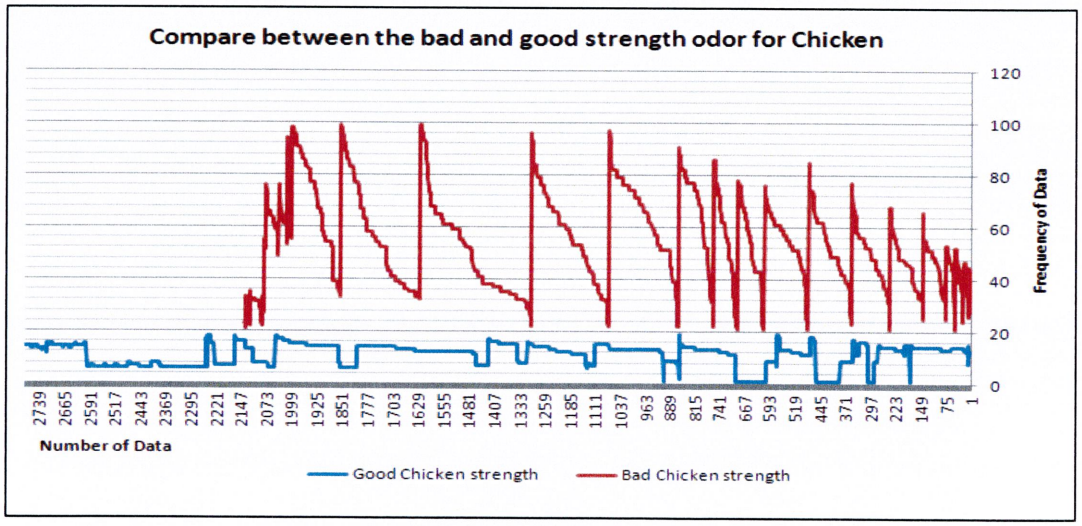


Figure (29)

Comparison of Chicken Odour Strengths between Good and Bad Chicken

The beef has a range for good beef from 1 to 15; in contrast, the bad beef ranges from 16 to 86. The graph is presented in Figure (30).

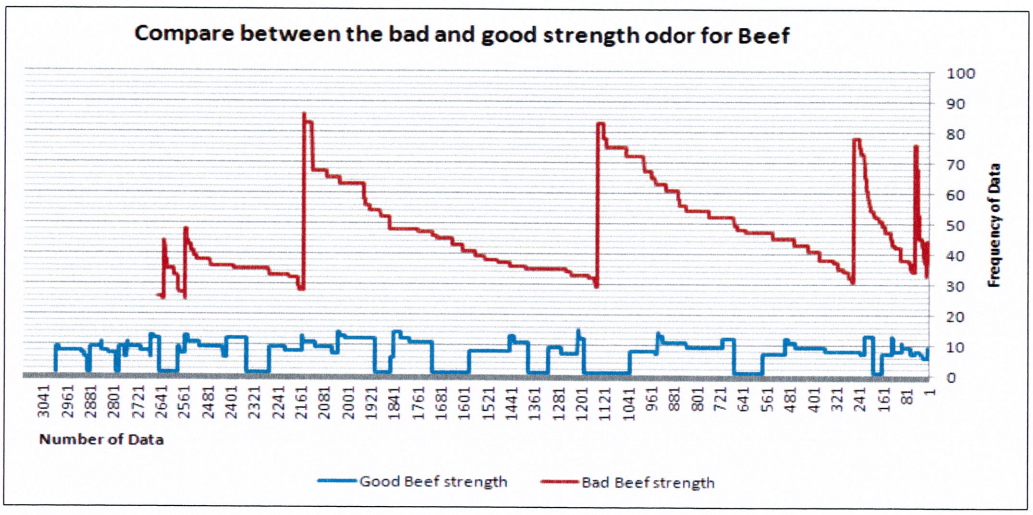


Figure (30)

Comparison of Beef Odour Strengths between Good and Bad Beef

Finally, the mutton obtains a range for good mutton from 1 to 14, which is similar to beef. In contrast, bad mutton ranges from 16 to 62 and is slightly less than beef. The graph is presented in Figure (31).

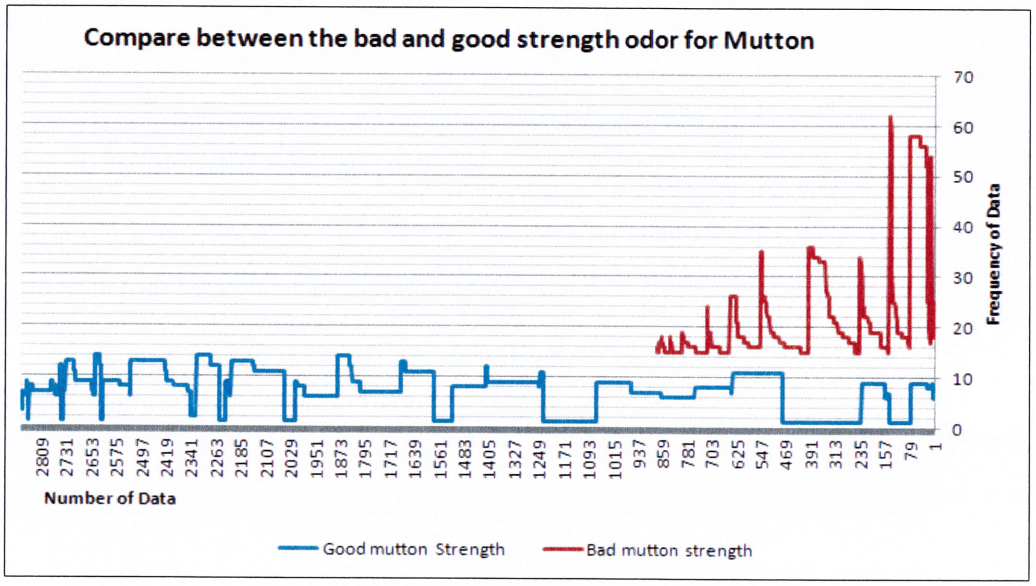


Figure (31)

Comparison of Mutton Odour strengths between Good and Bad mutton

4.4.3 The Balance between Bad and Good Meat through Both Inputs

This section will discuss the results when merging the two inputs to recognise the difference between both conditions of meat.

4.4.3.1 The Difference between the Conditions of Chicken

When the two inputs integrate (classification and strength of odour), the difference between the good and bad smells is clear; the *X*-axis shows the classification, and the *Y*-axis shows the strength Figure (32).

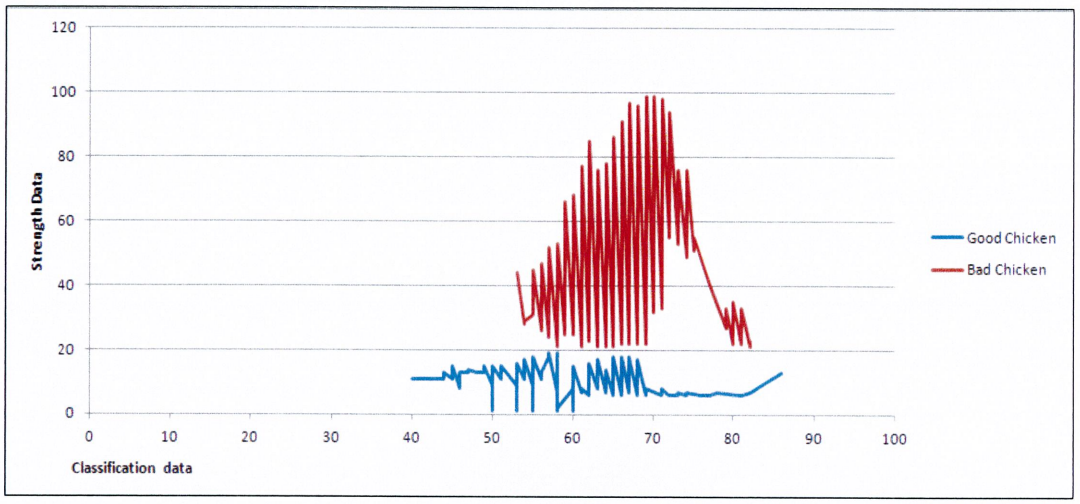


Figure (32)

Comparison of the Two Inputs between Good and Bad Chicken

4.4.3.2 The Difference between the Conditions of Beef

When the two inputs integrate (classification and strength of odour), the difference between the good and bad smells is clear; the *X*-axis shows the classification, and the *Y*-axis shows the strength Figure (33).

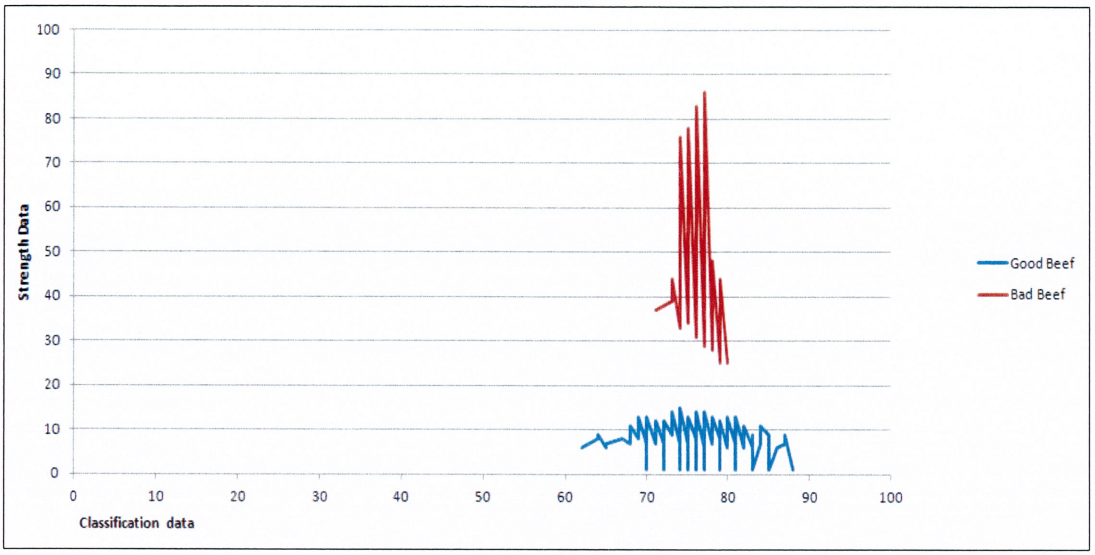


Figure (33)

Comparison of the Two Inputs between Good and Bad Beef

4.4.3.3 The Difference between the Conditions of Mutton

When the two inputs integrate (classification, strength of odour), the difference between the good and bad smells is clear; the *X*-axis shows the classification, and the *Y*-axis shows the strength Figure (34) .

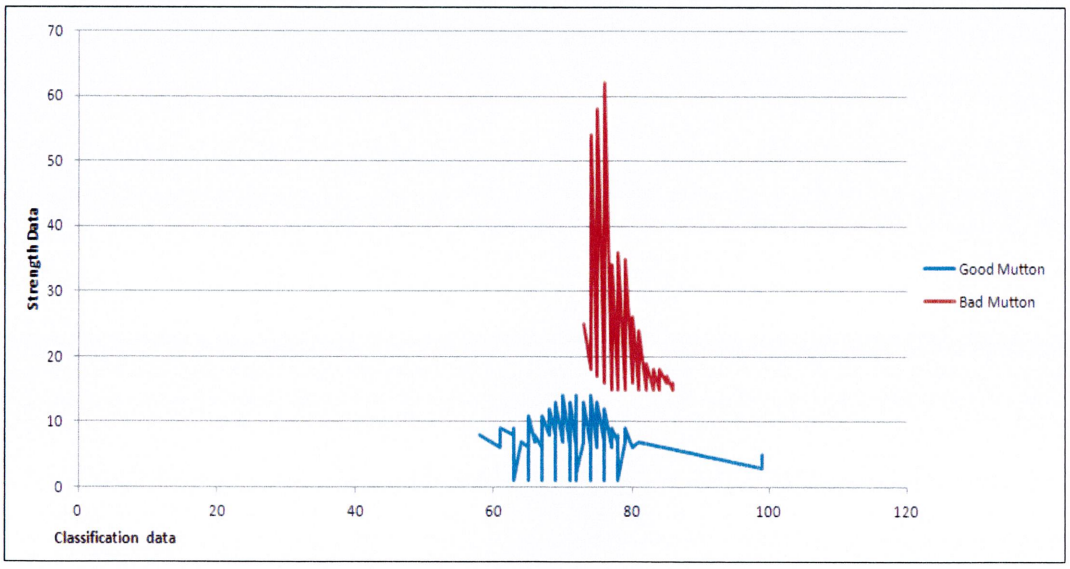


Figure (34)

Comparison of the Two Inputs between Good and Bad Mutton

The good input data have the lowest values, and the bad input data have the highest values.

4.5 Artificial Neural Network Structure

As previously mentioned, there are several kinds of structures and types of ANN. In this study, the backpropagation multi-layer ANN will be built, containing two inputs, one hidden layer, and one output. The input node includes the ID of odour and the strength, the hidden layer contains a number of nodes that have been chosen based on the results of the training, which obtained the least errors and best validation that supports the system purpose. Moreover, this study uses a sigmoid function. The sigmoid function parameters used in this study are as follows: sigmoid alpha value of 0.5 and a default output range from 0 to 1.

Network generalisation is important in order to allow the system to generalise new cases; in other words, it will train the new data within the same set. Therefore, to attain the best generalisation, one of the methods is to divide the system samples into three sets: 1) the training set (50%) used to train the network and to compute the gradient, while updating the weights and biases continually, 2) the validation (selection) set (25%), which verifies the network progress through the epoch number, which (when reached) stops the training to measure the performance of the untrained data, and 3) the last set is the testing set (25%), which can check the validation of data.

The training phase or learning phase will be stopped under two conditions.

- When it reaches the lowest limit error (this condition, the generalisation is allow and caused small amount of error according to Shashi & Minesh in thier reserach).
- When it reaches the number of epochs that should be learned.

When the training is not stopped, the data will be overstrained and the network performance will decrease as well as the error. When the training is complete, checking the testing phase is highly important and recommended (Shashi & Minesh, 1992).

The supervised ANN with the backpropagation algorithm is based on least mean sum of square error (SSE). To calculate the amount of error, the parameters used in this study after many testing and training experiments are taking from previous study (Hoyo, 2012). To check the best accuracy result with regards to our problem are as following:

- Threshold: accept (0.5), reject (0.5)
- Learning rate: 0.01
- Momentum: 0.3
- Maximum number of epochs: 50.000

4.5.1 Designing ANN Software

The proposed system was coded in C# using Visual Studio 2010; the ANN needed a special method to implement its program. See the flow chart in Figure (35).

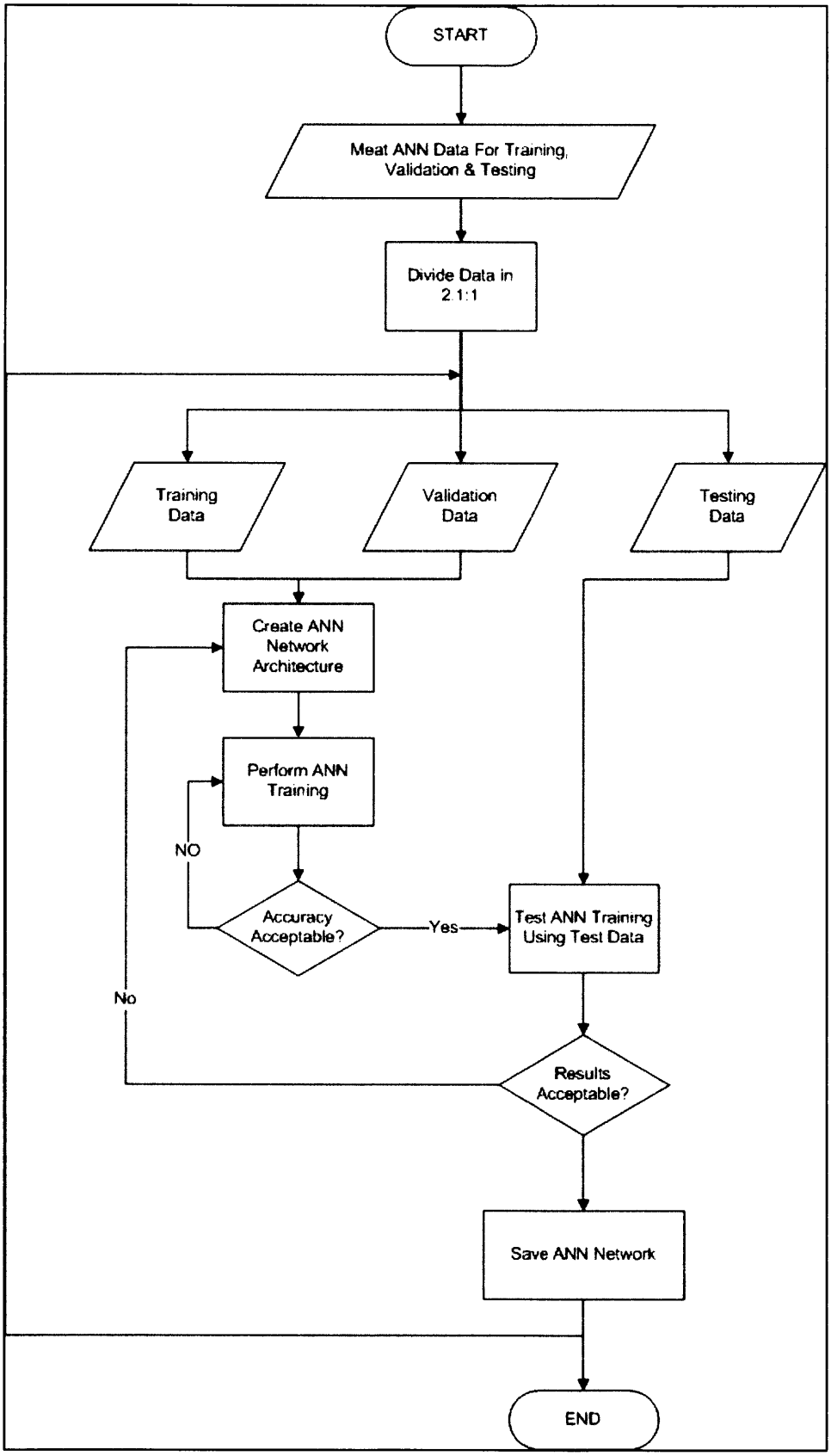


Figure (35)

Flow Diagram Showing the ANN Black Box System

4.5.2 Importing Training Data Sets

The proposed program requires data preparation, which was presented in Section 4.4. The data were prepared with Microsoft Excel, and the final data will be in the comma separated values (CSV) format, which can be imported into ANN software.

To import data into the system:

1. Open the ANN program icon, which is named electronic nose Figure (36).

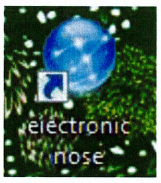


Figure (36)

The Proposed Program Electronic Nose

2. From the main menu, select the file then choose *Run Wizard* Figure (37).

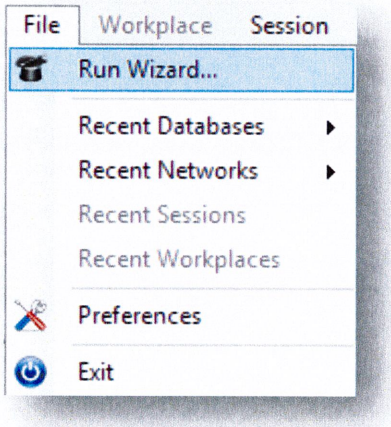


Figure (37)

Choose Run Wizard to Import Data

3. Then, a welcome window appears named *Import Data Wizard*; choose *next* Figure (38).

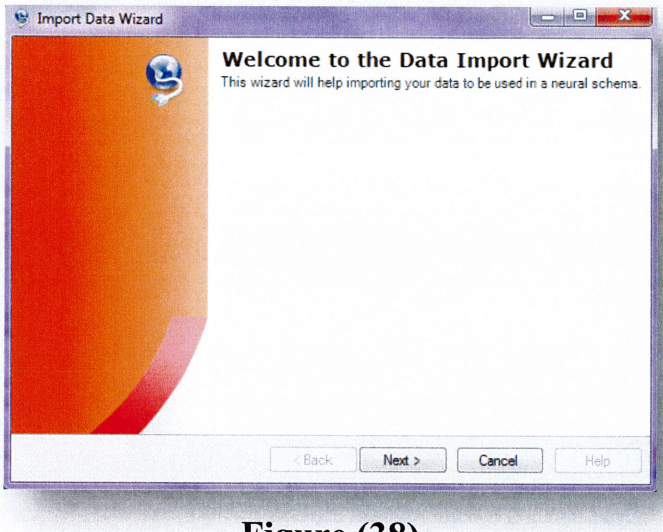


Figure (38)

Import Data Wizard Window

4. Then, browse to the location of the training data for the ANN system; the field delimiter character must be chosen, which can be separated in some cases as (tabulation, comma, space, or semicolon). In this system, the semicolon was used. Choose *next* Figure (39).

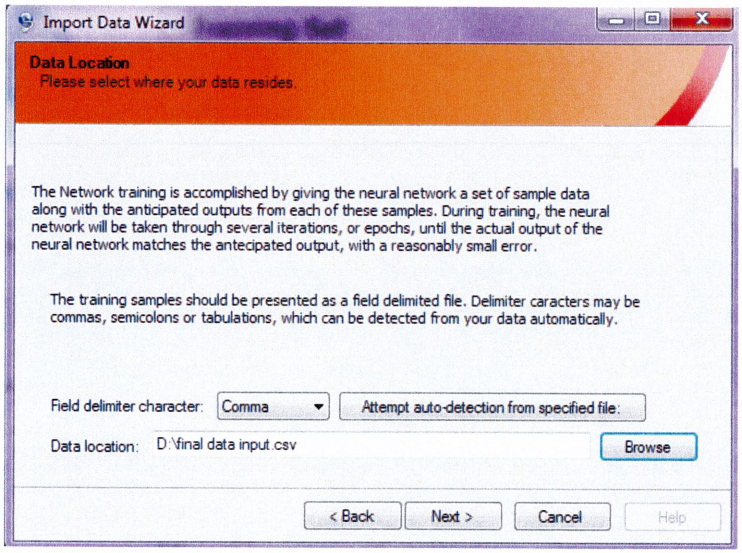


Figure (39)

Selection of the Data Location

5. In the data preparation step, data must be organised into columns and titled. Each column is based on the required data. In this step, input variables are selected, which are classification and strength Figure (40).

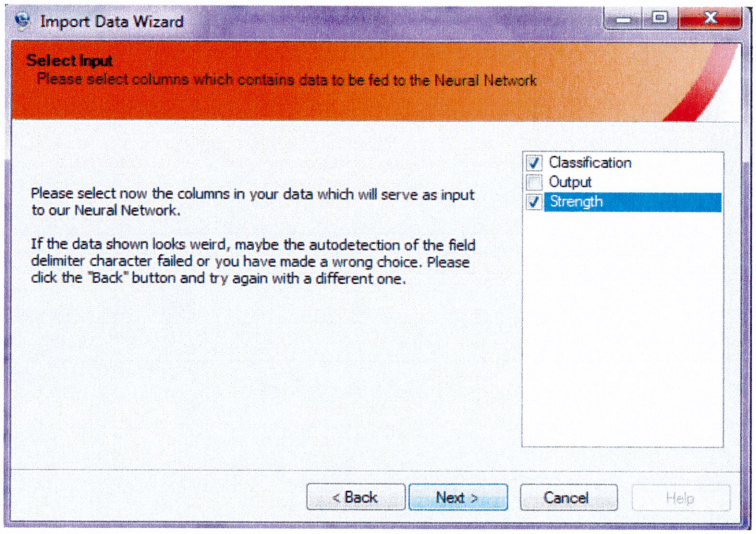


Figure (40)

Select Input

6. Then, output variables are selected, which are simply called output, the output in the proposed system has only one neuron with a value of false =0 or True =1 Figure (41).

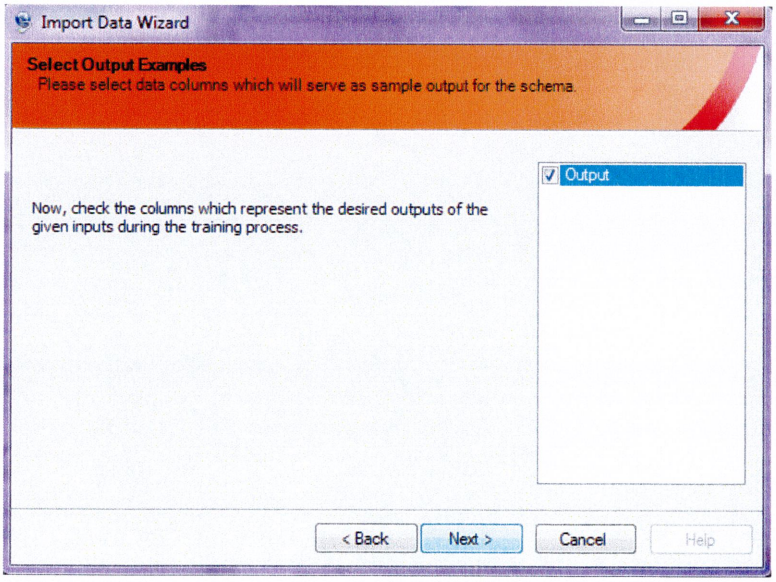


Figure (41)

Select Output

7. After the data import, the data must be shuffled to achieve more accuracy and reality. The shuffle icon is depicted in Figure (42).



Figure (42)

Shuffle Icon in ANN

8. All data samples are fed into the ANN software and divided into three subsets with (14136) total, which is split into a training set (7068), validation set (3534), and testing set (3534) in a ratio of 2:1:1, which is shown in Figure (43).

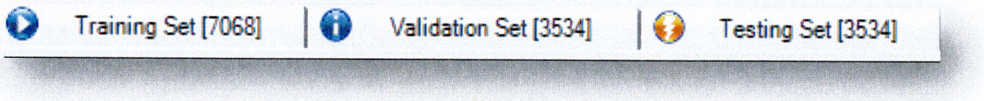


Figure (43)

Data Sample Ratio

9. The parameters are set as follows in the ANN software for all training attempts:
   - Threshold: accept (0.5), reject (0.5)
   - Learning rate: 0.01
   - Momentum: 0.3
   - Maximum number of epochs: 50.000

To set the parameters, select *New* from the Network tab main menu to create a new network. The window appears to adjust the previous parameters that have been chosen, which were taken from another system in previous studies Figure (44), (Reitermanová, 2010; Emanuele, Panicker, Gurbaxani, Lin, & Unger, 2012).

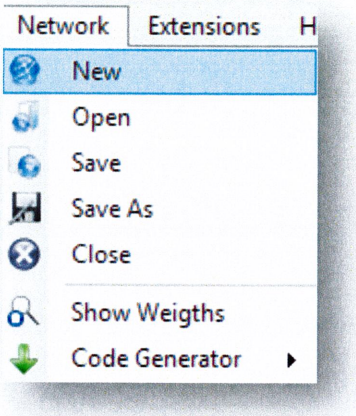


Figure (44)

Create New Network

10. Set the number of hidden layers to 1, and the number of neurons in hidden layers from 1 to 15, which has been tested in this system. For each particular test, all the steps are required to be repeated with the changing parameters to attain the best learning rate that can be measured by the amount of errors and the validation. The activation function that used is sigmoid, which suits the station of the output Figure (45).

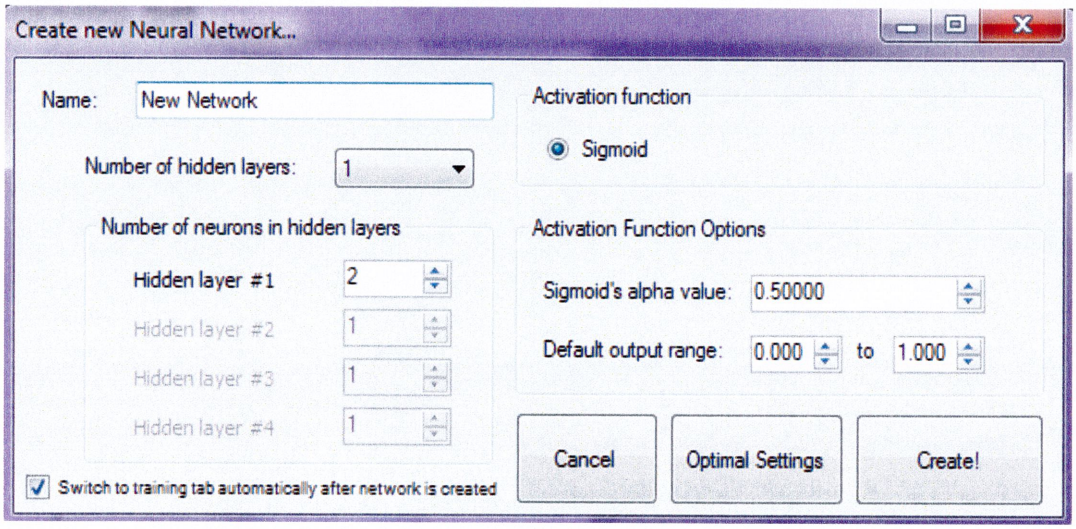


Figure (45)

Create and Set the Parameters of the Network

11. Then, it is also required to set the network trainer, which can trained the network in two ways: by the number of epochs or through the error limit; in this study, training is via the number of epochs. Furthermore, the learning rate is 0.01 and the momentum is 0.3; there is another option to save the data at certain points (every 1.000 epoch) Figure (46).

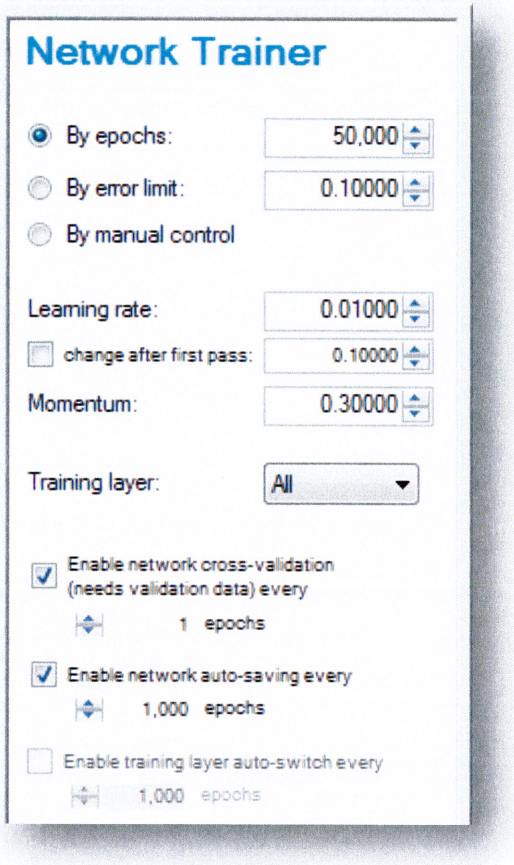


Figure (46)

Setting the Network Trainer Parameters

4.5.3 The Flow Code of ANN

The proposed program is based on other methodologies of the ANN programs; in this study, C# is used to code the ANN program, and the program is divided into the following steps:

1. Create the ANN architecture. Appendix 1,2,3

2. Create the network and activate the activation function (sigmoid). Appendix 1,4
3. Initialise the weight, random weight. Appendix 3
4. Setting the parameters of ANN. Appendix1,5
5. Setting the number of epochs. Appendix 5
6. Setting threshold/alpha value. Appendix 4,3
7. Run the backpropagation while the learning phase is starting, compute the output, which is dependent on the desired output and the input fed into the ANN.Update the weight and in the array until it reaches the desired limit error or number of epochs. Appendix 7
8. Validate and test the network .
9. Run query form for the new data, and get the result with an image and sound too. Appendix 6,7

- Database:All data will be stored in a database labelled Sinapse.Databases Figure (47).

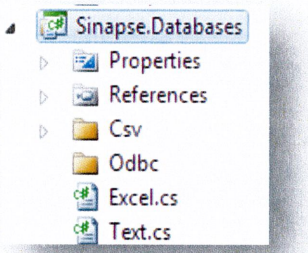


Figure (47)

Sinapse.Databases

- Graphical user interface (GUI): all code will be in Sinapse.Forms.Controls and Sinapse.Simplifier Figure (48).

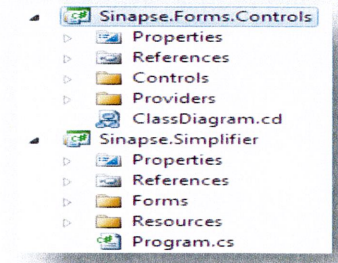


Figure (48)

GUI Code Items

- Run wizard: when importing the data into ANN, in the main menu bar, select *file*, then *Run wizard*, Figure (49).

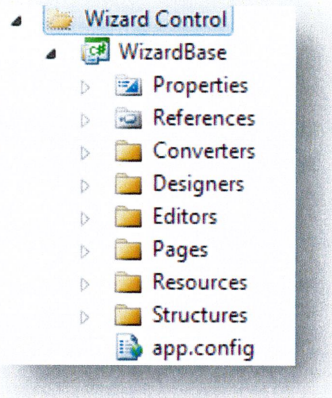


Figure (49)

Wizard Control

- The library of ANN is a mixture of C# and C++, defined in the AForge.NET subfolder in the main folder *lib* Figure (50).

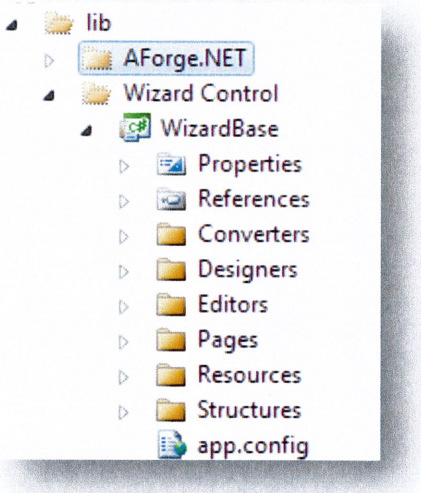


Figure (50)

AForge.NET

- The main libraries used include the following:
  - AForge.Core.
  - AForge.Mathematics.
  - AForge.Neuro, which includes:
    - Activation functions, which contain the SigmoidFunction.
    - Layers.

- Learning: contains the backpropagation function.
- Networks: creates the ANN architecture with the network activation.
- Neurons: computes the weighted sum of its inputs, adds the threshold value, and then applies the activation function Figure (51).

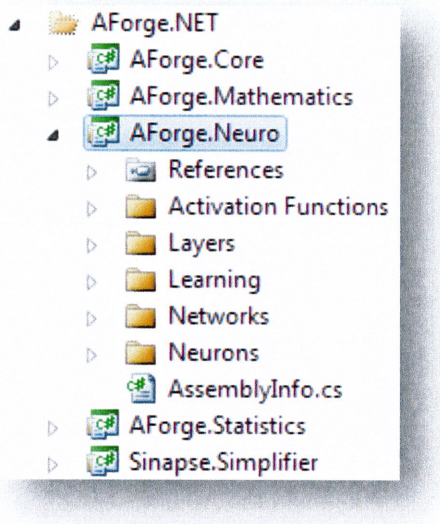


Figure (51)

AForge.NET and AForge.Neuro

4.5.4 Training Phase

The training phase requires parameters to be set, as shown in Section 4.5.2. Then, it will create a graph plot showing the progress of the performance of the ANN by monitoring a special function called the least mean sum of square error (SSE). The training graph has been created where the *X*-axis presents the number of epochs, and the *Y*-axis presents the number of least mean square errors, as shown in Figure (52). The training will stop when the epoch reaches 50.

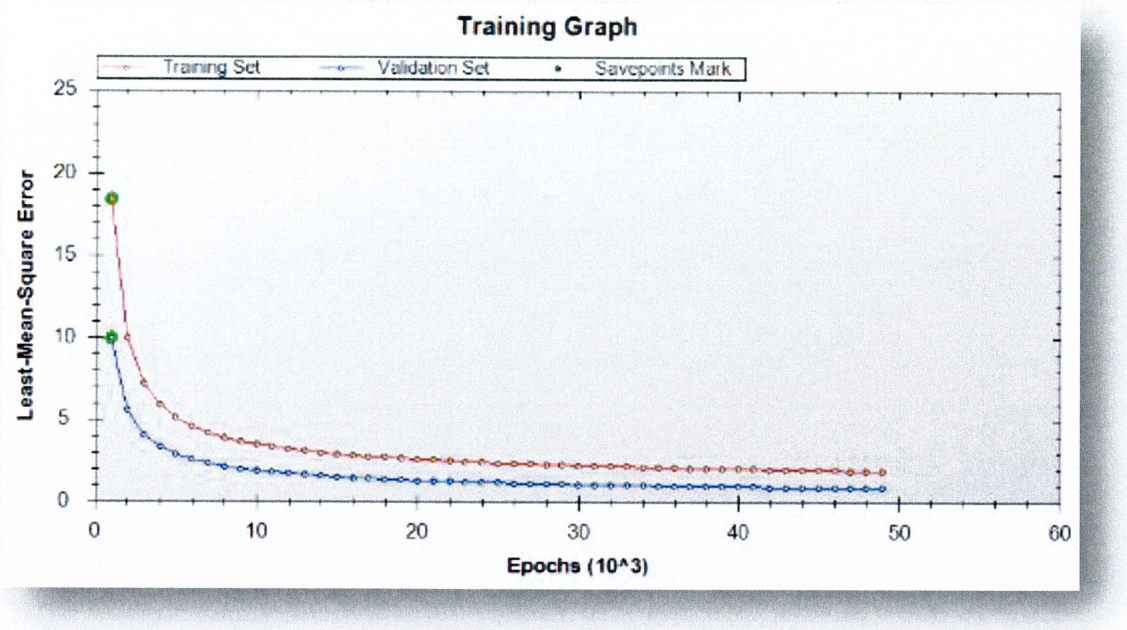


Figure (52)

Training Graph

4.5.5 Performance Report

Figure (53) displays a summary or performance report for the entire training, containing the following:

- Network details (name, type, layout, function, and description).
- Network schema (Input [classification or strength] and output).
- Training details (training entries, validation entries, and training deviation).
- Testing summary (testing entries, testing items, hits [positives/ negatives], errors [positives/negatives], total deviation, and final deviation).
- Testing details (network test performance table).
- Data normalisation (activation function range).
- Network weights (for each layer).

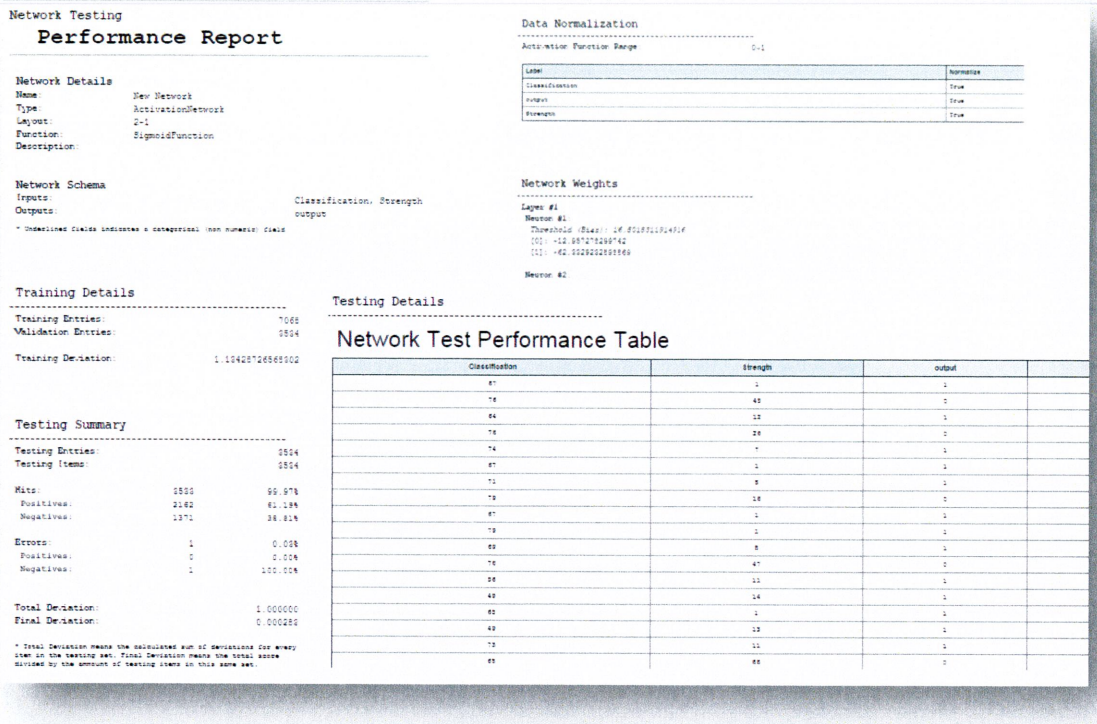


Figure (53)

Performance Report for ANN system

4.6 Results

The final evaluation of the ANN proposed system uses the least mean sum of square error for the selection data. There are many architectures of ANN through using a variety of neuron numbers and different parameters, which result in around 91 different ANN architectures. The best ANN architecture has a ratio of 2:6:1, as it scored the best error limit and validation with the best test data as shown in Figure (54). It has one input layer with two neurons, one hidden layer with four neurons, and an output layer with one neuron. The best ANN architecture scored 1.72729 in training error, 0.77849 in validation, and 0.03% in testing. The 14 results from the various ANN architectures must obtain 100.00%-99.24% accuracy to obtain the test set results necessary to compute the results and compare the output of data and outputs calculated by the network. Therefore, it is necessary use the round

function to the nearest integer to obtain the desired output, which is 0 if bad or 1 if good. After the round function is completed, the network test error obtains a very low error, which is 0.03%. This indicates that the design software works with high performance, and because all of the measures have some similarities, this causes the same test error value, as presented in Table (8).

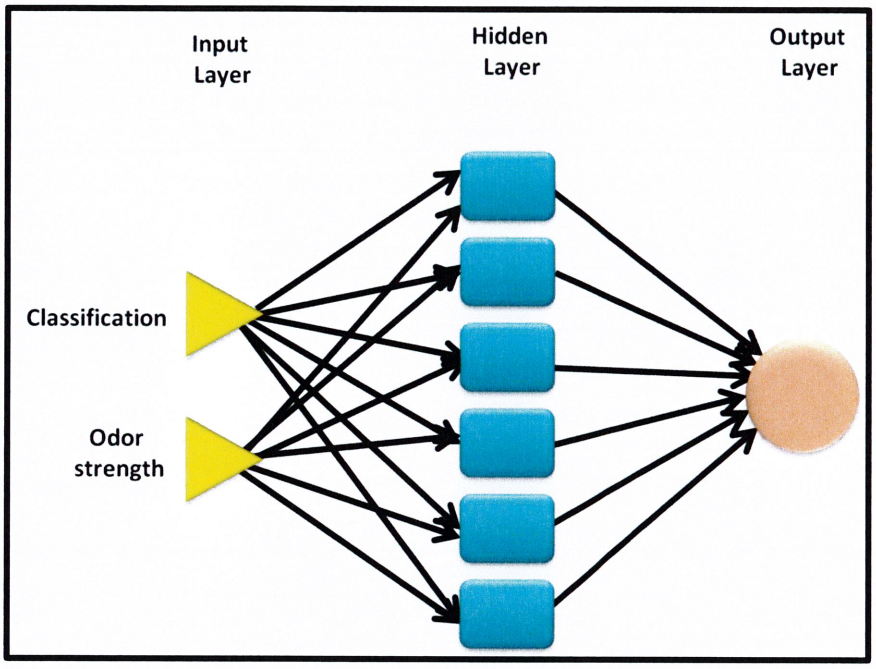


Figure (54)

The Best ANN Architecture is 2:6:1

Table (8)

The Results for Training, Validation, and Testing of ANN

Test Number	ANN Architecture	Least Mean Sum of Square Error		
		Training	Validation	Testing
1	2:2:1	1.95871	0. 91784	0.03%
2	2:3:1	1.90112	0. 88032	0.03%
3	2:4:1	1.96232	0. 88106	0.03%
4	2:5:1	1.90949	0. 88573	0.03%
5	**2:6:1**	**1.72729**	**0. 77849**	**0.03%**
6	2:7:1	1.90356	0. 88150	0.03%
7	2:8:1	1.90300	0.88108	0.03%
8	2:9:1	1.90536	0.88291	0.03%
9	2:10:1	1.90542	0.88225	0.03%
10	2:11:1	1.90437	0.88182	0.03%
11	2:12:1	1.90475	0.88216	0.03%
12	2:13:1	1.905435	0.88228	0.03%
13	2:14:1	1.90431	0.88179	0.03%
14	2:15:1	1.90468	0.88190	0.03%

4.7 Multimedia Elements

Most ANN systems have an output to classify as true or false; in contrast, in the proposed system, when importing data, it will reflect the output with an image formatted as *.png and a sound formatted as *.wav for the feedback of good or bad meat in interesting and user friendly ways, as shown in Figure (55). When the output displays, a *show* button appears, and when the user clicks on it, it will present an image and play a sound. The multimedia elements have been ignored so far; however, these elements are vital when there is a project to implement any application to increase the usability and enhance the output text by adding images and sounds that will enforce the application.



Figure (55)

The Multimedia Elements Added into the Output of the ANN System

Chapter 5: Conclusion

5.1 Discussion.

5.2 Conclusions.

5.3 Future Work.

5.1 Discussion

This study investigated the implementation of a system for the e-nose to detect food poisoning for certain type of meat, that is represented from odour-sensing (OMX-GR sensor), embedded with a classification system that can distinguish between good or bad samples of food. This system is reinforced with multimedia elements, such as images and sounds, which can improve the output display and usability for the user.

The odour-sensing device (OMX-GR sensor) will measure the odours of different kinds of meats (beef, chicken, and beef) in good or fresh condition and in bad or rotten condition. The data was automatically created in a *txt document with the main factors of the ID/classification and the strength of the odour. Then, these data will be prepared in certain conditions to be accepted into the AAN software, as discussed in Section 4.4.

This data was calculated at 14136, which will be fed into the ANN software called Electronic Nose.exe, which starts the training journey. Furthermore, the data is divided into three subsets with the ratio of 2:1:1, in the training set (7068), validation set (3534), and the testing set (3534).

In total, 14 ANN architectures were investigated, and the winner is the 2:6:1 ratio ANN architecture, which scored values of 1.72729 for the training error, 0.77849 for the validation, and 0.03% for testing, which give it the best generalisation based on the least mean of sum of square error (SSE) as shown in Table (8). To evaluate the entire system, we will check the value of validation because it will measure the entire ANN system performance. In contrast, in the training and testing sets, there is a chance to have over-trained, but the training set will stop to solve the problem of over-trained and reduce the validation errors.

When the training phase ends, the testing phase begins; therefore, new data is added to the testing query and output is obtained. In most studies, the output displays as a text, but in our ANN software, it will support the text output with both sounds and images, which gain the software more usability and a user-friendly interface.

The type of ANN used in this study is the backpropagation, which is considered the best tool for training the ANN system. Nevertheless, there are some disadvantages to using this approach, for instance, there are no rules for designing an optimal ANN model. It depends on the problem and requires a trial and error strategy until discovering the optimal ANN model for the problem that needed to be solved. Another issue is the speed; it takes more time when training with the minimum error because it is difficult to find the best parameters for the problems' situations, especially when each problem is singular or unique. (Mas & Flores, 2008).

The ANN is considered a very influential tool. Many studies have been trying to create an optimal ANN system. For example, a study that proposed a system to classify wine in a new way used partial least squared with pattern recognition and registered a successful outcome. The values for the three types of wine were as follows: 97% for young white wine, 87% for young red wines, and 84% for aged red wines (Aguilera, Lozano, Paredes, & Alvarez, 2012).

In 2013, another study developed an electronic system integrated with an ANN to detect many diseases in early stages in the field of clinical diagnoses and obtained great results. Some of these results include the diagnosis of lung cancer at almost 91.9% accurate detection with positive predictive values at 66.6% and negative values at 93.4%. Furthermore, it has 84.4% correct diagnoses for cancer tissues, as well as respiratory diseases, which measured tuberculosis (TB) patients with

87.2% accuracy classification, and urinary tract infections as well (Chen, Wang, & Choi, 2013).

In the same area as the proposed study, another study combined different types of electronic chemical gas sensors to recognise foods, such as grains (more than 80%), cheese (80%), coffee, meat, mushrooms, sugar, fish, beer, and other beverages, and all other foods attained successful results (Schaller, Bosset, & Escher, 1998).

The agriculture and forestry industries also used the e-nose based on the ANN to assess the data analysis in the science field and industries and other forestry applications for log tracking, wood and paper processing, forest management, and more. For example, they obtained correct recognition of 87–95% for four fermented milk types (Wilson, 2013).

The accuracy of the ANN system proposed in this study has a correct recognition at 99.97%, which is a very high rate, which means the proposed ANN worked in an efficient way.

5.2 Conclusions

Over past decades, the e-nose has significantly developed, despite the tendency to ignore the sense of smell in the artificial intelligence techniques. This study proposed a system with an e-nose combined with the ANN, which is considered the core of the system, to process and analyse the data and obtain a recognition pattern in order to obtain the best results similar to the human nose function. Moreover, adding multimedia elements, such as sounds and images, in displaying the output for bad and good food in three types of food, has been examined. The supervised learning used backpropagation, and the best architecture of the ANN was determined to be a 2:6:1 ratio with a 1.72729 training error, 0.77849 validation, and 0.03% testing. Multimedia elements add a creative and interesting ways to display the output.

The proposed system achieved very high performance and a low amount of error, with a high efficiency. Additionally, it acquired low cost tools to achieve the entire system. The e-nose system could be developed in the future to train in an application to sense the good or bad food via a small sensor in smart devises, such as mobile phones.

Moreover, this study elevate a revolutionary tool which can be counted as new added benefit to various multimedia applications by adding an odour sensing element as a new multimedia tool. This function can be similar feature as to the other common ones such as text, image, sound, video and other more.

5.3 Future Work

This study used a specific odour sensor (OMX-GR sensor) to obtain results in three categories of bad or good meat. In the future, another superior sensor could be used to detect the odour in a wide range of foods for classification.

In addition, the type of ANN backpropagation and the parameters of ANN and the activation function used could use other parameters or activation functions, other sorts of ANNs, and other types of learning as unsupervised learning. The ANN program could also be updated to a new version with some enhancements.

Furthermore, in the matter of the challenging question as adding odour elements besides the widespread multimedia elements, could be a subject of research. Of course this part of the research need a special sensor or a combination of sensors and requires further studies on the chemical nature that case the odour and how to convert the nature interaction to an artificial one that can be to an effect to the human sense of smell. Despite the difficulty of this matter, we never can expect the outcome of any discovery might occur in the future or with any similar studies conducted.

References

Aguilera, T., Lozano, J., Paredes, J. A., & Alvarez, F. J. (2012). Electronic Nose Based on Independent Component Analysis. *Sensors* , 12, 8055-8072.

Al-Bastaki, Y. (2009). An Artificial Neural Networks-Based on-Line Monitoring Odor Sensing System. Jour*nal of Computer Science* , 11, 878-882.

Al-Bastaki, Y. (2011). ANN Based Approach to Integrate Smell Sense in Multimedia Systems. *International Journal of Technology Diffusion* , 14-23.

Beale, M. H., Hagan, M. T., & Demuth, H. B. (2015). *Neural Network Toolbox™ User's Guide.* Natick: MathWorks.

Bhadeshia, H. K., Dimitriu, R. C., Forsik, S., Pak, J. H., & Ryu, J. H. (2009). Performance of neural networks in materials science. *Materials Science and Technology* , 25 (4), 504-510.

Carney, J. G., & Cunningham, P. (1998). *The Epoch Interpretation of Learning.* Ireland.

Carraro, S., Rezzi, S., Reniero, F., Héberger, K., Giordano, G., Zanconato, S., et al. (2007). Metabolomics Applied to Exhaled Breath Condensate in Childhood Asthma. *American Journal of Respiratory and Critical Care Medicine* , 175 (10), 986-990.

Castro, R., Mandal, M. K., Ajemba, P., & Istihad, M. A. (2003). An Electronic Nose for Multimedia Applications. *IEEE Transactions on Consumer Electronics* , pp. 1431-1437.

Chakraborty, R. C. (2010). Fundamentals of Neural Networks. Guna: Myreaders.info.

Chen, S., Wang, Y., & Choi, S. (2013). Applications and Technology of Electronic Nose for Clinical Diagnosis. *Open Journal of Applied Biosensor* , 2, 39-50.

Degenhardt, D. C., Greene, J. K., & Khalilian, A. (2012). Temporal Dynamics and Electronic Nose Detection of Stink Bug-Induced Volatile Emissions from Cotton Bolls. *Psyche* , 1-9.

Djarfour, N., Aifa, T., Baddari, K., Mihobi, A., & Farahtia, J. (2008). Application of Feed-back connection Artifical Neural Network to seismic Data filtering. *C.R. Geoscinence* , 1-17.

Emanuele II, V. A., Panicker, G., Gurbaxani, B. M., Lin, J.-M. S., & Unger, E. R. (2012). Sensitive and Specific Peak Detection for SELDI-TOF. *PLOS ONE* , 7 (11), 1-10.

Fausett, L. V. (1994). *Fundamentals of neural network architecture, algorithms, and applications.* UK: Prentice-Hall.

Güngören, Ö. C. (2013). Authentic Learning in Multimedia. *The Online Journal of Distance Education and e-Learning* , 1 (3), 14-19.

Haykin, S. (1994). *Neural Networks A Comprehensive Foundation, Macmillan College Publishing Company,* (1 ed.). USA: Macmillan College Publishing.

He, Q. (1999). *Neural Network and Its Application in IR.* Champaign: UIUCLIS.

Helmy, W., Rivai, M., & Djoko, P. (2012). Neural Network for Electronic Nose using Field Programmable. *International Journal of Electrical and Computer Engineering (IJECE)* , 2, 739-747.

Hoskins, J. C., & Himmelblau, D. M. (1992). Process control via artificial neural networks and reinforcement learning. *Computers & Chemical Engineering* , 16 (4), 241–251.

Hoyo, D. d. (2012). *Optimization of Back-propagation.* Stuttgart: The University of Stuttgart.

Hussain, T. (2009). *Checking the integrity of Global Positioning Recommended Minimum (GPRMC) sentences using Artificial Neural Network (ANN).* Sweden: University of Gavle.

Julian W. Gardner, P. N. (1994). A brief history of electronic noses. *Sensors and Actuators* , 18 (19), 211-220.

Karlik, B., & Bastaki, Y. (2004). Real Time Monitoring Odor Sensing System Using OMX-GR. *WSEAS Transactions on Electronics* , 1 (2), 337-342.

Karlik. B, &. Bastaki, Y. (2004). Bad breath diagnosis system using OMX-GR sensor and Neural Network for telemedicine. *the European Journal of Biomedical Informatics (EJBI)* , 2, 237-239.

Keller, P., Kouzes, R., & Kangas, L. (1994). Three neural network based, sensor systems for environmental monitoring. Boston, MA: IEEE.

Macías, M. M., Agudo, J. E., & Manso, A. G. (2013). A Compact and Low Cost Electronic Nose for Aroma Detection. *Sensors* , 13, 5528-5541.

Mas, J. F., & Flores, J. J. (2008). The application of artificial neural networks to the analysis of remotely sensed data. *International Journal of Remote Sensing* , 29 (3), 617-663.

Persaud, K., & Dodd, G. (1982). Analysis of Discrimination Mechanisms in the Mammalian Olfactory System Using a Model Nose. *Nature* , 299, pp. 352-355.

Rafiq, M. Y., Bugmann, G., & Easterbrook, D. J. (2001). Neural Network Design for Engineering Applications. *International Journal of Computers & Structures* , 79/17, 1541-1552.

Rajthilak, T., & Sasikumar, M. (2008). ARTIFICIAL NEURAL NETWORKS FOR E-NOSE. *Research Journal on Computer Engineering* , 69-73.

Reitermanová, Z. (2010). Data Splitting. *WDS '10 Proceedings of Contributed Papers* , 1, 31–36.

Rojas, R. (1996). *Neural Networks - A Systematic Introduction* (1 ed.). Berlin, New-York: Springer-Verlag.

Schaller, E., Bosset, J. O., & Escher, F. (1998). 'Electronic Noses' and Their Application to Food. pp. 305–316.

Shashi, S., & Minesh, A. B. (1992). *Generalization by Neural Networks.* Minneapolis: IEEE Trans. on Knowledge and Data Eng.

Shiffman, D. (2012). Chapter 10. Neural Networks. In S. Fry (Ed.), *THE NATURE OF CODE.* California: Creative Commons Attribution.

Sibi, P., Jones, S. A., & Siddharth, P. (2013). Analysis Of Different Activation Functions Using Back Propagation Neural Networks. *Journal of Theoretical and Applied Information Technology* , 47, 1264-1268.

Smith, S. W. (1997). Neural Networks (and more!). In *The Scientist and Engineer's Guide to* (pp. 451-480). San Diego: California Technical Publishing.

Srinonchat, J. (2013). Development of Electronic Nose and Program for. *International Journal of Computer and Electrical Engineering* , 5, 61-64.

Stuetz, R. M., Egnin, G., & Fenner, R. A. (1998). Sewage odour measurements using a sensory panel and an electronic nose. *Water Science and Technology* , 38 (3), 331-335.

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. Atlanta, Georgia, USA: JMLR:W&CP.

Wasserman, P. D. (1993). *Advanced Methods in Neural Computing* (1 ed.). New york: Van Nostrand Reinhold.

Wilson, A. D. (2013). Diverse Applications of Electronic-Nose Technologies in. *sensors* , 13, 2295-2348.

Wilson, A. D., & Baietto, M. (2009). Applications and Advances in Electronic-Nose Technologies. pp. 5099–5148.

Appendix.1

This piece of code represent the Network Training.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Text;
using System.Windows.Forms;
using Sinapse.Data;
using Sinapse.Data.Network;


namespace Sinapse.Controls.MainTabControl
{

   internal sealed partial class TabPageTraining :
Sinapse.Controls.MainTabControl.Base.TabPageDataControlBase
   {

      private int m_trainingLayer;


      //---------------------------------------


      #region Constructor
      public TabPageTraining()
      {
         InitializeComponent();
         SetUp(NetworkSet.Training, "Training Set");

         cbTrainingLayer.DataSource = new object[] { "All", 1, 2, 3, 4, 5, };
      }
      #endregion
```

```
//---------------------------------------


#region Control Events
protected override void OnCurrentDatabaseChanged()
{
   base.OnCurrentDatabaseChanged();

   this.setTabPageEnabled(this.NetworkDatabase != null);

   if (this.NetworkDatabase != null)
   {
      DataGridViewColumn column;
      column = new DataGridViewTextBoxColumn();
      column.DataPropertyName = NetworkDatabase.ColumnTrainingLayerId;
      column.HeaderText = "Layer";
      column.DefaultCellStyle.BackColor = SystemColors.Window;
      column.SortMode = DataGridViewColumnSortMode.Automatic;
      column.AutoSizeMode = DataGridViewAutoSizeColumnMode.None;
      this.dataGridView.Columns.Add(column);
   }
}

private void btnShuffle_Click(object sender, EventArgs e)
{
  // this.BindingSource.RemoveSort();
   this.NetworkDatabase.Shuffle(3);
}

#endregion


//---------------------------------------
```

```
#region Private Methods
/*
private void populateCombobox()
{
    cbTrainingSets.BeginUpdate();
    cbTrainingSets.Items.Clear();
    cbTrainingSets.Items.Add("All");
    cbTrainingSets.Items.Add("---");

    if (this.ParentControl != null && this.ParentControl.NetworkData != null)
    {
        for (ushort i = 1; i <= this.ParentControl.NetworkData.TrainingLayerCount; ++i)
        {
            cbTrainingSets.Items.Add(i);
        }
    }

    cbTrainingSets.EndUpdate();
    cbTrainingSets.SelectedIndex = 0;

}
*/
private void cbTrainingSets_SelectedIndexChanged(object sender, EventArgs e)
{
    if (cbTrainingLayer.SelectedIndex == 0)
    {
        this.m_trainingLayer = 0;
        this.BindingSource.Filter = this.GetFilterString();
    }
    else
    {
        this.m_trainingLayer = cbTrainingLayer.SelectedIndex;
        this.BindingSource.Filter = String.Format("{0} AND {1} = '{2}'",
            this.GetFilterString(),
            NetworkDatabase.ColumnTrainingLayerId,
            this.m_trainingLayer);
    }
```

```
        }
        #endregion


        //---------------------------------------


        protected override void OnRowValidating(DataRowView row)
        {
            if (m_trainingLayer > 0)
                row.Row[NetworkDatabase.ColumnTrainingLayerId] = m_trainingLayer;

            base.OnRowValidating(row);
        }

        protected override void OnDataImported(DataTable table)
        {
            this.NetworkDatabase.ImportData(table, NetworkSet.Training, m_trainingLayer);
        }


        //---------------------------------------


    }
}
```

Appendix.2

This piece of code represent the Network Testing.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Text;
using System.Windows.Forms;

using Sinapse.Data;
using Sinapse.Data.Network;
using Sinapse.Forms.Dialogs;


namespace Sinapse.Controls.MainTabControl
{

   internal sealed partial class TabPageTesting :
Sinapse.Controls.MainTabControl.Base.TabPageDataControlBase
   {

      //---------------------------------------


      #region Constructor
      public TabPageTesting()
      {
         InitializeComponent();
         SetUp(NetworkSet.Testing, "Testing Set");
      }
      #endregion


      //---------------------------------------


      #region Properties
```

```
#endregion


//--------------------------------------


#region Control Events
protected override void OnLoad(EventArgs e)
{
   base.OnLoad(e);
}

protected override void OnCurrentDatabaseChanged()
{
   base.OnCurrentDatabaseChanged();

   this.setTabPageEnabled(this.NetworkDatabase != null);

   if (this.NetworkDatabase != null)
   {
    //   DataGridViewColumn column;

      foreach (String colName in this.NetworkDatabase.Schema.OutputColumns)
      {

         this.dataGridView.Columns.Add(this.CreateColumn(
            colName + " (network)",
            NetworkDatabase.ColumnComputedPrefix + colName,
            panelNetworkCaption.BackColor));

         this.dataGridView.Columns.Add(this.CreateColumn(
            colName + " (deviation)",
            NetworkDatabase.ColumnDeltaPrefix + colName,
            panelDeltaCaption.BackColor));

      }
   }
}

protected override void OnCurrentNetworkChanged()
```

```
{
    base.OnCurrentNetworkChanged();

    btnCompare.Enabled = (this.NetworkContainer != null);
    btnQuery.Enabled = (this.NetworkContainer != null);
}
#endregion


//---------------------------------------


#region Buttons
private void btnQuery_Click(object sender, EventArgs e)
{
    this.Compute();
}

private void btnCompare_Click(object sender, EventArgs e)
{
    this.Compare();
}

private void btnRound_Click(object sender, EventArgs e)
{
    this.EndEdit();
    this.NetworkDatabase.Round(true);
}
#endregion


//---------------------------------------


#region Public Methods
public void Compute()
{
    this.EndEdit();
    this.NetworkDatabase.ComputeTable(this.NetworkContainer, true);
}
```

```
        public void Compare()
        {
            new NetworkReportDialog(this.NetworkContainer, this.NetworkDatabase).ShowDialog(this);
        }
        #endregion


        //----------------------------------------


        #region Private Methods



        #endregion


    }
}
```

Appendix.3

This piece of code represent the Network Validation.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Text;
using System.Windows.Forms;

using Sinapse.Data;
using Sinapse.Data.Network;


namespace Sinapse.Controls.MainTabControl
{

   internal sealed partial class TabPageValidation :
Sinapse.Controls.MainTabControl.Base.TabPageDataControlBase
   {


      //--------------------------------------


      #region Constructor
      public TabPageValidation()
      {
         InitializeComponent();
         SetUp(NetworkSet.Validation, "Validation Set");
      }
      #endregion


      //--------------------------------------


      #region Control Events
```

```
        protected override void OnCurrentDatabaseChanged()
        {
            base.OnCurrentDatabaseChanged();

            this.setTabPageEnabled(this.NetworkDatabase != null);
        }
        #endregion


        //---------------------------------------


    }
}
```

Appendix.4

This piece of code represent the Network Query.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

using Sinapse.Data;
using Sinapse.Data.Network;


namespace Sinapse.Controls.MainTabControl
{
   internal partial class TabPageQuery :
Sinapse.Controls.MainTabControl.Base.TabPageDataControlBase
   {

      //---------------------------------------


      #region Constructor
      public TabPageQuery()
      {
         InitializeComponent();
         SetUp(NetworkSet.Query, "Network Query");
      }
      #endregion


      //---------------------------------------


      #region Control Events
      protected override void OnCurrentNetworkChanged()
      {
```

```
    base.OnCurrentNetworkChanged();

    this.setTabPageEnabled(this.NetworkContainer != null);

    if (this.NetworkContainer != null)
    {
       this.NetworkDatabase = new NetworkDatabase(this.NetworkContainer.Schema);
    //   this.BindingSource.DataSource = m_networkDatabase;
    }
    else
    {
       this.NetworkDatabase = null;
    //   this.BindingSource.DataSource = null;
    }
}

#endregion


//---------------------------------------


#region Buttons
private void btnQuery_Click(object sender, EventArgs e)
{
   if (this.dataGridView.ColumnCount <= 3)
   {
      DataGridViewImageColumn oImageColumn = new DataGridViewImageColumn();
      oImageColumn.Name = "Image";
      oImageColumn.Visible = false;
      oImageColumn.ImageLayout = DataGridViewImageCellLayout.Normal;
      this.dataGridView.Columns.Add(oImageColumn);
      this.dataGridView.Columns["Image"].DataPropertyName = "Image";

      DataGridViewButtonColumn oButtonColumn = new DataGridViewButtonColumn();
      oButtonColumn.Name = "Audio";
      oButtonColumn.Visible = false;
      oButtonColumn.Text = "Play";
      oButtonColumn.UseColumnTextForButtonValue = true;
      this.dataGridView.Columns.Add(oButtonColumn);
```

```
            DataGridViewTextBoxColumn oAudioColumn = new DataGridViewTextBoxColumn();
            oAudioColumn.Visible = false;
            oAudioColumn.Name = "Play";
            oAudioColumn.DataPropertyName = "Audio";
            this.dataGridView.Columns.Add(oAudioColumn);

            DataGridViewButtonColumn oButtonColumn1 = new DataGridViewButtonColumn();
            oButtonColumn1.Name = "ANN Decision";
            oButtonColumn1.Text = "Show";
            oButtonColumn1.UseColumnTextForButtonValue = true;
            this.dataGridView.Columns.Add(oButtonColumn1);
        }

        this.EndEdit();

        this.NetworkDatabase.ComputeTable(this.NetworkContainer, false);
    }

    private void btnRound_Click(object sender, EventArgs e)
    {
        this.EndEdit();

        this.NetworkDatabase.Round(false);
    }
    #endregion


    //--------------------------------------


}
}
```

Appendix.5

This piece of code represent the Learning Graph.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Text;
using System.Windows.Forms;

using ZedGraph;

using Sinapse.Data;
using Sinapse.Data.Network;
using Sinapse.Forms.Dialogs;


namespace Sinapse.Controls.MainTabControl
{
   internal sealed partial class TabPageLearningGraph : Sinapse.Controls.Base.TabPageControlBase
   {

      private NetworkContainer m_networkContainer;
      private IPointListEdit m_trainingPoints;
      private IPointListEdit m_validationPoints;
      private IPointListEdit m_currentSavepoint;
      private IPointListEdit m_savePoints;

//      private bool newSavepoint;

      private SideTabControl.SidePageTrainer m_networkTrainer;


      //---------------------------------------


      #region Constructor
      internal TabPageLearningGraph()
```

```
    {
        InitializeComponent();

        this.SetStyle(ControlStyles.AllPaintingInWmPaint |
            ControlStyles.UserPaint |
            ControlStyles.OptimizedDoubleBuffer,
            true);


        this.TabPageName = "Training History";
        this.dataGridView.AutoGenerateColumns = false;
        this.CreateChart(zedGraphControl);
    }
    #endregion


    //--------------------------------------


    #region Properties
    [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
    internal NetworkContainer NetworkContainer
    {
        get { return this.m_networkContainer; }
        set
        {
            this.m_networkContainer = value;

            if (value != null && this.NetworkTrainer.NetworkDatabase != null)
            {
                this.dataGridView.DataSource = this.m_networkContainer.Savepoints;
                this.m_networkContainer.Savepoints.SavepointRegistered +=
networkContainer_savepointRegistered;
                this.m_networkContainer.Savepoints.SavepointRestored +=
networkContainer_savepointRestored;
                this.m_networkContainer.Savepoints.CurrentSavepointChanged +=
networkContainer_savepointChanged;
            }
            else
            {
```

```
            this.dataGridView.DataSource = null;
        }

        this.UpdateEnabled();
    }
}

[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
internal SideTabControl.SidePageTrainer NetworkTrainer
{
    get { return this.m_networkTrainer; }
    set
    {
        this.m_networkTrainer = value;

        if (value != null)
        {
        //    this.m_networkTrainer.TrainingStarted += networkTrainer_TrainingStarted;
        }
    }
}

[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
internal IPointListEdit TrainingPoints
{
    get { return this.m_trainingPoints; }
}

[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
internal IPointListEdit ValidationPoints
{
    get { return this.m_validationPoints; }
}

[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
internal IPointListEdit SavePoints
{
    get { return this.m_savePoints; }
}
#endregion
```

```
//--------------------------------------


#region Control Events
protected override void OnLoad(EventArgs e)
{
   base.OnLoad(e);
}

private void trackBar_Scroll(object sender, EventArgs e)
{
   Properties.Settings.Default.training_delay = this.trackBar.Value*10;
}
#endregion


#region Graph Events
private void zedGraphControl_DoubleClick(object sender, EventArgs e)
{
   if (!GraphOptionsDialog.HasInstance)
   {
      new GraphOptionsDialog().Show(this);
   }
}
#endregion


#region Object Events
private void networkContainer_savepointRegistered(object sender, EventArgs e)
{
   this.SavePoints.Add(this.m_networkContainer.Savepoints.CurrentSavepoint.Epoch,
      this.m_networkContainer.Savepoints.CurrentSavepoint.ErrorTraining);

   this.SavePoints.Add(this.m_networkContainer.Savepoints.CurrentSavepoint.Epoch,
      this.m_networkContainer.Savepoints.CurrentSavepoint.ErrorValidation);

   HistoryListener.Write("Savepoint Registered!");
}
```

```
        private void networkContainer_savepointRestored(object sender, EventArgs e)
        {
    //      this.newSavepoint = true;

            HistoryListener.Write("Savepoint Restored!");
        }

        private void networkContainer_savepointChanged(object sender, EventArgs e)
        {
        }
        #endregion


        #region DataGridView Events
        private void dataGridView_CellMouseDoubleClick(object sender,
DataGridViewCellMouseEventArgs e)
        {
            if (e.ColumnIndex >= 0 && e.RowIndex >= 0)
            {
                NetworkSavepoint save = (this.dataGridView.Rows[e.RowIndex].DataBoundItem as
NetworkSavepoint);

                if (save != null)
                    this.m_networkContainer.Savepoints.Restore(save);
            }
        }

        private void dataGridView_CurrentCellChanged(object sender, EventArgs e)
        {
            this.m_currentSavepoint.Clear();

            if (dataGridView.CurrentRow != null)
            {
                NetworkSavepoint sp = dataGridView.CurrentRow.DataBoundItem as NetworkSavepoint;

                if (sp != null)
                {
                    this.m_currentSavepoint.Add(sp.Epoch, sp.ErrorTraining);
                    this.m_currentSavepoint.Add(sp.Epoch, sp.ErrorValidation);
```

```
            }

            this.UpdateGraph();
        }
    }

    private void dataGridView_DataBindingComplete(object sender,
DataGridViewBindingCompleteEventArgs e)
    {
        NetworkSavepoint currSavepoint = this.m_networkContainer.Savepoints.CurrentSavepoint;
        NetworkSavepoint bestSavepoint = this.m_networkContainer.Savepoints.BestSavepoint;

        foreach (DataGridViewRow row in this.dataGridView.Rows)
        {
            if ((NetworkSavepoint)row.DataBoundItem == currSavepoint)
            {
                row.DefaultCellStyle.BackColor = Color.AliceBlue;
            }
            if ((NetworkSavepoint)row.DataBoundItem == bestSavepoint)
            {
                row.DefaultCellStyle.BackColor = Color.Honeydew;
            }
            else
            {
                row.DefaultCellStyle.BackColor = Color.White;
            }
        }

        if (Properties.Settings.Default.graph_autoScrollSavepoints)
        {

            if (dataGridView.Rows.Count > 0)
            {
                dataGridView.FirstDisplayedCell = dataGridView.Rows[dataGridView.Rows.Count -
1].Cells[0];
            }
        }
    }
    #endregion
```

```
//--------------------------------------


#region Buttons
private void btnGraphSave_Click(object sender, EventArgs e)
{
   this.zedGraphControl.SaveAs();
}

private void btnUpdate_Click(object sender, EventArgs e)
{
   this.UpdateGraph();
}

private void btnClear_Click(object sender, EventArgs e)
{
   this.ClearGraph();
}

private void btnTrim_Click(object sender, EventArgs e)
{
   if (this.dataGridView.CurrentRow != null)
   {
      NetworkSavepoint save = (this.dataGridView.CurrentRow.DataBoundItem as
NetworkSavepoint);

      if (save != null)
         this.TrimGraph(save.Epoch);
   }
}


private void btnSavepointLoad_Click(object sender, EventArgs e)
{
   if (this.dataGridView.CurrentRow != null)
   {
      NetworkSavepoint save = (this.dataGridView.CurrentRow.DataBoundItem as
NetworkSavepoint);
```

```
        if (save != null)
            this.m_networkContainer.Savepoints.Restore(save);
    }
}

private void btnSavepointMark_Click(object sender, EventArgs e)
{
    this.m_networkContainer.Savepoints.Register(m_networkTrainer.NetworkState);
}

private void btnSavepointClear_Click(object sender, EventArgs e)
{
    this.m_networkContainer.Savepoints.Clear();
}
#endregion


//---------------------------------------


#region Public Methods
public void UpdateGraph()
{
    this.zedGraphControl.AxisChange();
    this.zedGraphControl.Invalidate();
    this.Invalidate();
}

public void ClearGraph()
{
    this.TrainingPoints.Clear();
    this.ValidationPoints.Clear();
    this.SavePoints.Clear();
    this.UpdateGraph();
}


public void TrimGraph(int startEpoch)
{
    lock (this)
```

```
    {
      if (this.m_trainingPoints.Count > 0)
      {
        while (this.m_trainingPoints[this.m_trainingPoints.Count - 1].X > startEpoch)
        {
          this.m_trainingPoints.RemoveAt(this.m_trainingPoints.Count - 1);
        }
      }

      if (this.m_validationPoints.Count > 0)
      {
        while (this.m_validationPoints[this.m_validationPoints.Count - 1].X > startEpoch)
        {
          this.m_validationPoints.RemoveAt(this.m_validationPoints.Count - 1);
        }
      }

      if (this.m_savePoints.Count > 0)
      {
        while (this.m_savePoints[this.m_savePoints.Count - 1].X > startEpoch)
        {
          this.m_savePoints.RemoveAt(this.m_savePoints.Count - 1);
        }
      }

      this.UpdateGraph();
    }
  }

  public void UpdateEnabled()
  {
    if (m_networkContainer != null && this.NetworkTrainer.NetworkDatabase != null)
      this.setTabPageEnabled(true);
    else this.setTabPageEnabled(false);
  }

  public void SavepointNext()
  {
  }
```

```
        public void SavepointPrev()
        {
        }
        #endregion


        //---------------------------------------


        #region Private Methods
        private void CreateChart(ZedGraphControl zgc)
        {

            GraphPane myPane = zgc.GraphPane;

            // Set the title and axis labels
            myPane.Title.Text = "Training Graph";
            myPane.XAxis.Title.Text = "Epochs";
            myPane.YAxis.Title.Text = "Least-Mean-Square Error";

            RollingPointPairList trainingList = new RollingPointPairList(5000);
            RollingPointPairList validationList = new RollingPointPairList(5000);
            RollingPointPairList savepointList = new RollingPointPairList(5000);
            RollingPointPairList currentSavepointList = new RollingPointPairList(5000);

            // Add a curve
            LineItem curve;

            curve = myPane.AddCurve("Training Set", trainingList, Color.Red, SymbolType.Diamond);
            curve.Symbol.Fill = new Fill(Color.White);
            curve.Symbol.Size = 4;
            this.m_trainingPoints = curve.Points as IPointListEdit;
            // trainingCurve.Line.IsSmooth = true;
            // trainingCurve.Line.SmoothTension = 0.5F;


            curve = myPane.AddCurve("Validation Set", validationList, Color.Blue,
SymbolType.Diamond);
            curve.Symbol.Fill = new Fill(Color.White);
            curve.Symbol.Size = 4;
```

```
            this.m_validationPoints = curve.Points as IPointListEdit;
            // validationCurve.Line.IsSmooth = true;
            // validationCurve.Line.SmoothTension = 0.5F;


            curve = myPane.AddCurve("Savepoints Mark", savepointList, Color.DarkGreen,
SymbolType.Circle);
            curve.Symbol.Fill = new Fill(Color.DarkGreen);
            curve.Symbol.Size = 4;
            curve.Line.IsVisible = false;
            this.m_savePoints = curve.Points as IPointListEdit;
            // validationCurve.Line.IsSmooth = true;
            // validationCurve.Line.SmoothTension = 0.5F;

            curve = myPane.AddCurve(String.Empty, currentSavepointList, Color.LimeGreen,
SymbolType.Circle);
            curve.Symbol.Fill = new Fill(Color.LimeGreen);
            curve.Symbol.Size = 8;
            curve.Line.IsVisible = false;
            this.m_currentSavepoint = curve.Points as IPointListEdit;
            // validationCurve.Line.IsSmooth = true;
            // validationCurve.Line.SmoothTension = 0.5F;

            myPane.XAxis.Scale.MinAuto = true;
            myPane.XAxis.Scale.MaxAuto = true;
            myPane.YAxis.Scale.MinAuto = true;
            myPane.YAxis.Scale.MaxAuto = true;
            myPane.XAxis.Scale.MagAuto = true;
            myPane.YAxis.Scale.MagAuto = true;

            // Fill the axis background with a color gradient
            myPane.Chart.Fill = new Fill(Color.White, Color.LightGray, 45.0F);

            // Calculate the Axis Scale Ranges
            zgc.AxisChange();
        }
        #endregion


        //---------------------------------------
```

```
    }
}
```

Appendix.6

This piece of code represent the Import Wizard.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;

using WizardBase;

using Sinapse.Data.Network;
using Sinapse.Utils.CsvParser;


namespace Sinapse.Forms.Dialogs
{

   internal sealed partial class ImportWizard : Form
   {

      private DataTable m_dataTable;
      private NetworkSchema m_networkSchema;

      //----------------------------------------


      #region Constructor
      public ImportWizard()
      {
         InitializeComponent();
         cbDelimiter.DataSource = Enum.GetValues(typeof(CsvDelimiter));
      }
      #endregion
```

```
        //-----------------------------------------


        #region Public Methods
        internal NetworkDatabase GetNetworkData()
        {
            return new NetworkDatabase(m_networkSchema, m_dataTable);
        }

        internal DataTable GetDataTable()
        {
            return m_dataTable;
        }

        internal NetworkSchema GetSchema()
        {
            return m_networkSchema;
        }
        #endregion


        //-----------------------------------------


        #region Wizard Events
        private void wizardControl_NextButtonClick(WizardControl sender,
WizardNextButtonClickEventArgs e)
        {
            switch (wizardControl.CurrentStepIndex)
            {
                case 0:
                    m_dataTable = null;
                    break;

                case 1:
                    if (File.Exists(tbDatapath.Text) && loadDataTable())
                    {
                        loadInputCombobox();
                    }
                    else
```

```
            {
                e.Cancel = true;
            }
            break;

        case 2:
            loadOutputCombobox();
            break;

        case 3:
            loadStringCombobox();
            break;

        default:
            break;
    }
}


private void wizardControl_CancelButtonClick(object sender, EventArgs e)
{
    this.m_dataTable = null;
    this.Close();
}


private void wizardControl_FinishButtonClick(object sender, EventArgs e)
{

    List<String> inputColumns = new List<String>();
    foreach (string strColumn in clbInput.CheckedItems)
        inputColumns.Add(strColumn);

    List<String> outputColumns = new List<String>();
    foreach (string strColumn in clbOutput.CheckedItems)
        outputColumns.Add(strColumn);

    List<String> stringColumns = new List<String>();
    foreach (string strColumn in clbString.CheckedItems)
        stringColumns.Add(strColumn);
```

```
        //Remove unusable columns
        List<DataColumn> removeCols = new List<DataColumn>();
        foreach (DataColumn col in m_dataTable.Columns)
        {
            if (!inputColumns.Contains(col.ColumnName) &&
                !outputColumns.Contains(col.ColumnName))
                removeCols.Add(col);
        }

        foreach (DataColumn col in removeCols)
            m_dataTable.Columns.Remove(col);


        //Create Network Schema
        this.m_networkSchema = new NetworkSchema(
            inputColumns.ToArray(),
            outputColumns.ToArray(),
            stringColumns.ToArray());

        this.DialogResult = DialogResult.OK;
        this.Close();
    }
    #endregion


    //-----------------------------------------


    #region Load Controls
    private bool loadDataTable()
    {
        try
        {
            this.m_dataTable = CsvParser.Parse(tbDatapath.Text, Encoding.Default, true,
(CsvDelimiter)cbDelimiter.SelectedValue);
        }
        catch (IOException e)
        {
```

```
            MessageBox.Show(e.Message, "Error opening file. Please ensure no other process is using
that file and try again.");
#if DEBUG
            throw;
#else
            return false;
#endif
        }
        return true;
    }

    private void loadInputCombobox()
    {
        clbInput.Items.Clear();
        //      clbString.Items.Clear();
        foreach (DataColumn col in m_dataTable.Columns)
        {
            clbInput.Items.Add(col.ColumnName, false);
            //          clbString.Items.Add(col.ColumnName, false);
        }
    }

    private void loadOutputCombobox()
    {
        clbOutput.Items.Clear();
        foreach (DataColumn col in m_dataTable.Columns)
        {
            if (!clbInput.CheckedItems.Contains(col.ColumnName))
                clbOutput.Items.Add(col.ColumnName, false);
        }
    }

    private void loadStringCombobox()
    {
        clbString.Items.Clear();
        foreach (DataColumn col in m_dataTable.Columns)
        {
            if (clbInput.CheckedItems.Contains(col.ColumnName) ||
                clbOutput.CheckedItems.Contains(col.ColumnName))
            {
```

```
            clbString.Items.Add(col.ColumnName, false);
        }
    }
}
#endregion


//-----------------------------------------


#region Buttons
private void btnBrowse_Click(object sender, EventArgs e)
{
    openFileDialog.ShowDialog(this);
}

private void openFileDialog_FileOk(object sender, CancelEventArgs e)
{
    tbDatapath.Text = openFileDialog.FileName;
    this.autoDetectFieldDelimiter(tbDatapath.Text);
}

private void btnAutodetect_Click(object sender, EventArgs e)
{
    if (tbDatapath.Text.Length > 0 && File.Exists(tbDatapath.Text))
    {
        this.autoDetectFieldDelimiter(tbDatapath.Text);
    }
    else
    {
        MessageBox.Show("Invalid path specified");
    }
}
#endregion


//-----------------------------------------
```

```
        #region Private Methods
        private void autoDetectFieldDelimiter(string path)
        {
            cbDelimiter.SelectedItem = CsvUtils.DetectFieldDelimiterChar(tbDatapath.Text,
Encoding.Default);
        }
        #endregion

    }
}
```

Appendix.7

This piece of code represent the meat decision based on neural network output.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.IO;
using System.Windows.Forms;

namespace Sinapse.Forms.Dialogs
{
  public partial class frmMeatDecisionDialog : Form
  {
    string sMediaFile = "";
    Image  oANNDecisionFileStream = null;
    string sMainDirectory = Application.StartupPath;
    string sDataFolder = "\\Data";

    public frmMeatDecisionDialog(Image ImageFileStream, string sWavFile)
    {
      InitializeComponent();
      oANNDecisionFileStream = ImageFileStream;
      sMediaFile = sWavFile;

      picBox.Image = ImageFileStream;
      picBox.Refresh();
    }

    private void btnPlay_Click(object sender, EventArgs e)
    {
      // Check if file exists
      if (Directory.Exists(sMainDirectory + sDataFolder))
      {
        if (File.Exists(sMainDirectory + sDataFolder + sMediaFile))
```

```
{
    this.network = network;

    // create error and deltas arrays
    neuronErrors = new double[network.LayersCount][];
    weightsUpdates = new double[network.LayersCount][][];
    thresholdsUpdates = new double[network.LayersCount][];

    // initialize errors and deltas arrays for each layer
    for (int i = 0, n = network.LayersCount; i < n; i++)
    {
        Layer layer = network[i];

        neuronErrors[i] = new double[layer.NeuronsCount];
        weightsUpdates[i] = new double[layer.NeuronsCount][];
        thresholdsUpdates[i] = new double[layer.NeuronsCount];

        // for each neuron
        for (int j = 0; j < layer.NeuronsCount; j++)
        {
            weightsUpdates[i][j] = new double[layer.InputsCount];
        }
    }
}

/// <summary>
/// Runs learning iteration
/// </summary>
///
/// <param name="input">input vector</param>
/// <param name="output">desired output vector</param>
///
/// <returns>Returns squared error of the last layer divided by 2</returns>
///
/// <remarks>Runs one learning iteration and updates neuron's
/// weights.</remarks>
///
public double Run(double[] input, double[] output)
{
    // compute the network's output
```

```
    network.Compute(input);

    // calculate network error
    double error = CalculateError(output);

    // calculate weights updates
    CalculateUpdates(input);

    // update the network
    UpdateNetwork();

    return error;
}

public double MeasureError(double[] input, double[] output)
{
    // compute the network's output
    network.Compute(input);

    // calculate network error
    double error = CalculateError(output);

    return error;
}

public double MeasureEpochError(double[][] input, double[][] output)
{
    double error = 0.0;

    for (int i = 0; i < input.Length; ++i)
    {
        error += MeasureError(input[i], output[i]);
    }

    return error;
}

/// <summary>
/// Runs learning epoch
/// </summary>
```

```
///
/// <param name="input">array of input vectors</param>
/// <param name="output">array of output vectors</param>
///
/// <returns>Returns sum of squared errors of the last layer divided by 2</returns>
///
/// <remarks>Runs series of learning iterations - one iteration
/// for each input sample. Updates neuron's weights after each sample
/// presented.</remarks>
///
public double RunEpoch(double[][] input, double[][] output)
{
    double error = 0.0;

    // run learning procedure for all samples
    for (int i = 0, n = input.Length; i < n; i++)
    {
        error += Run(input[i], output[i]);
    }

    // return summary error
    return error;
}



/// <summary>
/// Calculates error values for all neurons of the network
/// </summary>
///
/// <param name="desiredOutput">Desired output vector</param>
///
/// <returns>Returns summary squared error of the last layer divided by 2</returns>
///
private double CalculateError(double[] desiredOutput)
{
    // current and the next layers
    ActivationLayer layer, layerNext;
    // current and the next errors arrays
    double[] errors, errorsNext;
    // error values
```

```
            {
                System.Media.SoundPlayer player = new System.Media.SoundPlayer(sMainDirectory +
sDataFolder + sMediaFile);
                player.Play();
            }
        }
    }
}
```

Appendix.8

This piece of code represent the activation network which is part of create the ANN architecture with activation functions.

```
namespace AForge.Neuro
{
  using System;

  /// <summary>
  /// Activation network
  /// </summary>
  ///
  /// <remarks>Activation network is a base for multi-layer neural network
  /// with activation functions. It consists of <see cref="ActivationLayer">activation
  /// layers</see>.</remarks>
  ///
  [Serializable]
  public class ActivationNetwork : Network
  {
    /// <summary>
    /// Network's layers accessor
    /// </summary>
    ///
    /// <param name="index">Layer index</param>
    ///
    /// <remarks>Allows to access network's layer.</remarks>
    ///
    public new ActivationLayer this[int index]
    {
      get { return ((ActivationLayer)layers[index]); }
    }

    /// <summary>
    /// Initializes a new instance of the <see cref="ActivationNetwork"/> class
    /// </summary>
    /// <param name="function">Activation function of neurons of the network</param>
    /// <param name="inputsCount">Network's inputs count</param>
```

```
        /// <param name="neuronsCount">Array, which specifies the amount of neurons in
        /// each layer of the neural network</param>
        ///
        /// <remarks>The new network will be randomized (see <see
    cref="ActivationNeuron.Randomize"/>
        /// method) after it is created.</remarks>
        ///
        /// <example>The following sample illustrates the usage of <c>ActivationNetwork</c> class:
        /// <code>
        ///             // create activation network
        ///             ActivationNetwork network = new ActivationNetwork(
        ///                     new SigmoidFunction( ), // sigmoid activation function
        ///                     3,              // 3 inputs
        ///                     4, 1 );         // 2 layers:
        ///                      // 4 neurons in the first layer
        ///                      // 1 neuron in the second layer
        /// </code>
        /// </example>
        ///
        public ActivationNetwork(IActivationFunction function, int inputsCount, params int[]
    neuronsCount)
            : base(inputsCount, neuronsCount.Length)
        {
            // create each layer
            for (int i = 0; i < layersCount; i++)
            {
                layers[i] = new ActivationLayer(
                    // neurons count in the layer
                    neuronsCount[i],
                    // inputs count of the layer
                    (i == 0) ? inputsCount : neuronsCount[i - 1],
                    // activation function of the layer
                    function);
            }
        }
    }
}
```

Appendix.9

This piece of code represent the creation of layers which is part of construct the ANN architecture which used in multi -layer.

```
namespace AForge.Neuro
{
    using System;

    /// <summary>
    /// Activation layer
    /// </summary>
    ///
    /// <remarks>Activation layer is a layer of <see cref="ActivationNeuron">activation
neurons</see>.
    /// The layer is usually used in multi-layer neural networks.</remarks>
    ///
  [Serializable]
    public class ActivationLayer : Layer
    {
        /// <summary>
        /// Layer's neurons accessor
        /// </summary>
        ///
        /// <param name="index">Neuron index</param>
        ///
        /// <remarks>Allows to access layer's neurons.</remarks>
        ///
        public new ActivationNeuron this[int index]
        {
            get { return (ActivationNeuron) neurons[index]; }
        }


        /// <summary>
        /// Initializes a new instance of the <see cref="ActivationLayer"/> class
        /// </summary>
        /// <param name="neuronsCount">Layer's neurons count</param>
        /// <param name="inputsCount">Layer's inputs count</param>
```

```
        /// <param name="function">Activation function of neurons of the layer</param>
        ///
        /// <remarks>The new layet will be randomized (see <see
cref="ActivationNeuron.Randomize"/>
        /// method) after it is created.</remarks>
        ///
        public ActivationLayer( int neuronsCount, int inputsCount, IActivationFunction
function )
                                                        : base( neuronsCount, inputsCount )
        {
            // create each neuron
            for ( int i = 0; i < neuronsCount; i++ )
                neurons[i] = new ActivationNeuron( inputsCount, function );
        }
    }
}
```

Appendix.10

This piece of code represent the activation neuron, by adding threshold value and then applies activation function which is used multi-layer network .

```
namespace AForge.Neuro
{
  using System;

  /// <summary>
  /// Activation neuron
  /// </summary>
  ///
  /// <remarks>Activation neuron computes weighted sum of its inputs, adds
  /// threshold value and then applies activation function. The neuron is
  /// usually used in multi-layer neural networks.</remarks>
  ///
  [Serializable]
  public class ActivationNeuron : Neuron
  {
    /// <summary>
    /// Threshold value
    /// </summary>
    ///
    /// <remarks>The value is added to inputs weighted sum.</remarks>
    ///
    protected double threshold = 0.0f;

    /// <summary>
    /// Activation function
    /// </summary>
    ///
    /// <remarks>The function is applied to inputs weighted sum plus
    /// threshold value.</remarks>
    ///
    protected IActivationFunction function = null;
```

```
/// <summary>
/// Threshold value
/// </summary>
///
/// <remarks>The value is added to inputs weighted sum.</remarks>
///
public double Threshold
{
  get { return threshold; }
  set { threshold = value; }
}

/// <summary>
/// Neuron's activation function
/// </summary>
///
public IActivationFunction ActivationFunction
{
  get { return function; }
}

/// <summary>
/// Initializes a new instance of the <see cref="ActivationNeuron"/> class
/// </summary>
///
/// <param name="inputs">Neuron's inputs count</param>
/// <param name="function">Neuron's activation function</param>
///
public ActivationNeuron(int inputs, IActivationFunction function)
  : base(inputs)
{
  this.function = function;
}

/// <summary>
/// Randomize neuron
/// </summary>
///
/// <remarks>Calls base class <see cref="Neuron.Randomize">Randomize</see> method
/// to randomize neuron's weights and then randomize threshold's value.</remarks>
```

```
///
public override void Randomize()
{
  // randomize weights
  base.Randomize();
  // randomize threshold
  threshold = rand.NextDouble() * (randRange.Length) + randRange.Min;
}

/// <summary>
/// Computes output value of neuron
/// </summary>
///
/// <param name="input">Input vector</param>
///
/// <returns>Returns neuron's output value</returns>
///
/// <remarks>The output value of activation neuron is equal to value
/// of nueron's activation function, which parameter is weighted sum
/// of its inputs plus threshold value. The output value is also stored
/// in <see cref="Neuron.Output">Output</see> property.</remarks>
///
public override double Compute(double[] input)
{
  // check for corrent input vector
  if (input.Length != inputsCount)
    throw new ArgumentException();

  // initial sum value
  double sum = 0.0;

  // compute weighted sum of inputs
  for (int i = 0; i < inputsCount; i++)
  {
    sum += weights[i] * input[i];
  }
  sum += threshold;

  return (output = function.Function(sum));
}
```

Appendix.11

This piece of code represent the sigmoid activation function which is used in the system.

```
namespace AForge.Neuro
{
    using System;

    /// <summary>
    /// Sigmoid activation function
    /// </summary>
    ///
    /// <remarks>The class represents sigmoid activation function with
    /// the next expression:<br />
    /// <code>
    ///                 1
    /// f(x) = ------------------
    ///        1 + exp(-alpha * x)
    ///
    ///           alpha * exp(-alpha * x )
    /// f'(x) = ---------------------------- = alpha * f(x) * (1 - f(x))
    ///           (1 + exp(-alpha * x))^2
    /// </code>
    /// Output range of the function: <b>[0, 1]</b><br /><br />
    /// Functions graph:<br />
    /// <img src="sigmoid.bmp" width="242" height="172" />
    /// </remarks>
  [Serializable]
    public class SigmoidFunction : IActivationFunction
    {
        // sigmoid's alpha value
        private double alpha = 2;

        /// <summary>
        /// Sigmoid's alpha value
        /// </summary>
        ///
        /// <remarks>The value determines steepness of the function. Default value:
<b>2</b>.
```

```
/// </remarks>
public double Alpha
{
        get { return alpha; }
        set { alpha = value; }
}

/// <summary>
/// Initializes a new instance of the <see cref="SigmoidFunction"/> class
/// </summary>
public SigmoidFunction( ) { }

/// <summary>
/// Initializes a new instance of the <see cref="SigmoidFunction"/> class
/// </summary>
///
/// <param name="alpha">Sigmoid's alpha value</param>
public SigmoidFunction( double alpha )
{
        this.alpha = alpha;
}


/// <summary>
/// Calculates function value
/// </summary>
///
/// <param name="x">Function input value</param>
///
/// <returns>Function output value, <i>f(x)</i></returns>
///
/// <remarks>The method calculates function value at point <b>x</b>.</remarks>
///
public double Function( double x )
{
        return ( 1 / ( 1 + Math.Exp( -alpha * x ) ) );
}

/// <summary>
/// Calculates function derivative
```

```
        /// <summary>
        ///
        /// <param name="x">Function input value</param>
        ///
        /// <returns>Function derivative, <i>f'(x)</i></returns>
        ///
        /// <remarks>The method calculates function derivative at point
<b>x</b>.</remarks>
        ///
        public double Derivative( double x )
        {
            double y = Function( x );

            return ( alpha * y * ( 1 - y ) );
        }

        /// <summary>
        /// Calculates function derivative
        /// </summary>
        ///
        /// <param name="y">Function output value - the value, which was obtained
        /// with the help of <see cref="Function"/> method</param>
        ///
        /// <returns>Function derivative, <i>f'(x)</i></returns>
        ///
        /// <remarks>The method calculates the same derivative value as the
        /// <see cref="Derivative"/> method, but it takes not the input <b>x</b> value
        /// itself, but the function value, which was calculated previously with
        /// the help of <see cref="Function"/> method. <i>(Some applications require as
        /// function value, as derivative value, so they can seve the amount of
        /// calculations using this method to calculate derivative)</i></remarks>
        ///
        public double Derivative2( double y )
        {
            return ( alpha * y * ( 1 - y ) );
        }


    public DoubleRange Range
    {
```

```
            get { return new DoubleRange(0, 1); }
        }
            }
    }
```

Appendix.12

This piece of code represent the Back propagation learning algorithm which is used in trained the multi-layer of ANN the system.

```
namespace AForge.Neuro.Learning
{
  using System;

  /// <summary>
  /// Back propagation learning algorithm
  /// </summary>
  ///
  /// <remarks>The class implements back propagation learning algorithm,
  /// which is widely used for training multi-layer neural networks with
  /// continuous activation functions.</remarks>
  ///
  public class BackPropagationLearning : ISupervisedLearning
  {
    // network to teach
    private ActivationNetwork network;
    // learning rate
    private double learningRate = 0.1;
    // momentum
    private double momentum = 0.0;

    // neuron's errors
    private double[][] neuronErrors = null;
    // weight's updates
    private double[][][] weightsUpdates = null;
    // threshold's updates
    private double[][] thresholdsUpdates = null;


    /// <summary>
    /// Learning rate
    /// </summary>
    ///
```

```
    /// <remarks>The value determines speed of learning. Default value equals to 0.1.</remarks>
    ///
    public double LearningRate
    {
      get { return learningRate; }
      set
      {
        learningRate = Math.Max(0.0, Math.Min(1.0, value));
      }
    }

    /// <summary>
    /// Momentum
    /// </summary>
    ///
    /// <remarks>The value determines the portion of previous weight's update
    /// to use on current iteration. Weight's update values are calculated on
    /// each iteration depending on neuron's error. The momentum specifies the amount
    /// of update to use from previous iteration and the amount of update
    /// to use from current iteration. If the value is equal to 0.1, for example,
    /// then 0.1 portion of previous update and 0.9 portion of current update are used
    /// to update weight's value.<br /><br />
    /// Default value equals to 0.0.</remarks>
    ///
    public double Momentum
    {
      get { return momentum; }
      set
      {
        momentum = Math.Max(0.0, Math.Min(1.0, value));
      }
    }

    /// <summary>
    /// Initializes a new instance of the <see cref="BackPropagationLearning"/> class
    /// </summary>
    ///
    /// <param name="network">Network to teach</param>
    ///
    public BackPropagationLearning(ActivationNetwork network)
```

```
double error = 0, e, sum;
// neuron's output value
double output;
// layers count
int layersCount = network.LayersCount;

// assume, that all neurons of the network have the same activation function
IActivationFunction function = network[0][0].ActivationFunction;

// calculate error values for the last layer first
layer = network[layersCount - 1];
errors = neuronErrors[layersCount - 1];

for (int i = 0, n = layer.NeuronsCount; i < n; i++)
{
   output = layer[i].Output;
   // error of the neuron
   e = desiredOutput[i] - output;
   // error multiplied with activation function's derivative
   errors[i] = e * function.Derivative2(output);
   // squre the error and sum it
   error += (e * e);
}

// calculate error values for other layers
for (int j = layersCount - 2; j >= 0; j--)
{
   layer = network[j];
   layerNext = network[j + 1];
   errors = neuronErrors[j];
   errorsNext = neuronErrors[j + 1];

   // for all neurons of the layer
   for (int i = 0, n = layer.NeuronsCount; i < n; i++)
   {
      sum = 0.0;
      // for all neurons of the next layer
      for (int k = 0, m = layerNext.NeuronsCount; k < m; k++)
      {
         sum += errorsNext[k] * layerNext[k][i];
```

```
            }
            errors[i] = sum * function.Derivative2(layer[i].Output);
        }
    }

    // return squared error of the last layer divided by 2
    return error / 2.0;
}

/// <summary>
/// Calculate weights updates
/// </summary>
///
/// <param name="input">Network's input vector</param>
///
private void CalculateUpdates(double[] input)
{
    // current neuron
    ActivationNeuron neuron;
    // current and previous layers
    ActivationLayer layer, layerPrev;
    // layer's weights updates
    double[][] layerWeightsUpdates;
    // layer's thresholds updates
    double[] layerThresholdUpdates;
    // layer's error
    double[] errors;
    // neuron's weights updates
    double[] neuronWeightUpdates;
    // error value
    double error;

    // 1 - calculate updates for the last layer fisrt
    layer = network[0];
    errors = neuronErrors[0];
    layerWeightsUpdates = weightsUpdates[0];
    layerThresholdUpdates = thresholdsUpdates[0];

    // for each neuron of the layer
    for (int i = 0, n = layer.NeuronsCount; i < n; i++)
```

```
    {
      neuron = layer[i];
      error = errors[i];
      neuronWeightUpdates = layerWeightsUpdates[i];

      // for each weight of the neuron
      for (int j = 0, m = neuron.InputsCount; j < m; j++)
      {
        // calculate weight update
        neuronWeightUpdates[j] = learningRate * (
          momentum * neuronWeightUpdates[j] +
          (1.0 - momentum) * error * input[j]
          );
      }

      // calculate treshold update
      layerThresholdUpdates[i] = learningRate * (
        momentum * layerThresholdUpdates[i] +
        (1.0 - momentum) * error
        );
    }

    // 2 - for all other layers
    for (int k = 1, l = network.LayersCount; k < l; k++)
    {
      layerPrev = network[k - 1];
      layer = network[k];
      errors = neuronErrors[k];
      layerWeightsUpdates = weightsUpdates[k];
      layerThresholdUpdates = thresholdsUpdates[k];

      // for each neuron of the layer
      for (int i = 0, n = layer.NeuronsCount; i < n; i++)
      {
        neuron = layer[i];
        error = errors[i];
        neuronWeightUpdates = layerWeightsUpdates[i];

        // for each synapse of the neuron
        for (int j = 0, m = neuron.InputsCount; j < m; j++)
```

```
            {
                // calculate weight update
                neuronWeightUpdates[j] = learningRate * (
                    momentum * neuronWeightUpdates[j] +
                    (1.0 - momentum) * error * layerPrev[j].Output
                    );
            }

            // calculate treshold update
            layerThresholdUpdates[i] = learningRate * (
                momentum * layerThresholdUpdates[i] +
                (1.0 - momentum) * error
                );
        }
    }
}

/// <summary>
/// Update network'sweights
/// </summary>
///
private void UpdateNetwork()
{
    // current neuron
    ActivationNeuron neuron;
    // current layer
    ActivationLayer layer;
    // layer's weights updates
    double[][] layerWeightsUpdates;
    // layer's thresholds updates
    double[] layerThresholdUpdates;
    // neuron's weights updates
    double[] neuronWeightUpdates;

    // for each layer of the network
    for (int i = 0, n = network.LayersCount; i < n; i++)
    {
        layer = network[i];
        layerWeightsUpdates = weightsUpdates[i];
        layerThresholdUpdates = thresholdsUpdates[i];
```

```
            // for each neuron of the layer
            for (int j = 0, m = layer.NeuronsCount; j < m; j++)
            {
              neuron = layer[j];
              neuronWeightUpdates = layerWeightsUpdates[j];

              // for each weight of the neuron
              for (int k = 0, s = neuron.InputsCount; k < s; k++)
              {
                // update weight
                neuron[k] += neuronWeightUpdates[k];
              }
              // update treshold
              neuron.Threshold += layerThresholdUpdates[j];
            }
          }
        }
      }
```

ملخص الأطروحة

تقترح هذه الدراسة نظام لنموذج أولي لجهاز استشعار الأنف اللإلكتروني باستخدام جهاز الاستشعار handheld odour meter (OMX-GR)، وذلك للتعرف على عدد من الروائح التي تركز عليها هذه الدراسة، والتي تتكون من ثلاثة أنواع من اللحوم، حيث من الممكن تمييز صالح منها والغير صالح. والشبكة العصبية الاصطناعية (ANN) هي جوهر النظام المقترح الذي تم إنتاجه باستخدام برنامج C # من خلال Visual Studio 2010 ، وهذه الشبكة هي التقنية المسؤولة عن تحليل البيانات والتعرف على الروائح. هذه الشبكة العصبية الاصطناعية (ANN) تم استخدام نوع محدد وهو backpropagation multi-layer algorithm مع عدد من المعايير المهمة الواجب توفرها لعمل النظام بنجاح مثل momentum و learning rates مع استخدام نوع التعلم (supervised learning) و sigmoid activation function. على أن تكون البيانات التي ستستخدم كبيانات إدخال أساسية في النظام تمتاز بنوعية ووتيرة خاصة بالشبكة العصبية الاصطناعية (ANN) .

هناك حاجة لنظام الـ ANN لأن يقرر ما هو أفضل بنية أو تصميم يمكن إختياره ومن ثم تطبيقه على النظام لتحقيق أفضل النتائج، وبعد العديد من التصاميم والأبنية خلال فترة التدريب، ونتيجة لذلك فإن البنية الفائزة، تبين أن نسبة 2: 6: 1 تحقق أفضل النتائج بالنسبة للنظام المقترح والتي حققت أقل نسبة معدل للخطأ (least mean of sum of square error) عند 0.03%، و training error 1.72729 ، و 0.77849 validation، و testing 0.03% ، وهذه النتائج هي أفضل نتائج سجلها النظام من خلال هذه البنية من أجل حل المشكلة. وتعتبر أفضل تصميم لـ ANN من أجل حل المشكلة التي لديها أقل نتيجة التحقق من الصحة.

في المستقبل، سيتم محاكاة وتقليد حاسة الشم وتطويرها لتطبق وتدمج في الكمبيوتر والأجهزة الذكية. وفقا لذلك، يعتقد بإن رائحة الشم ستضاف كعنصر جديد إلى العناصر الرئيسية في مجال الوسائط المتعددة (ملتميديا).

جامعة البحرين

كلية تقنية المعلومات



الأنف اللإلكتروني بالاستناد على الشبكة العصبية الاصطناعية(ANN) مع تطبيقات الوسائط المتعددة (الملتميديا)

مخطط بحث لاستكمال متطلبات الحصول على درجة الماجستير في تقنية المعلومات

إعداد

شيماء إبراهيم هجرس

الرقم الجامعي: 19990190

إشراف

الدكتور يوسف عبداللطيف نصرالله البستكي

(أستاذ مشارك)

جامعة البحرين

مملكة البحرين

مايو 2015