Intelligent Sensor Data Pre-processing Using Continuous Restricted Boltzmann Machine

by

Emil Zaidan bin Suhaimi

Dissertation submitted in partial fulfillment of
The requirements for the
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

JUNE 2007

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

Intelligent Sensor Data Pre-processing Using Continuous Restricted Boltzmann Machine

By

Emil Zaidan bin Suhaimi

A project dissertation submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
In partial fulfillment of the requirement for the
BACHELOR OF ENGINEERING (Hons)
(ELECTRICAL & ELECTRONICS ENGINEERING)

Approved by,

(DR. NOR HISHAM BIN HAMID)
Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

June 2007

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

EMIL ZAIDAN BIN SUHAIMI

ACKNOWLEDGEMENT

First and foremost, I would like to thank Allah for His Blessing and giving me the strength to complete this project. Verily all good comes from Him and all shortcomings are due to my own weaknesses.

My sincere and heartfelt thanks to my project supervisor, Dr. Nor Hisham bin Hamid, for his supervision and countless hours spent in sharing his knowledge and valuable experiences in order to make sure my project is done successfully. His MATLAB implementation of CRBM, which was modified for this project, is an essential component of the simulation process. As a supervisor, he has been a source of motivation towards the completion of this project.

Many thanks to Miss Illani M Nawi, the Final Year Research Project Coordinator, who kindly made the arrangement to provide the students with the necessary tools and materials for research works. A large measure of gratitude also goes to the all lecturers and laboratory technicians who had helped me in completing this project.

I would also like to express a special thank to my family members for their priceless support, encouragement, constant love, valuable advices, and their understanding of me. Without all of them, I would not be where I am right now.

Last but not least, thanks to all my friends who have involved directly and indirectly in making my research project a success.

ABSTRACT

The objective of the project is to find a solution to pre-process noisy signal for sensors in Lab-on-a-Chip (LOC) and System-on-Chip (SOC) technologies. This solution must be able to process continuous-time, analogue sensor signals directly. It must also be amenable to hardware implementation, with low power consumption. This solution is found in the Continuous Restricted Boltzmann Machine (CRBM), which is a type of Artificial Neural Network which exhibits probabilistic and stochastic behavior. CRBM utilizes continuous stochastic neurons, where Gaussian noise is applied to the input of the neurons. The noise inputs cause neurons to have continuous-valued, probabilistic outputs. The use of stochastic neurons in CRBM gives it modeling flexibility that is useful with real data. The training algorithm of CRBM requires only addition and multiplication, which is computationally inexpensive in hardware and software. The ability of CRBM to model any given data set is shown by training the CRBM on various data sets reflecting real-world data. In this study, CRBM is shown to be suitable to be implemented in LOC and SOC applications aforementioned.

TABLE OF CONTENTS

CHAPTER 5

LIST OF FIGURES

LIST OF ABBREVIATIONS

ANN	Artificial Neural Network
CRBM	Continuous Restricted Boltzmann Machine
LOC	Lab-on-a-Chip
SOC	System-on-Chip
SLP	Single-layer Perceptron

CHAPTER 1
INTRODUCTION

1.1 BACKGROUND OF STUDY

Development in Lab-on-a-Chip (LOC) and System-on-Chip (SoC) technologies has increased the interest in electronic health care [1], with applications such as telemedicine, bioanalysis, patient monitoring and implantable devices. These biomedical instruments provide constant monitoring of essential physiological parameters such as temperature, pH, oxygen and pressure [2–5]. For the applications mentioned, it is advantageous that the instruments be integrated and miniaturized. However, miniaturization increase the difficulty of extracting useful information from a far more noisy and unstable measurements. Due to this, there is a need for a robust, adaptive algorithm for sensor data pre-processing, which can process continuous-time, analogue sensor signals directly [6]. This algorithm must be amenable to hardware implementation, with low power consumption, to meet the requirement of the LOC or SOC application. One of the candidates for this application is the Continuous Restricted Boltzmann Machine, a type of Artificial Neural Network.

Artificial Neural Network (ANN) is a computational network consisting of interconnected processing units called neurons [7]. These processing units are connected together in a specific way to perform a particular task. Data processing is done collectively and in parallel by the neurons, different from conventional computer which is a sequential machine that process data step by step [8]. ANN also possess the ability to learn, and they are suitable for solve problems which is nonlinear or mathematically ill-defined.

The Continuous Restricted Boltzmann Machine (CRBM) is a type of Artificial Neural Network architecture which exhibits probabilistic and stochastic behavior. This is because CRBM utilizes continuous stochastic neurons, where Gaussian noise is applied to the input of the neurons. The noise inputs cause neurons to have continuous-valued, probabilistic outputs. The use of stochastic neurons in CRBM gives it modeling flexibility that is useful with real data. CRBM has been proven experimentally to be able to model continuous data successfully with a simple, reliable training algorithm [9].

1.2 PROBLEM STATEMENT

The instruments and sensors used in SoC and LOC applications have to be integrated directly into the hardware, and this means that they have to be miniaturized. However, it becomes more difficult to extract useful information from a far more noisy and unstable measurements. Therefore, a sensor pre-processing system is needed to process the various sensor signals before it is processed by the main microcontroller of the system. By doing this, the workload of the microcontroller will be reduced, as the microcontroller does not need to directly process all the raw sensor data. This task will be done by the pre-processing units in parallel, thus reducing the overall time the system needs to complete its tasks.

Due to this, there is a need for a robust, adaptive algorithm for sensor pre-processing, which can be implemented directly in hardware, with low power consumption. This algorithms must be able to process continuous-time, analogue sensor signals directly. One of the candidates for the task of sensor data pre-processing is the Continuous Restricted Boltzmann Machine (CRBM).

1.3 OBJECTIVE & SCOPE OF STUDY

1.3.1 Objective

The objective of the project is to find a solution to pre-process noisy signal for sensors in Lab-on-a-Chip (LOC) & System-on-Chip (SOC) technologies. This solution must be able to process continuous-time, analogue sensor signals directly. It must also be amenable to hardware implementation, with low power consumption.

1.3.2 Scope of Study

The project focuses on several subjects:

- Detail familiarization of the Artificial Neural Network, specifically the Continuous Restricted Boltzmann Machine (CRBM) architecture
- Understanding of the MATLAB implementation of the CRBM
- Training of the CRBM on various data sets
- Finding the initial parameters for the CRBM so that an optimum result will be obtained.

CHAPTER 2
LITERATURE REVIEW

2.1 ARTIFICIAL NEURAL NETWORK

Artificial Neural Network (ANN) is a computational network consisting of interconnected processing units called **neurons** [7]. These processing units are connected together in a specific way to perform a particular task. Data processing is done collectively and in parallel by the neurons, different from conventional computer which is a sequential machine that process data step by step [8]. Learning is done in an ANN by adjusting the weight on the connections between neurons. This follows the Hebbian Rule, which states that "*When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased*" [8].

ANN may be classified into three different classes of **network architecture**: single-layer feedforward network, multilayer feedforward network, and recurrent network [6].

- In **feedforward network** (*Figure 2.1a*), neurons are organized to form layers. Basically, the layers consist of the input layer and output layer, where the signal from the input layer are projected onto the output layer, which acts as the computation nodes.
- In the case of **multilayer feedforward network** (*Figure 2.1b*), one or more hidden layer is present between the input and output layers to produce more synaptic connections, enabling the network to compute more complex problem.

- On the other hand, the **recurrent network** (*Figure 2.1c*) differs from the other two by having at least one feedback loop, in which a neuron feeds its output back to the inputs of other neurons or itself (self-feedback). The presence of feedback loops improves the performance and learning capability of the network.

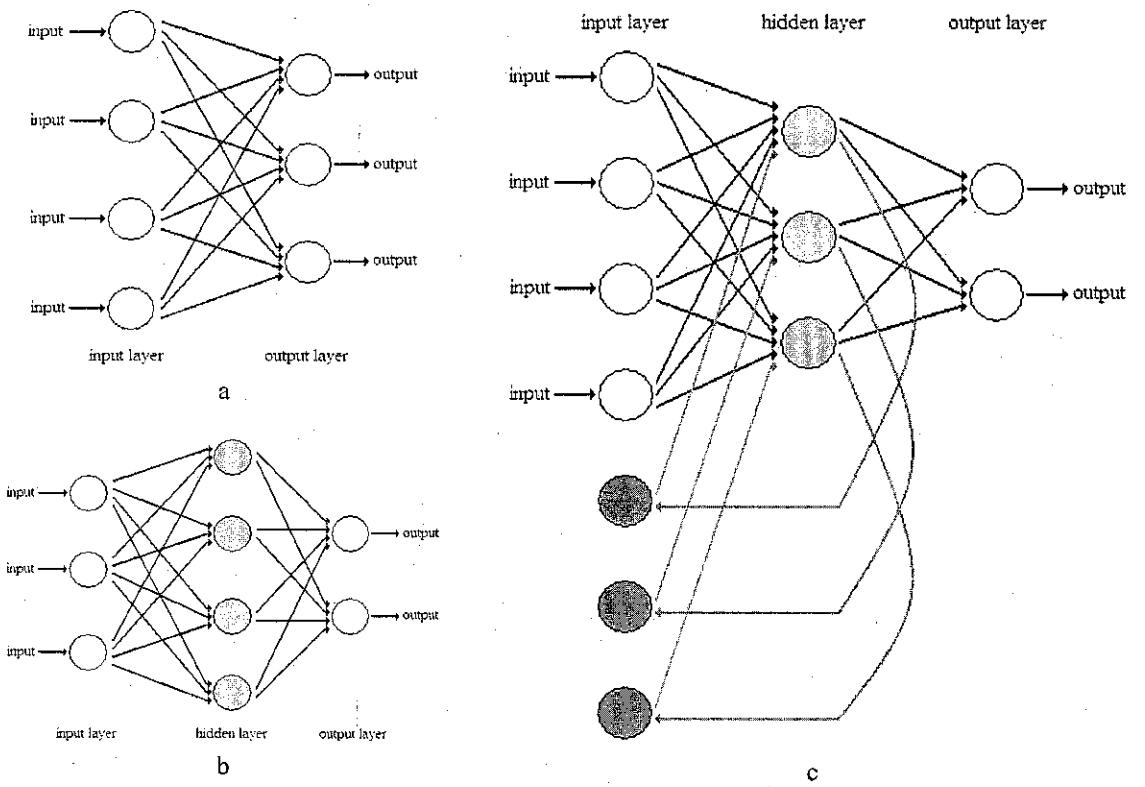


Figure 2.1: Artificial Neural Network architecture

a Single-Layer Feedforward Network
b Multilayer Feedforward Network
c Multilayer Recurrent Network

2.2 BASIC MODEL OF A NEURON

The neuron (*Figure 2.2*) is the basic unit of an ANN. A neuron typically consists of n inputs s_j, where $j = \{1,2,3,...,n\}$. Each input is weighted by the weight factor w_{ij} (**synaptic weight**), and the products of the inputs and weights are summed.

$$x_i = \sum_{j=1}^{n} w_{ij} s_j$$

The value x_i is then passed through a **non-linear function** $\varphi_i(x_i)$ to obtain the output of the neuron, s_i. The function $\varphi_i(x_i)$ is also called a transfer function, and can be the sigmoid function, piecewise linear functions, or step functions [12].

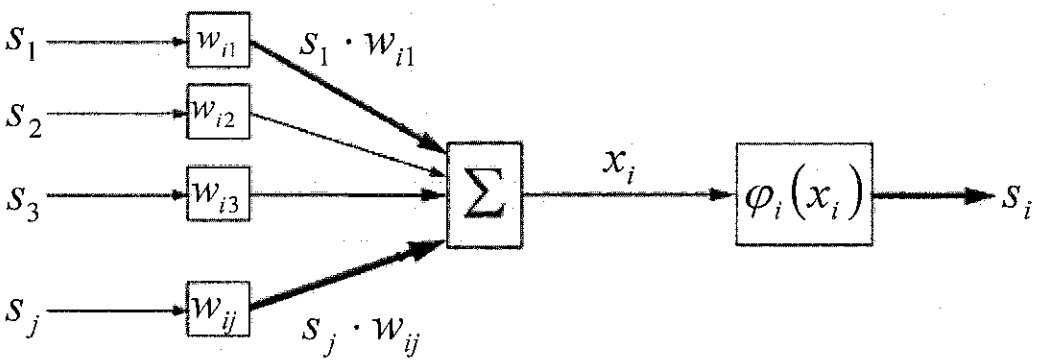


Figure 2.2: Basic neuron model

The purpose of the nonlinearity function $\varphi_i(x_i)$ is to keep the output of a neuron bounded. This means that the output is dampened or conditioned, and this keeps it under control [11]. This stems from the fact that the nonlinear functions used in neurons have an upper and/or lower limit, such as ±1, ±½, or {+1,0}.

2.3 LEARNING IN ARTIFICIAL NEURAL NETWORK

One of the interesting properties of ANN is its ability to learn. Learning is defined as the process where neural network adapt to inputs and adjust itself to produce a desired output. During the process of learning, a network will adjust its synaptic weights when given input so that the value of its output approaches the target output. When the resulting output is equal to the target output, the network is said to have undergone its learning process sufficiently [11].

There are several types of learning, each suitable for different kind of ANN. Some of them are listed below [7][11].

- **Supervised learning**

  In supervised learning, a training data set is prepared, which consists of pairs of input object, and desired output. The input is applied to the ANN, and the resulting output is compared with the desired output. If the actual output is different from the desired output, an error signal will be generated, which is used to calculate the adjustments for synaptic weights to make actual output equal the desired output. This is done continuously until the error is sufficiently small and the output equals the input.

- **Unsupervised learning**

  In unsupervised learning, no target output is set. During training, the ANN will receive many inputs, and the network will categorize the inputs according to a set guideline. This guideline determines how the ANN forms groups or classes for the inputs. When an input is received, the ANN will indicate which class the input belongs. If a class is not found, a new class will be generated.

- **Reinforced learning**

  This type of learning requires one or more neurons at the output layer and a "teacher". During training, input is given, and the teacher will give "pass" indication if the actual output is the same as target output, or "fail" indication otherwise. For a "fail" signal, the network will adjust its parameters, and the output is checked again by the teacher. This is repeated until the teacher gives

a "pass" indication. There is no indication whether the outputs are improving or how close the actual output to the target output. Due to this, the learning technique must have certain boundaries so that the ANN will not keep trying to get the correct output indefinitely.

Practically, learning in an ANN is done through the use of **learning algorithm**. This is a mathematical tool to specify how an ANN will reach a steady state of its parameters. The learning algorithm usually consists of an **error function**, which is usually expressed in terms of weights, inputs and outputs of a neuron. To reach a steady state, the ANN will adjust its parameters so that the error will be minimized sufficiently or reach zero. When a steady state is reached, the ANN has completed its learning phase [11].

There are many different types of learning algorithm available, and each of the algorithms suits a certain type of ANN model. Selection of a suitable algorithm is important so that the ANN model will operate at its full potential.

2.4 CONTINUOUS RESTRICTED BOLTZMANN MACHINE

The **Continuous Restricted Boltzmann Machine** (CRBM) is a type of Artificial Neural Network architecture which shows probabilistic and stochastic behavior. This probabilistic behavior is due to the stochastic neurons inside the architecture. The noise inputs cause neurons to have continuous-valued, probabilistic outputs.

CRBM is classified as a Recurrent Network due to the presence of feedback loop in the architecture; however there is no self-feedback in this architecture, as no unit has a connection with itself. All connections are symmetric, meaning that the weight of the connection between neuron A and neuron B is equal both ways (*Figure 2.3*).

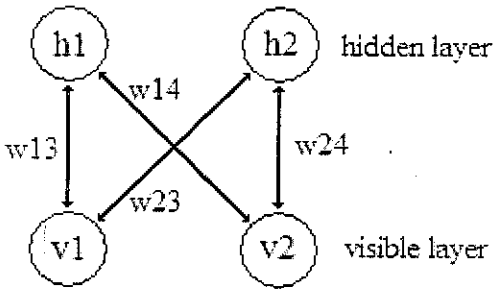


Figure 2.3: CRBM architecture, with no self-feedback and restricted & symmetric connection

The neuron employed by CRBM can be seen in *Figure 2.4*. The inputs, outputs, and synaptic weights of the neuron are represented by s_j, s_i and w_{ij} respectively. The output is obtained using the equation

$$s_i = \varphi_i\left(\sum_j w_{ij}s_j + \sigma \cdot N_i(0,1)\right),$$

Where $\varphi_i(x)$ is the sigmoid function

$$\varphi_i(x_i) = \theta_L + (\theta_H - \theta_L)\cdot\frac{1}{1+\exp(-a_i x_i)}$$

The stochastic nature of the neurons is due to the Gaussian input noise. This input noise is represented by $n_i = \sigma \cdot N_i(0,1)$, where σ is a constant and $N_i(0,1)$, is a Gaussian random variable with zero mean and unit variance. This input noise follows the probability distribution:

$$p(n_i) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-n_i^2}{2\sigma^2}\right)$$

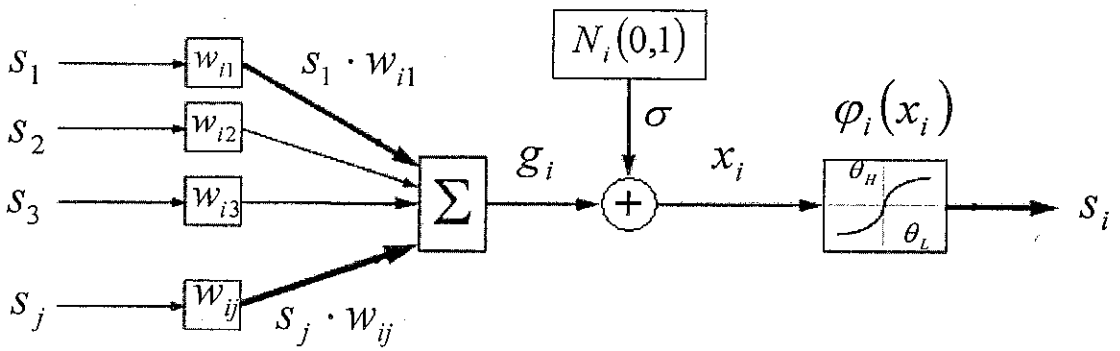


Figure 2.4: Continuous stochastic neurons employed by CRBM

The "restricted" in its name signifies that CRBM does not allow connection between hidden units, which leads to a more efficient learning process. This is due to the fact that restriction on the connection will speed up the model significantly. The neurons in the same layer are not connected to each other, so they are conditionally independent. This allows neurons on the same layer to be updated in parallel, reducing the processing time. This restriction also simplifies the mathematical analysis of the model, as the restricted connection is equivalent to setting a synaptic weight to zero. Furthermore, any decrease to performance due to restricted connection is insignificant, and it can be compensated by recruiting extra neurons into the hidden layer of the network [9].

CRBM uses the Minimizing-Contrastive-Divergence (MCD) method [9][12] as its learning algorithm. This method is used to update the weight, w_{ij} and sigmoid function variable, a_i on each iteration of the CRBM.

$$\Delta \hat{w}_{ij} = \eta_w \left(\left\langle s_i \cdot s_j \right\rangle_0 - \left\langle \hat{s}_i \cdot \hat{s}_j \right\rangle_1 \right)$$

$$\Delta \hat{a}_i = \frac{\eta_a}{a_i^2} \left(\left\langle s_i^2 \right\rangle_0 - \left\langle s_i^2 \right\rangle_1 \right)$$

$\hat{s}_i$ and $\hat{s}_j$ denotes the previous state of the input and output connected to the weight w_{ij}, while η_w and η_a represents the learning rates of w_{ij} and a_i respectively.

2.5 LAB-ON-A-CHIP (LOC) & SYSTEM-ON-CHIP (SOC)

Lab-on-a-chip (LOC) is a term for devices that integrate (multiple) laboratory functions on a single chip of only millimeters to a few square centimeters in size. These devices are capable of handling extremely small fluid volumes down to less than picoliters, reducing the amount of sample required to do a test. Due to its construction, such devices are also disposable, making it suitable for clinical applications [13].

System-on-a-chip or **system on chip** (SoC or SOC) is the integration of all components of an electronic system into a single integrated circuit. It may contain digital, analog, mixed-signal, and often radio-frequency functions all on one chip. There are several advantages of integrating all the components onto a single chip, such as lower cost, smaller space requirement, and higher system reliability [3][14].

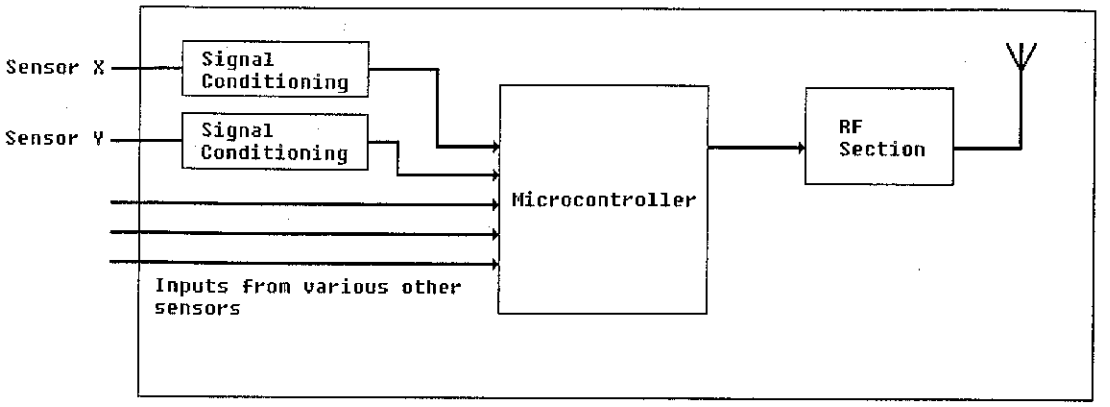


Figure 2.5: Patient Monitoring System

An example of an LOC system is illustrated in *Figure 2.5*. This system functions as a Patient Monitoring System. In this system, the various sensors available will measure the important physiological parameters of a patient, and transmit the information to a remote location for monitoring. This LOC is packaged in a pill or other shapes that allow it to be implantable under the skin.

The device will have many sensors measuring the various physiological parameters of the patient. This data will all be processed by the microcontroller of the system, and without pre-processing, the microcontroller have to do a lot more work interpreting the data, and this will slow down the system. The microcontroller will also have to deal with the noisy data from the sensors, and this will impair the performance of the system. By having a pre-processing unit on each sensor, the sensor data will be pre-processed simultaneously, decreasing the workload of the microcontroller and reducing the processing time of the system. The presence of a pre-processing unit is illustrated in *Figure 2.6*, where the unit takes in input from sensor X and sensor Y before giving a single output to the microcontroller.

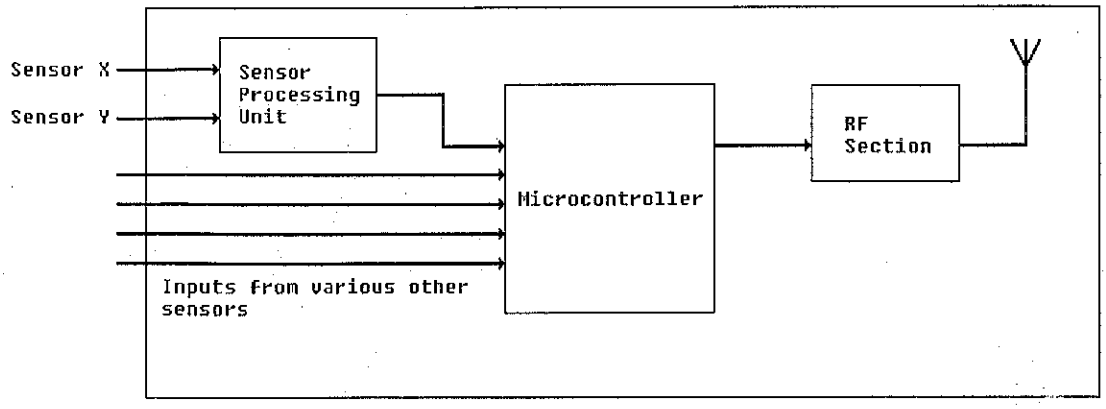


Figure 2.6: Patient Monitoring System (with Sensor Pre-processing Unit)

2.6 ADAPTIVE, INTEGRATED SENSOR PRE-PROCESSING IN LOC & SOC

In the example system, the type of pre-processing done is classification of the readings of sensor. In this case, two sensors will measure the pH value of blood and body temperature respectively. The temperature reading is taken as a reference, as the reading of a pH sensor will vary as temperature changes.

The normal pH of blood is 7.4, and a deviation of 0.2 either way will indicate that something is wrong. So the classifier needs to classify between 7.4 (normal) and otherwise (abnormal). This is illustrated in *Figure 2.7.*

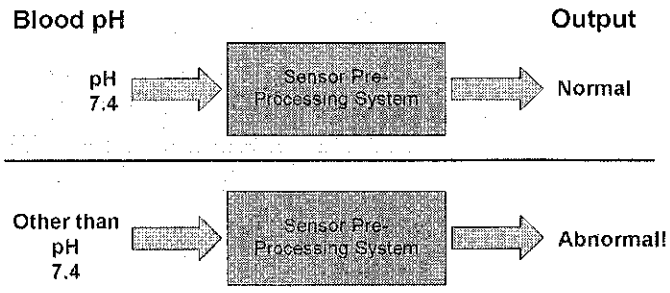


Figure 2.7: Classification done by the Sensor Pre-processing System

To preprocess the sensor data, the sensor pre-processing system needs to able to process the continuous & analogue temperature and pH measurement, and also accept the noisy readings of the sensor without any decrease in accuracy of the classification process.

2.6.1 CRBM as Sensor Data Pre-processor

CRBM possess all the qualities needed for the sensor processing of LOC and SOC application as mentioned above.

- Stochasticity

  Being a stochastic model, CRBM modeling ability is superior to deterministic model. Deterministic model requires accurate arithmetic computation to process data, which becomes infeasible due to SNR which degrades in noisy environment. On the other hand, a stochastic model is tolerant to noise, due to the use of internal noise in the architecture itself.

- Continuous Model

  Being a continuous model, CRBM is able to model continuous data faithfully. On the other hand, discrete-valued models will experience loss of data due to digitization or sampling process used in such models. Discrete-valued models also face the problem of noise, as repeated sampling process in such models will increase the level of noise [4].

- Simple Algorithm

  The learning algorithm of CRBM consists of additions and multiplications, which makes it easy to implement on VLSI. This is different from models employing mathematical operations such as integrations and derivations, which have more complex hardware requirement.

- Restricted Connection

  CRBM architecture uses a restricted connection, where there is no connection between neurons on the same layer. This reduces the complexity of the network, and also reduces the processing time of the network, as neurons on the same layer are conditionally independent [5], allowing them to update their state in parallel.

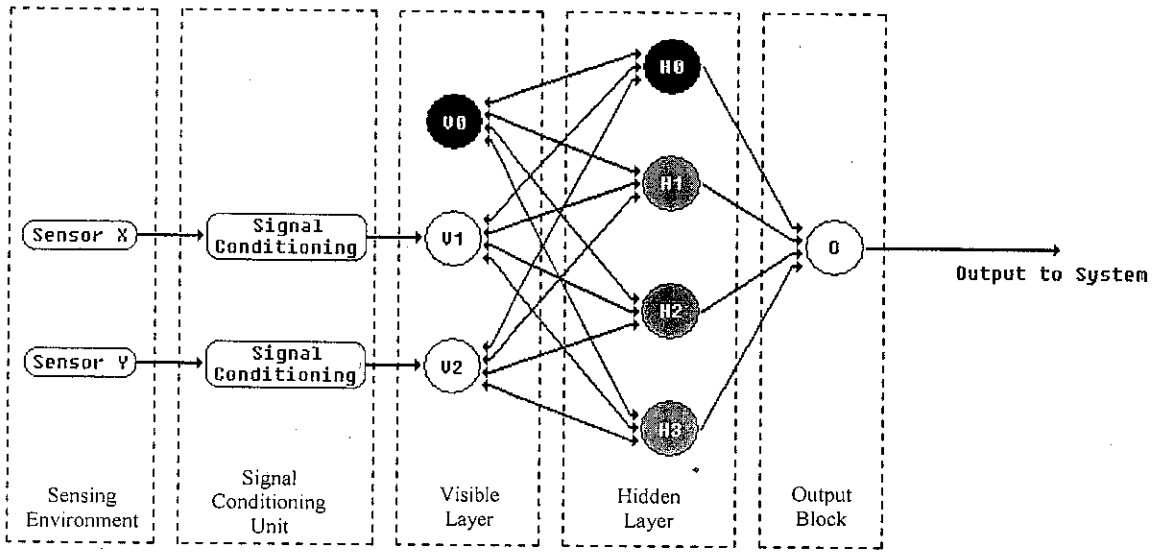


Figure 2.8: Sensor Pre-Processing Unit

Figure 2.8 illustrates how CRBM fits into the overall Sensor Pre-Processing Unit. The various stages of this unit are as follows:

- Stage I - Sensors

  This stage comprises of the pH and temperature sensors. The sensors will continuously take the respective readings and input it to the 2nd stage of the Sensor Pre-Processing Unit

- Stage II - Signal Conditioning Unit

  This stage will manipulate the input from the sensors so that they meet the requirements of the next stage.

- Stage III - CRBM (Visible & Hidden Layers)

  This stage will model the data distribution of the input, namely data set of pH against temperature.

- Stage IV - Output block

  This stage consists of a single-layer perceptron (SLP), which acts as a binary classifier for the output of CRBM. It will detect the state of the hidden layer of CRBM. For a certain input data, one of the neuron in the hidden layer will be activated, and this is detected by the SLP and labeled accordingly.

The output of the system will be binary, signifying whether the input detects a normal pH value or otherwise. The output of the Sensor Processing Unit is used by the Patient Monitoring System.

2.6.2 Training of CRBM

Before the system is employed, the CRBM must first be trained on data sets which have a certain data distribution. This data distribution represents the effect of noise on the readings of the sensors. *Figure 2.9a* shows the ideal readings of the sensors for pH 7.4 at temperature T. Due to environmental noise, sensor readings will be distributed in cluster close to the actual reading. *Figure 2.9b* shows the actual measurement of pH which forms a cluster. CRBM is trained to model the data distribution of this data set.

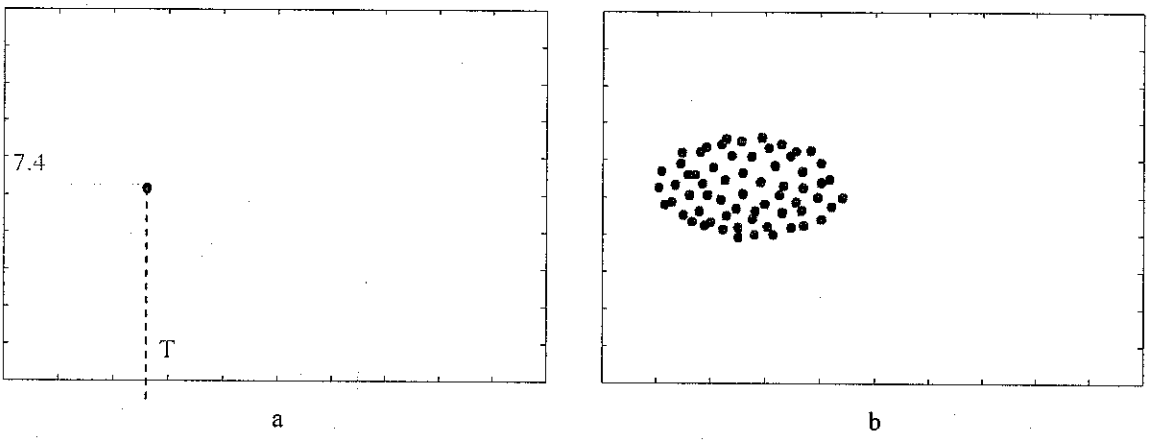


Figure 2.9: Readings of Sensors
a Ideal pH 7.4 at temperature T
b Measurements of pH 7.4 against temperature taken from the sensors, consisting of 100 data points

The training process is shown in *Figure 2.10*. The training data set is presented to the CRBM. CRBM uses the data to adjust its weights and noise input parameters according to the equation in Section 2.4. This parameter adjustment is done many times until the values of the weights and noise input parameters are stable. When the steady state is reached, the CRBM is considered to have learned the data distribution of the input data set. When a set of random data is given to this trained CRBM, it will be able to reconstruct the data according to the distribution it has been trained for.

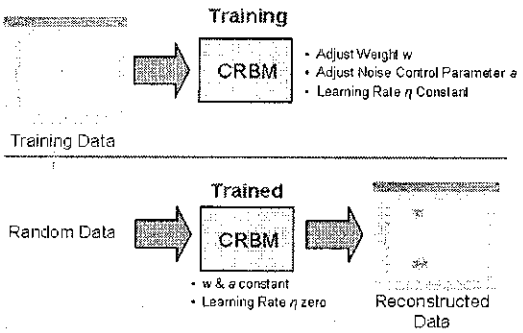


Figure 2.10: Training of CRBM

After the training of CRBM completes, certain hidden units will be an 'expert' of a data cluster, meaning that when a data belongs to a certain cluster, the hidden neuron will have a high activity. This can be seen in the graph of weight change against epoch (*Figure 2.11*), where 2 weights are seen to actively change; this corresponds to 2 input connections to a hidden neuron which is actively modeling the 2 clusters.

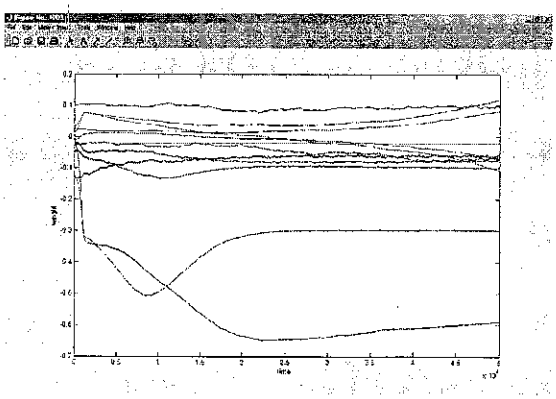


Figure 2.11: Weight change shows that a neuron is 'actively learning'

These activities of the experts, when classified, are used by the Sensor Processing Unit to recognize the cluster representing pH 7.4. This is done by training the SLP to classify the hidden unit activities of CRBM. First, all weights and noise control parameters of CRBM are clamped to their current values, and learning rates of CRBM are set to zero to prevent any learning from taking place. Data sets are then presented to the input of CRBM, and the activities of hidden layer are mapped by SLP to a known-correct output classification. The SLP will adjust its weights according to the Delta Rule, and after training completes, the SLP will be able to classify the output of CRBM.

As an end result, the Sensor Processing Unit will give a HIGH output if presented with data inside the pH 7.4 cluster, and LOW output if given data outside the cluster. This information is presented to the system microcontroller for further processing.

CHAPTER 3
METHODOLOGY

In this project, CRBM is implemented in MATLAB, with the code given in *Appendix I.* This implementation is separated into several modules, which does the different tasks in operation of the CRBM. The MATLAB code represents a CRBM with 2 inputs. There are 12 modules in total, as seen in *Appendix I.*

The function of several modules is as follows:

- **CRBM.m** – This is the main module, where all the parameters are set, such as the initial weights and learning rates for weight and noise parameter.
- **Sensormodela_2cluster.m** – Generates data sets that are used to train the CRBM. This module creates a data set where the data is distributed in 2 clusters.
- **sensor_plot.m** – Generates the graph of the training data, presented in 2-dimension, representing the 2 input of the CRBM.
- **aplot.m** – Generates the graph of the changing value of noise parameter in the visible units of CRBM over time.
- **aaplot.m** - Generates the graph of the changing value of noise parameter in the hidden units of CRBM over time.
- **bplot.m** - Generates the graph of the changing value of weights of connection of CRBM over time.
- **exam.m** – Generates random data sets and presents it to the trained CRBM to examine the learning ability of the architecture.

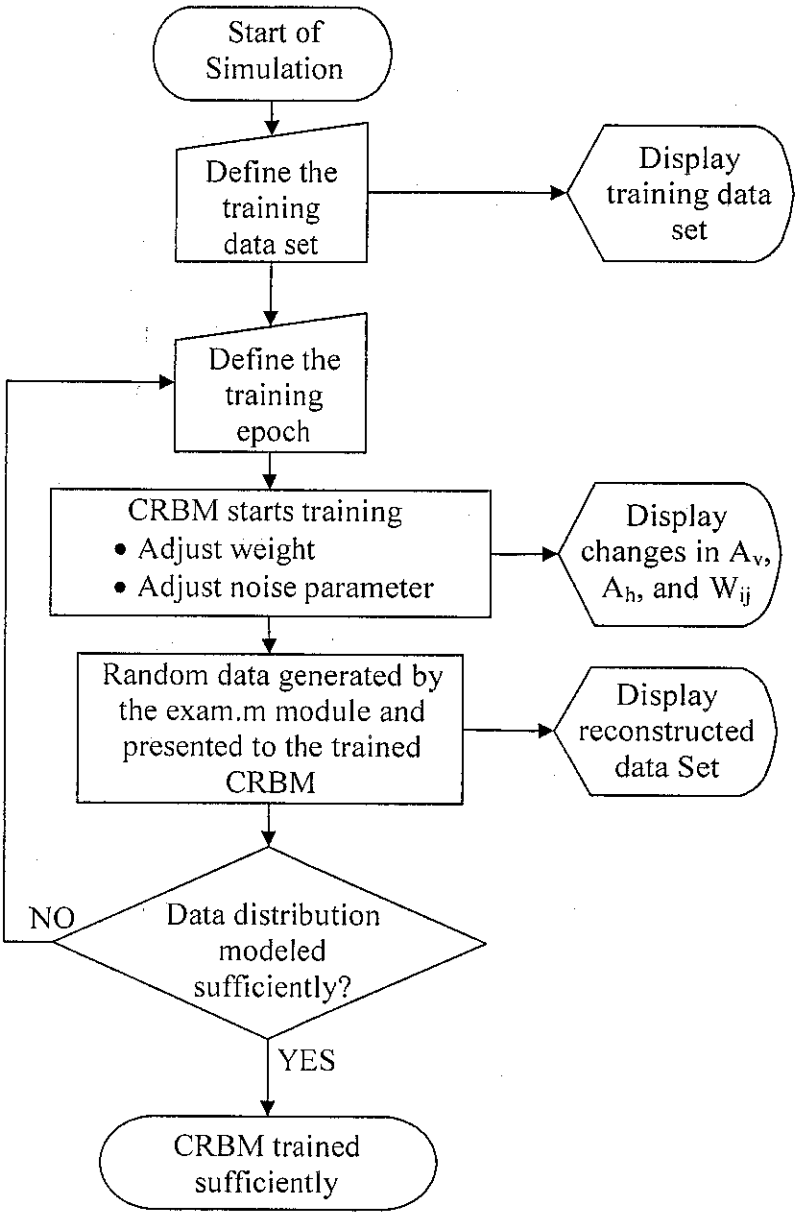


Figure 3.1: CRBM simulation workflow

In order to simulate the training of CRBM, a workflow of the simulation process is planned, and this is shown in *Figure 3.1*. First, a data set is chosen for the training. This data set consists of a set of data points which is clustered according to certain data distribution. These data sets are plotted on a 2-dimensional graph, representing a reading of a sensor against another reference sensor.

In the MATLAB implementation of the CRBM, these data sets are generated by the **Sensormodela_2cluster.m** module (*Appendix I, page 32*). By adjusting the values of the variables **x** and **y** (*Appendix I, page 30*), 2 clusters with different data distribution can be created. The variables **x** and **y** correspond to the coordinates of the center of the 2 clusters on the 2-dimensional plot.

Next, the value of epoch is set at the variable **num_iter** (*Appendix I, page 30*). This value determines how long the CRBM is trained. A suitable value must be chosen to make sure the training period is long enough for the CRBM to succeed in modeling the distribution of the training data set.

Once the initial parameters are set, the simulation is started by running the MATLAB program. The CRBM will adjust its weights and noise input parameters until the values of the weights and noise input parameters are stable. The changing values of the weights and noise parameters are displayed by MATLAB. When steady state is reached, the CRBM is considered to have learned the distribution of the input data set.

Lastly, a set of random data is given to this trained CRBM, and CRBM will reconstruct this data according to the distribution of the training data. MATLAB will display the reconstructed data plot, and observation is made to see whether CRBM have sufficiently learned the distribution of the training data set. If the reconstructed data matches the training data in its distribution, the CRBM is considered to have completed training. Otherwise, the simulation is repeated by using a different value of epoch.

In these simulations, the parameters that are varied are the training epoch and the training data set used. All other parameters are kept constant. The complete result can be found in *Appendix II*. Three different types of data set are used: non-symmetric clusters, horizontally-arranged clusters, and vertically-arranged clusters (*Figure 3.2*).

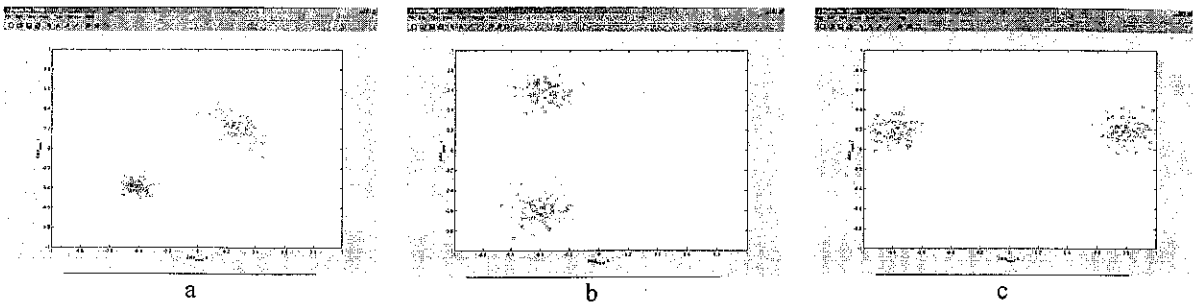


Figure 3.2: Data sets with 20 samples used to train CRBM

a Non-symmetric clusters
b Vertically-arranged clusters
c Horizontally-arranged clusters

These data sets may represent a particular kind of real-world data. Examples of real-world data are readings from sensors, such as temperature, pH, oxygen or concentration. These readings are distributed in such a fashion due to the presence of environmental noise, and also due to the inherent flaws of the sensors.

The CRBM is trained using a set of data for different values of epoch. A small value will result in insufficient time for the CRBM to train properly. On the other hand, a value too large will make the duration of training too long. This is a problem if the computer where the simulation is done is not powerful enough, causing system instability. This value of epoch is determined through experiments, and the epoch needed for the CRBM to train depends on the training data itself.

CHAPTER 4
RESULT & DISCUSSION

For a training data set with non-symmetric clusters (*Figure 3.1a*), it is observed that at epoch=10000, the CRBM has undergone sufficient training, and further increase in epoch shows no significant improvement (*Figure 4.1*). This shows that the CRBM has been trained successfully.

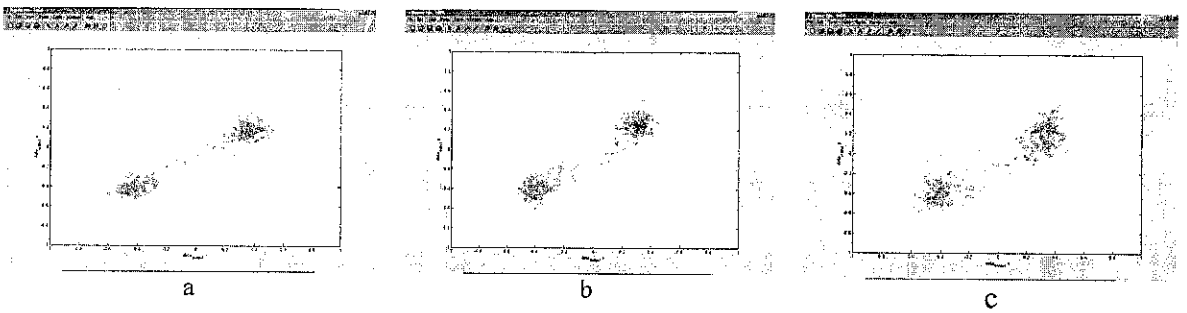
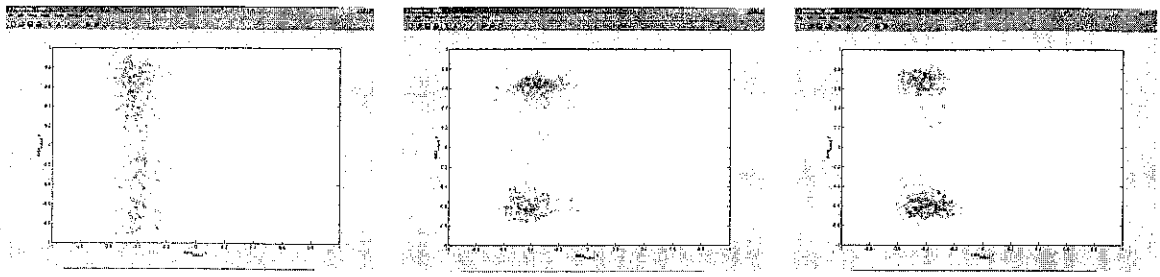


Figure 4.1: The output of CRBM trained with data set with non-symmetric clusters
a Epoch = 10000
b Epoch = 20000
c Epoch = 30000

For a training data set with vertically-arranged clusters (*Figure 3.1b*), it is observed that the time taken for proper training is longer than the previous one. The training is still not completed at epoch=10000. The CRBM completed its training after epoch=20000 and further increase in epoch shows no significant improvement (*Figure 4.2*). This longer duration may be due to the position of the clusters. The two clusters share the same value for the x-component, due to them being on the same vertical axis. This makes it harder for the CRBM to classify the clusters, prolonging the training duration.

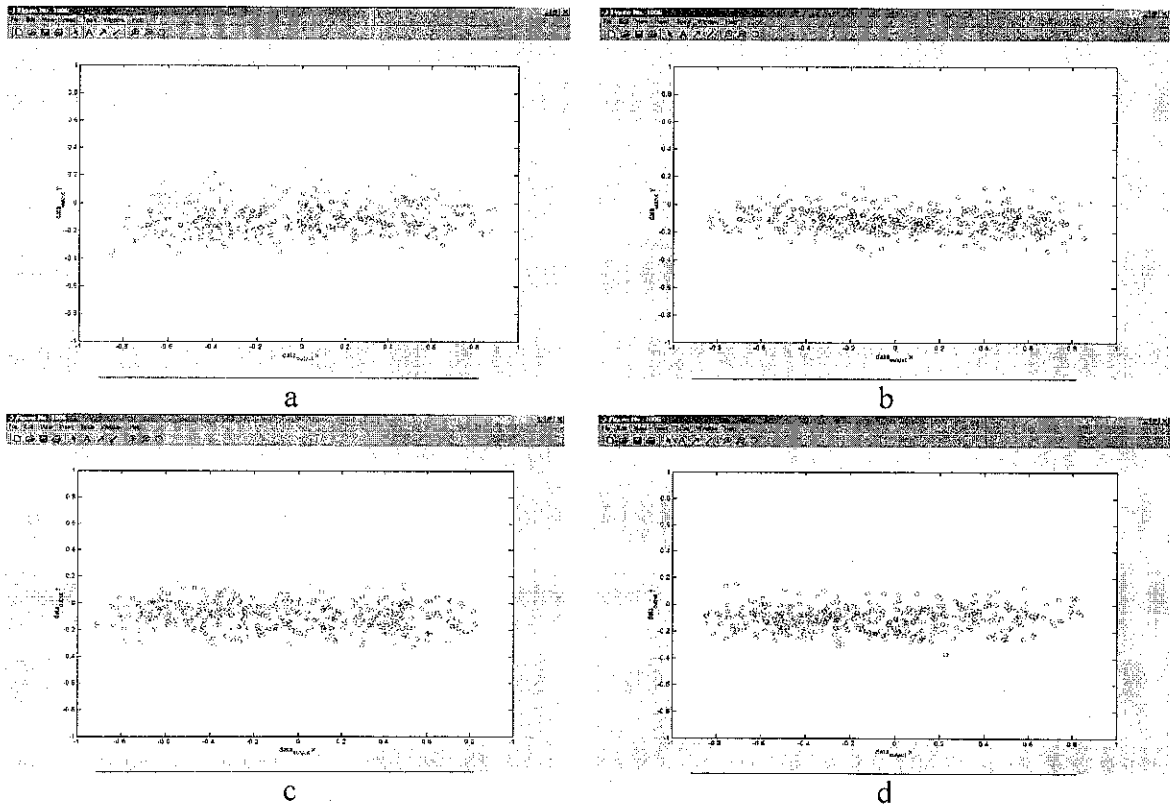


a b c

Figure 4.2: The output of CRBM trained with data set with vertically-arranged clusters

a Epoch = 10000
b Epoch = 20000
c Epoch = 30000

For a training data set with horizontally-arranged clusters (*Figure 3.1c*), it is observed that the CRBM fails to complete the training even after epoch = 50000 (*Figure 4.3*). Increasing the training duration might allow the CRBM to complete training, but simulation has not been done to test this yet.



a b c d

Figure 4.3: The output of CRBM trained with data set with horizontally-arranged clusters

a Epoch = 10000
b Epoch = 20000
c Epoch = 30000
d Epoch = 50000

The failure of CRBM to complete training for this data set might be due to the proximity of the two clusters. It is known that the closer the cluster of data is, the harder for the CRBM to do classification. To test this, a new data set with a different position of data cluster is used. Compare the old data set (*Figure 4.4 a*) with the new one (*Figure 4.4 b*) with a larger gap between the clusters. It is observed that the CRBM trained with the new data set completed its training successfully at epoch = 10000, whereas the one trained with the old one failed to do so. This shows that the closer the clusters of training data, the harder for the CRBM to classify.

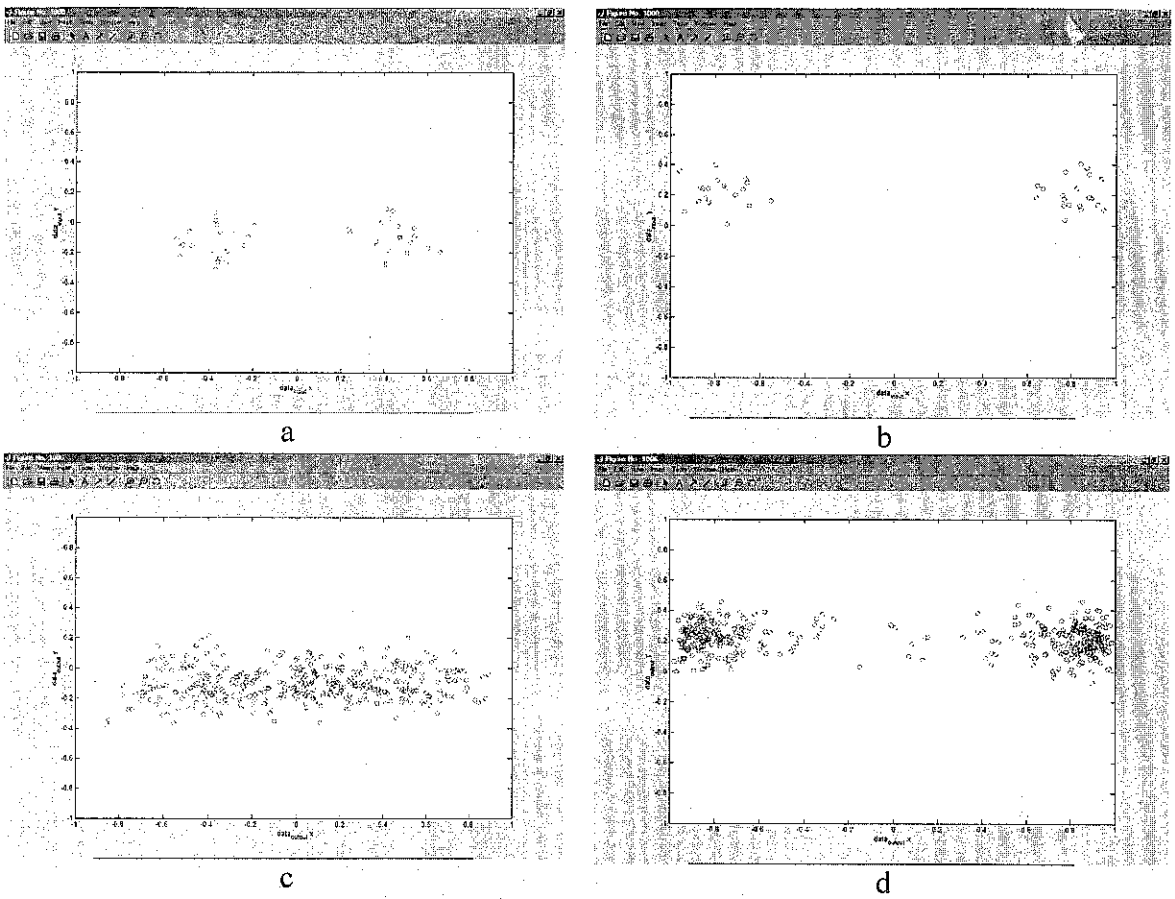


Figure 4.4: Training done with horizontally-arranged clusters

a Old training data set with small gap between clusters
b New training data set with large gap between clusters
c Output of CRBM trained with old data set after epoch = 10000
d Output of CRBM trained with new data set after epoch = 10000

CHAPTER 5
CONCLUSION & RECOMMENDATION

5.1 CONCLUSION

The Continuous Restricted Boltzmann Machine can be implemented to do sensor data pre-processing tasks in Lab-on-a-Chip & System-on-Chip applications, due to the CRBM being a probabilistic and stochastic model, and its ability to model continuous and noisy data. Its simple algorithm and restricted connection also makes CRBM amenable to hardware implementation, thus making it a suitable choice for LOC and SOC applications.

CRBM has been trained to model various data distribution expected to be seen in real-world application. After training, CRBM are able to reconstruct any random data set given to it according to the distribution it has been trained for, given that suitable initial parameters are given to the CRBM. The flexibility of CRBM is shown by its ability to model any data distribution given to it provided it is given a suitable initial parameter. Through trial and error, the initial parameter setting for CRBM to perform optimally for the different data sets has been identified.

5.2 FUTURE WORK & RECOMMENDATION

As a continuation of this project, the CRBM will be implemented in hardware, as currently, only simulation has been done on the CRBM MATLAB implementation. Hardware implementation can be done on FPGA using any hardware description language such as VHDL or Verilog. Hardware implementation is a natural extension of this project as the ultimate goal of the project is to implement CRBM in an LOC or SOC application.

The CRBM also needs to be tested with more real-world data-sets. Currently, the CRBM are trained on fairly simple data distribution that reflects data sets that is expected to be seen in real world application. More complex training data sets are needed to test CRBM and show its full potential as an ANN.

REFERENCE

[1] Aguilo, J., Millan, J., and Villa, R., **"Micro and nano technologies in medical applications: a challenge"**. Proceedings of the International Semiconductor Conference, Sinaia, Romania, 2001, Vol. 1, pp. 247–255

[2] Zhou, G.X., **"Swallowable or implantable body temperature telemeter - body temperature radio pill"**. Proceedings of 15th Annual Northeast Bioengineering Conference, Boston, MA, USA, 1989, pp. 165–166

[3] Evans, D.F., Pye, G., Bramley, R., Clark, A.G., Dyson, T.J., and Hardcastle, J.D., **"Measurement of gastrointestinal pH profiles in normal ambulant human subjects"**, Gut, 1988, 29, pp. 1035–1041

[4] Jobst, G., Urban, G., Jachimowicz, A., Kohl, F., Tilado, O., Lettenbichler, I., and Nauer, G., **"Thin film Clark-type oxygen sensor based on novel polymer membrane systems for in vivo and biosensor applications"**, Biosens. Bioelectron., 1993, 8, pp. 123–128

[5] Mackay, S., **"Radio telemetering from within the body"**, Science, 1961, 134, pp. 1196–1202

[6] Haykin, S., "**Neural Networks: A Comprehensive Foundation**", *Prentice Hall, New Jersey, 1999.*

[7] Wikipedia, "**Artificial Neural Network**", http://en.wikipedia.org/wiki/Artificial_neural_network

[8] Graupe, D., "**Principles of Artificial Neural Networks**" *World Scientific Publication, Singapore, 1997.*

[9] Chen, H., "**Continuous-valued Probabilistic Neural Computation in VLSI**", *PhD Thesis, Edinburgh University, UK, 2004.*

[10] Wikipedia, "**Artificial Neuron**", http://en.wikipedia.org/wiki/Artificial_neuron

[11] Kartapoulos, S. V., "**Understanding Neural Networks and Fuzzy Logic: Basic Concepts and Applications**", *IEEE Press, New York, 1996.*

[12] Chen, H. and Murray, A.F., "**A Continuous Restricted Boltzmann Machine with a Hardware-Amenable Learning Algorithm**", *Proceedings of the International Conference on Artificial Neural Networks (ICANN2002), p358-363, Madrid, Spain, 2002.* [Online] Available: http://www.ee.nthu.edu.tw/~hchen/pubs/icann2002.pdf

[13] Living La Vida LOC(a): A Brief Insight into the World of "Lab on a Chip" and Microfluidics, http://www.scq.ubc.ca/?p=621

[14] System on a Chip (SOC), http://www.siliconfareast.com/soc.htm, 2005

APPENDICES

Appendix I – Continuous Restricted Boltzmann Machine MATLAB Code

```
% Main code

Ca = [-0.20 -0.0 0.40 0.20];
Cb = [0.20 -0.40 0.0 -0.20];
y1 = CRBM(1000,Ca,Cb,1,0,1,1,0,0,0,0,4);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% CRBM()

function y = CRBM(figid, Ca, Cb, sw1, sw2, sw3, sw4, sd1, sd2, sd3, sd4, num_hid)

num_iter = 10000;
stddev_sample = 0.02;
stddev_sensor_min = 0.02;
stddev_sensor_max = 0.3;
initial_weight = 0.1;
num_sample = 100;

x = [-0.40 -0.40];
y = [0.30 0.20];

[concentration_a, concentration_b, visA, visB] = sensormodela_2cluster(num_sample,
stddev_sample, stddev_sensor_min, stddev_sensor_max, x, y, 1, 1, 1, 1, 0, 0, 0, 0);


[figid] = sensor_plot(visA, visB, figid);

% Global variables
num_vis = size(visA, 2);
num_hid = 4;
mui = 0.02;           % learning rate for weight vector of cluster = 0.02
kv = 1;               % learning rate for 'a' of visible units
kh = 1;               % learning rate for 'a' of hidden units
phiL = -1;
phiH = 1;
bh = 0.1;            % std dev of Gaussian noise for hidden units
bv = 0.1;             % std dev of Gaussian noise for visible units

% Initialise the weight
vishid = initial_weight.*randn(num_vis,num_hid);
ah = ones(1,num_hid);
av = 6*ones(1,num_vis); %original value 6

% Mixing two learning data alternatively
for i=1:(num_sample*2)
    if rem(i,2)
```

```
        vis(i,:) = visA(fix(i/2)+1,:);
    else
        vis(i,:) = visB(i/2,:);
    end
end
% Learning stage for CRBM

for i=1:num_iter
    hid = infer(vis, vishid, ah, bh, phiL, phiH);
    fvis = generate(hid, vishid, av, bv, phiL, phiH);
    fhid = infer(fvis, vishid, ah, bh, phiL, phiH);

    dvishid = mui*(gradient(vis,hid) - gradient(fvis,fhid));
    dav = gradA(vis, fvis, av, kv);
    dah = gradA(hid, fhid, ah, kh);

    % Update parameters
    vishid = vishid + dvishid;
    av = av + dav;
    ah = ah + dah;

    % Record parameters' change
    weight_change(i) = sum(sum(dvishid.^2));
    WEIGHT(:,:,i)=vishid;
    AV(i,:) = av;
    AH(i,:) = ah;
end

% Un-mixing the clusters' result
for i=1:(num_sample*2)
    if rem(i,2)
        fvisA(fix(i/2)+1,:) = fvis(i,:);
        fhidA(fix(i/2)+1,:) = fhid(i,:);
    else
        fvisB(i/2,:) = fvis(i,:);
        fhidB(i/2,:) = fhid(i,:);
    end
end

figid = aplot(AV, figid);
figid = aaplot(AH, figid);
figid = bplot(WEIGHT,figid);

figid = figid + 1;

% Examine the learning success with n-step reconstruction
figid = exam(num_vis,vishid,av,ah,bv,bh,phiL,phiH,figid);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% sensormodela_2cluster()

function [concentration_a, concentration_b, visA, visB] = sensormodela(num_sample,
stddev_sample, stddev_sensor_min, stddev_sensor_max, Ca, Cb, sensorOneWork,
sensorTwoWork, sensorThreeWork, sensorFourWork, sensorOneDrift,
sensorTwoDrift,sensorThreeDrift, sensorFourDrift)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Objectives:
% 1. To have two distinct sets of chemicals.
% 2. To have non-separable individual measurements due to the
%    cross-correlation between the measurands.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Variables
num_sensor = 2;
num_cluster = 2;

concentration_a = 3*randn(num_sensor,num_sample);
concentration_a(1,:) = Ca(1,1)+(stddev_sample*concentration_a(1,:));
concentration_a(2,:) = Ca(1,2)+(stddev_sample*concentration_a(2,:));



%% This is for symmetric gaussian distrubtion;
% concentration_b =randn(num_sensor,num_sample);
% concentration_b(1,:) = Cb(1,1)+(stddev_sample*concentration_b(1,:));
% concentration_b(2,:) = Cb(1,2)+(stddev_sample*concentration_b(2,:));
% %concentration_b(3,:) = Cb(1,3)+(stddev_sample*concentration_b(3,:));
% %concentration_b(4,:) = Cb(1,4)+(stddev_sample*concentration_b(4,:));


%% This is for non-symmetric gaussian distrubtion;
concentration_b =0.7*randn(num_sample,num_sensor);
sigma=[0.002 0.15;-0.15 0.1];
concentration_b=concentration_b*sigma;
concentration_b(:,1) = Cb(1,1)+(concentration_b(:,1));
concentration_b(:,2) = Cb(1,2)+(concentration_b(:,2));
concentration_b=concentration_b';


% Sensors' parameters

% For non-trivial problem, assume the linearity is poor and cross-correlation is strong
first_order =[0.1 0; 0 0.1];
second_order = [0 0.1;0.8 0];

sensor_noise_a = cat(1,stddev_sensor_min*randn(num_sensor-
1,num_sample),stddev_sensor_max*randn(1,num_sample));
```

```
sensor_noise_b = cat(1,stddev_sensor_min*randn(num_sensor-
1,num_sample),stddev_sensor_max*randn(1,num_sample));

% Sensors' equationsstddev_sensor*randn(num_sensor,1)
for I=1:num_sample,
    Sa(:,I) = (first_order*concentration_a(:,I)) +
nonzeros(eye(num_sensor,num_sensor).*(second_order*(concentration_a(:,I)*concentration_a
(:,I)'))) + sensor_noise_a(:,I);
    Sb(:,I) = (first_order*concentration_b(:,I)) +
nonzeros(eye(num_sensor,num_sensor).*(second_order*(concentration_b(:,I)*concentration_b
(:,I)'))) + sensor_noise_b(:,I);
end

% Additional time-drifting components, if sensor is working

if sensorOneWork == 1
    Sa(1,:) = Sa(1,:) + sensorOneDrift*ones(1,num_sample);
    Sb(1,:) = Sb(1,:) + sensorOneDrift*ones(1,num_sample);
else
    Sa(1,:) = zeros(1,num_sample);
    Sb(1,:) = zeros(1,num_sample);
end

if sensorTwoWork == 1
    Sa(2,:) = Sa(2,:) + sensorTwoDrift*ones(1,num_sample);
    Sb(2,:) = Sb(2,:) + sensorTwoDrift*ones(1,num_sample);
else
    Sa(2,:) = zeros(1,num_sample);
    Sb(2,:) = zeros(1,num_sample);
end

visA=(cat(2,ones(num_sample,1),concentration_a'));
visB=(cat(2,ones(num_sample,1),concentration_b'));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

% sensor_plot()

```
function [figid] = sensor_plot(Sa, Sb, figid)

figure(figid);
title('Sensor inputs');

plot(Sa(:,2),Sa(:,3),'ro');
hold on; xlabel('data_i_n_p_u_t x'); ylabel('data_i_n_p_u_t y');
axis([-1 1 -1 1]);
plot(Sb(:,2),Sb(:,3),'ro');

figid = figid + 1;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% infer()

function hid = infer(vis, vishid, ah, bh, phiL, phiH)

vs = size(vis);
ws = size(vishid);
noise = randn(vs(1), ws(2));

hid = sigmoid(repmat(ah,vs(1),1).*(vis*vishid+bh*noise), phiL, phiH);
hid(:,1) = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% generate()

function vis = generate(hid, vishid, av, bv, phiL, phiH)

hs = size(hid);
ws = size(vishid);
noise = randn(hs(1),ws(1));

vis = sigmoid(repmat(av,hs(1),1).*(hid*vishid'+bv*noise), phiL, phiH);
vis(:,1) = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% gradA()

function da = gradA(stage1, stage2, a, learning_rate)


nfac = size(stage1);                % Size of data set

x = diag((stage1-stage2)'*(stage1+stage2))';

da = learning_rate*x./(a.^2);
da = da/nfac(1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% aplot()

function [figid] = aplot(a, figid)

n = size(a,2);
```

```
figure(figid);
hold off;

for i=2:n
    plot(a(:,i));
    hold on; xlabel('epoch'); ylabel('a_v_i_s_i_b_l_e');
end

figid = figid + 1;
```

%%

```
% aaplot()
```

```
function [figid] = aaplot(a, figid)

n = size(a,2);

figure(figid);
hold off;

for i=2:n
    plot(a(:,i));
    hold on; xlabel('epoch'); ylabel('a_h_i_d_d_e_n');
end
figid = figid + 1;
```

%%

```
% bplot()
```

```
function [figid] = bplot(a, figid)

n = prod(size(a(:,:,1)));

figure(figid);
hold off;

for i=1:n
    b=permute(a,[3,2,1]);
    plot(b(:,i));
    hold on; xlabel('epoch'); ylabel('weight');
end

figid = figid + 1;
```

%%

```
% exam()
```

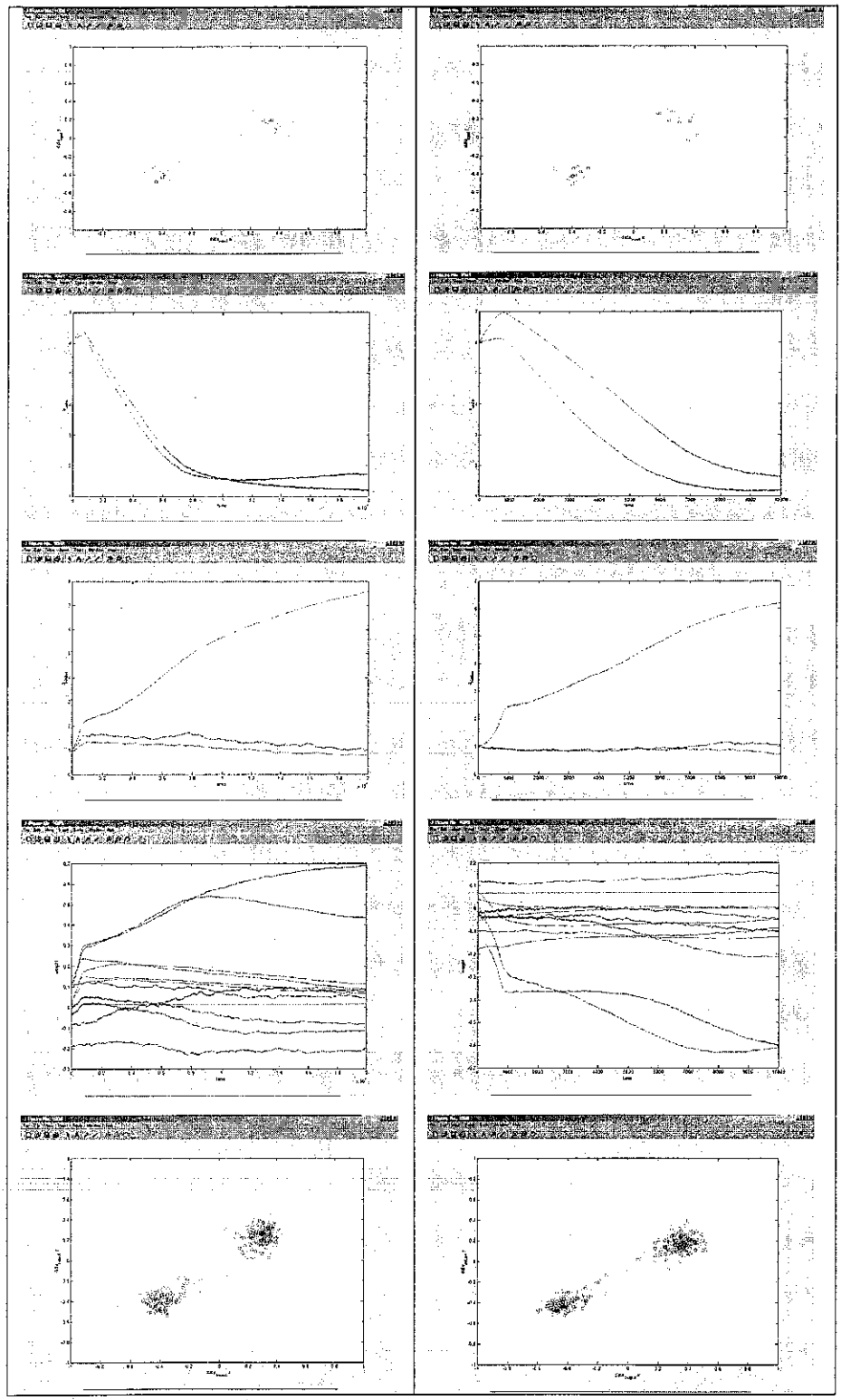
```
function [figid] = exam(num_vis,vishid,av,ah,bv,bh,phiL,phiH,figid)
```

```
% Examine learning ability of the network with normal distributed noise
num_sample = 400;
nstep = 20;
l = 0;

vis_test = cat(2,ones(num_sample,1),2*rand(num_sample,num_vis-1)-
ones(num_sample,num_vis-1));

for i=1:num_sample
    if rem(i,2)
        visA(fix(i/2)+1,:) = vis_test(i,:);
    else
        visB(i/2,:) = vis_test(i,:);
    end
end

hid_test = infer(vis_test, vishid, ah, bh, phiL, phiH);

for i=1:nstep
    vis_test = generate(hid_test, vishid, av, bv, phiL, phiH);
    hid_test = infer(vis_test, vishid, ah, bh, phiL, phiH);

end

figid = figid + 1;

figure(figid);
title('Output of CRBM');

plot(vis_test(:,2),vis_test(:,3),'ro');
xlabel('data_o_u_t_p_u_t x'); ylabel('data_o_u_t_p_u_t y');
axis([-1 1 -1 1]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

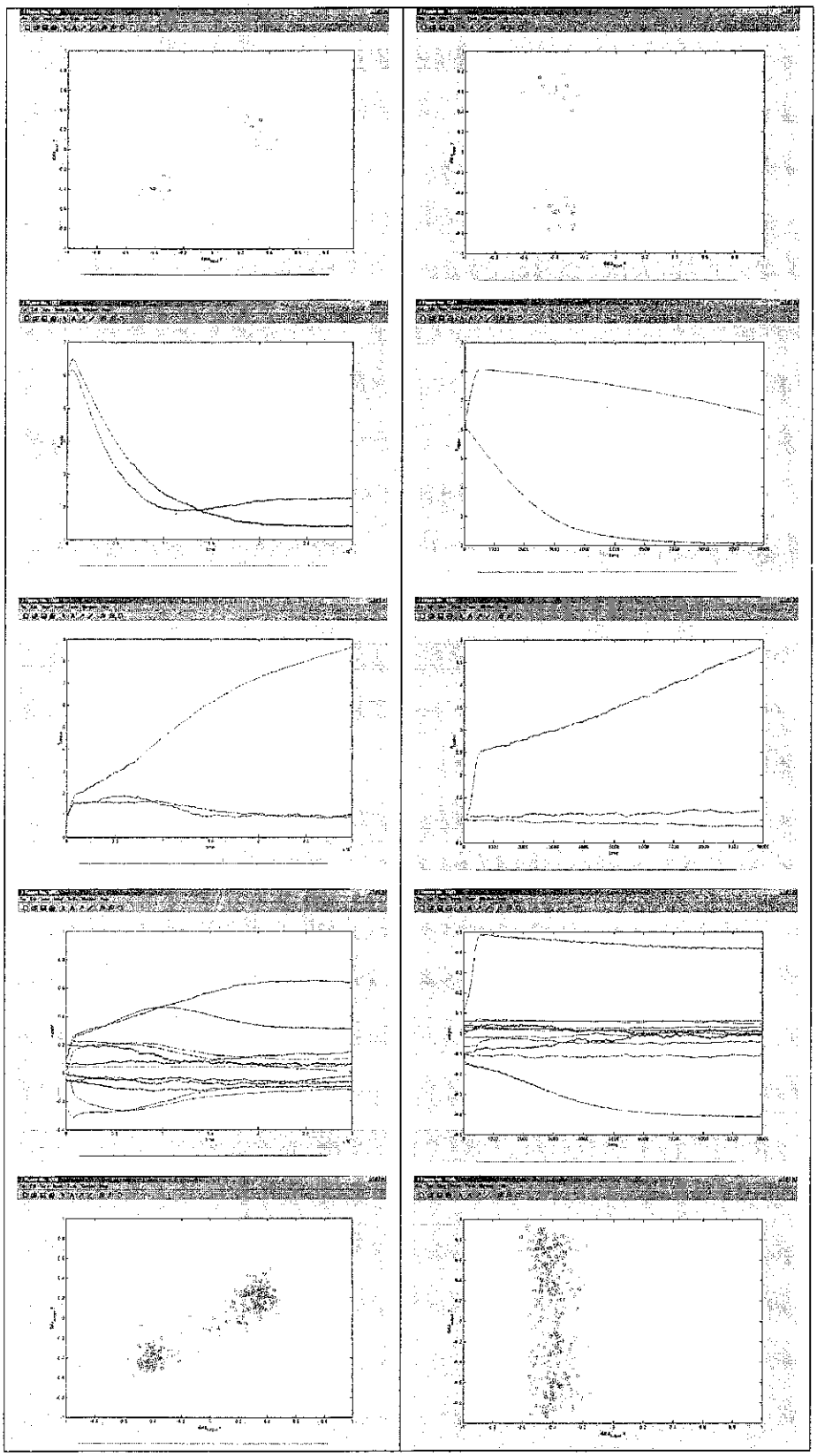
% sigmoid()

```
function y = sigmoid(x, phiL, phiH)
y = phiL + (phiH-phiL)./(1+exp(-x));
```

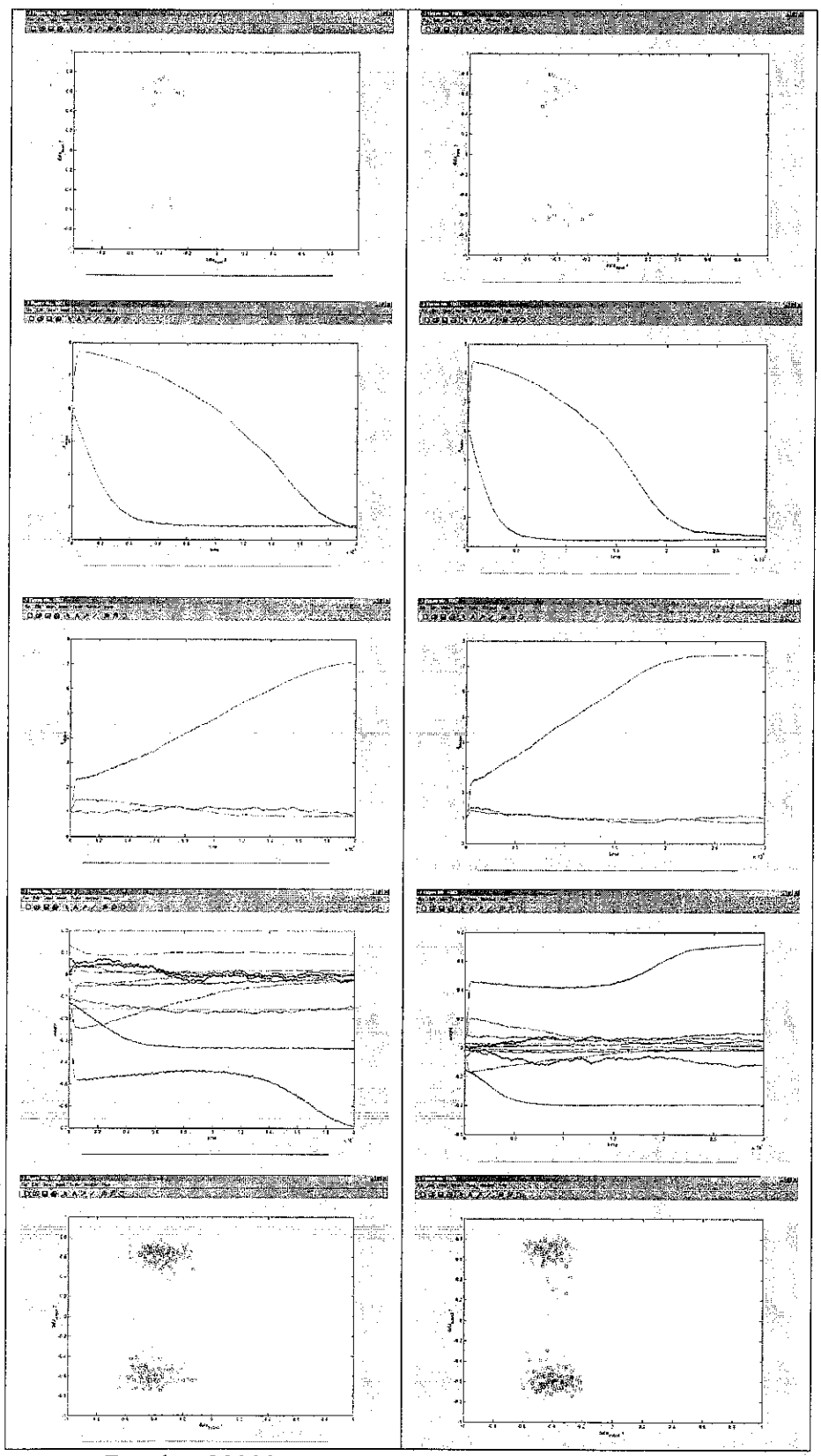
Appendix II - Results of Testing



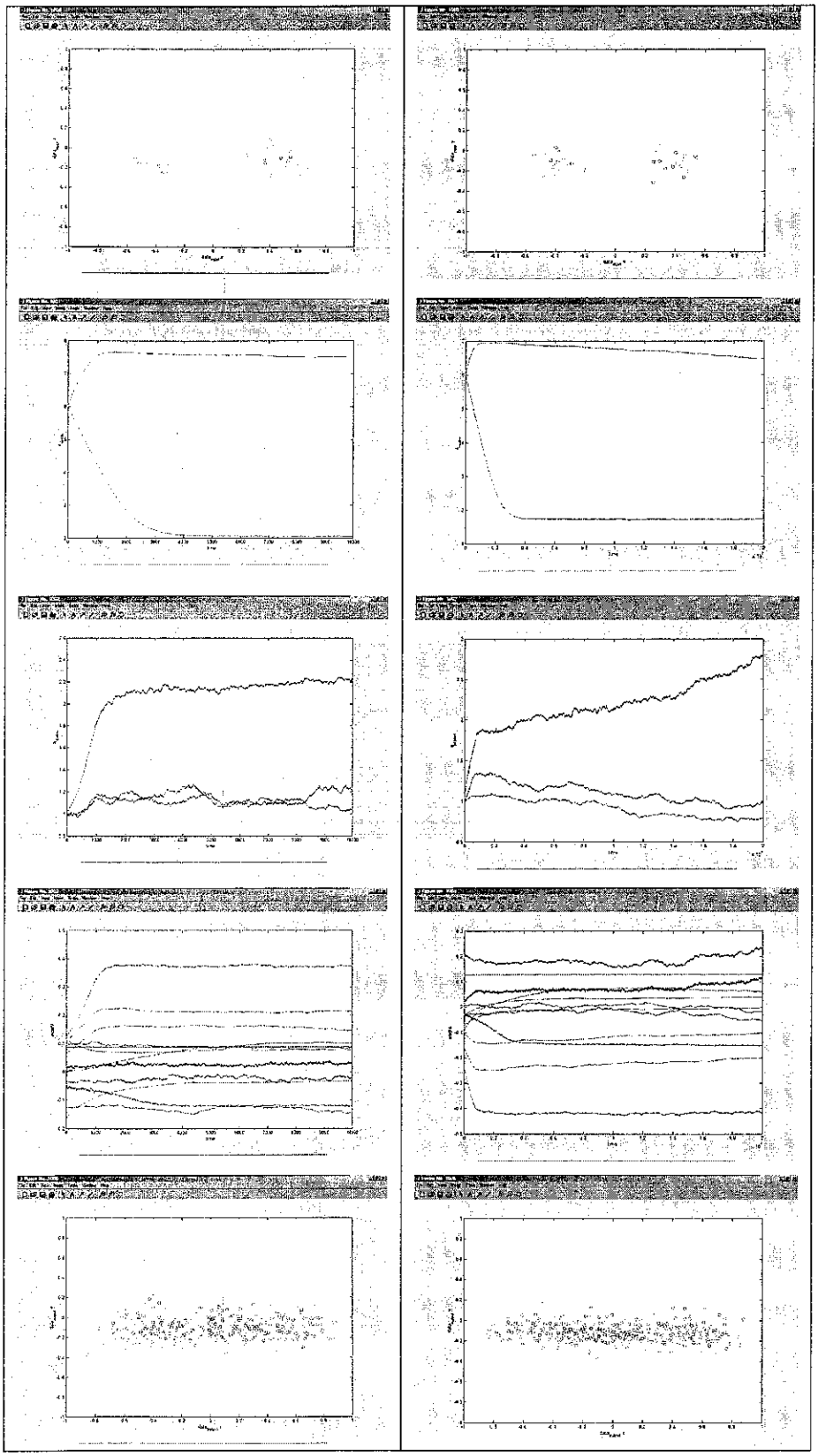
Epoch = 10000 Epoch = 20000



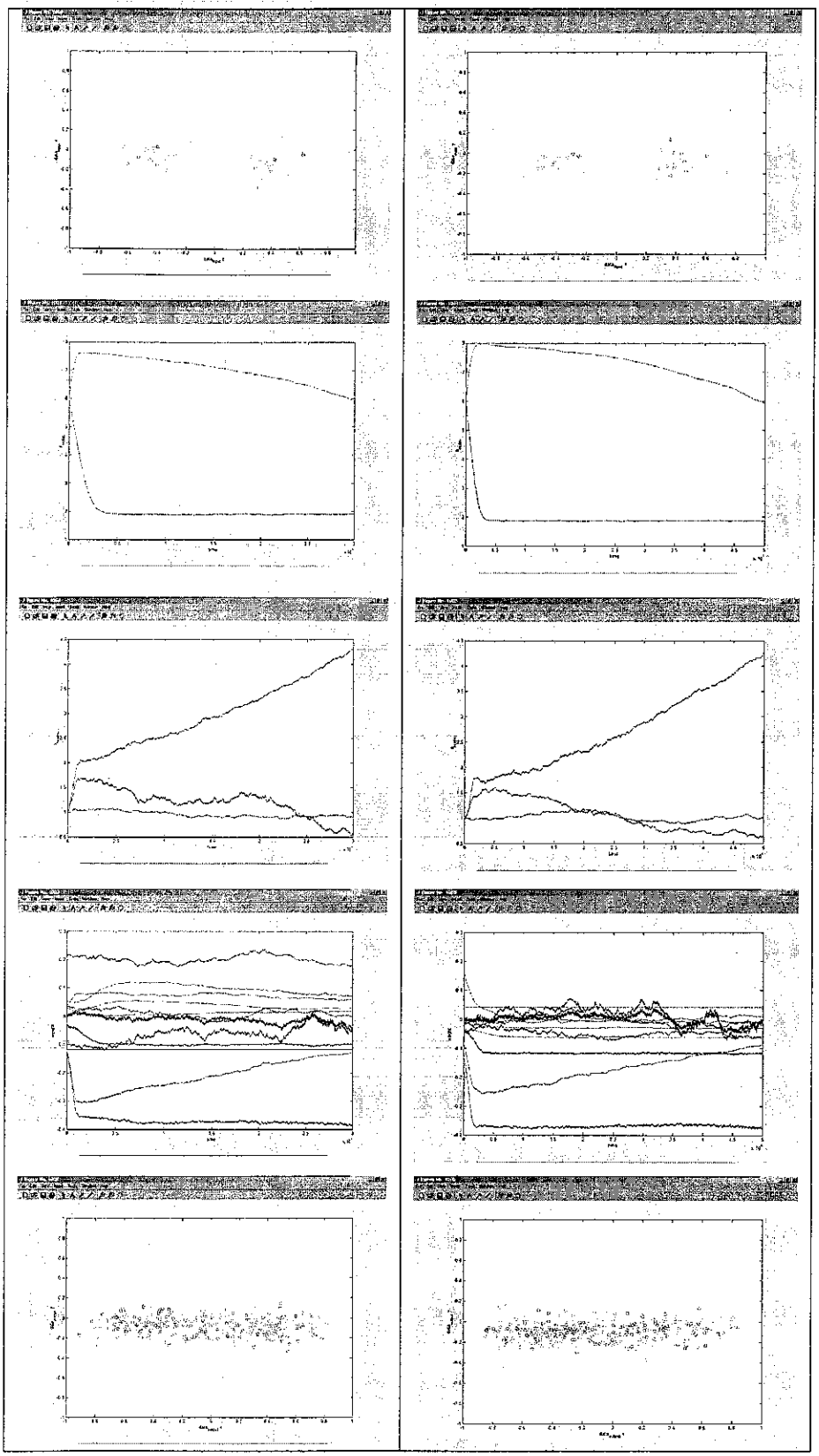
Epoch = 30000 Epoch = 10000



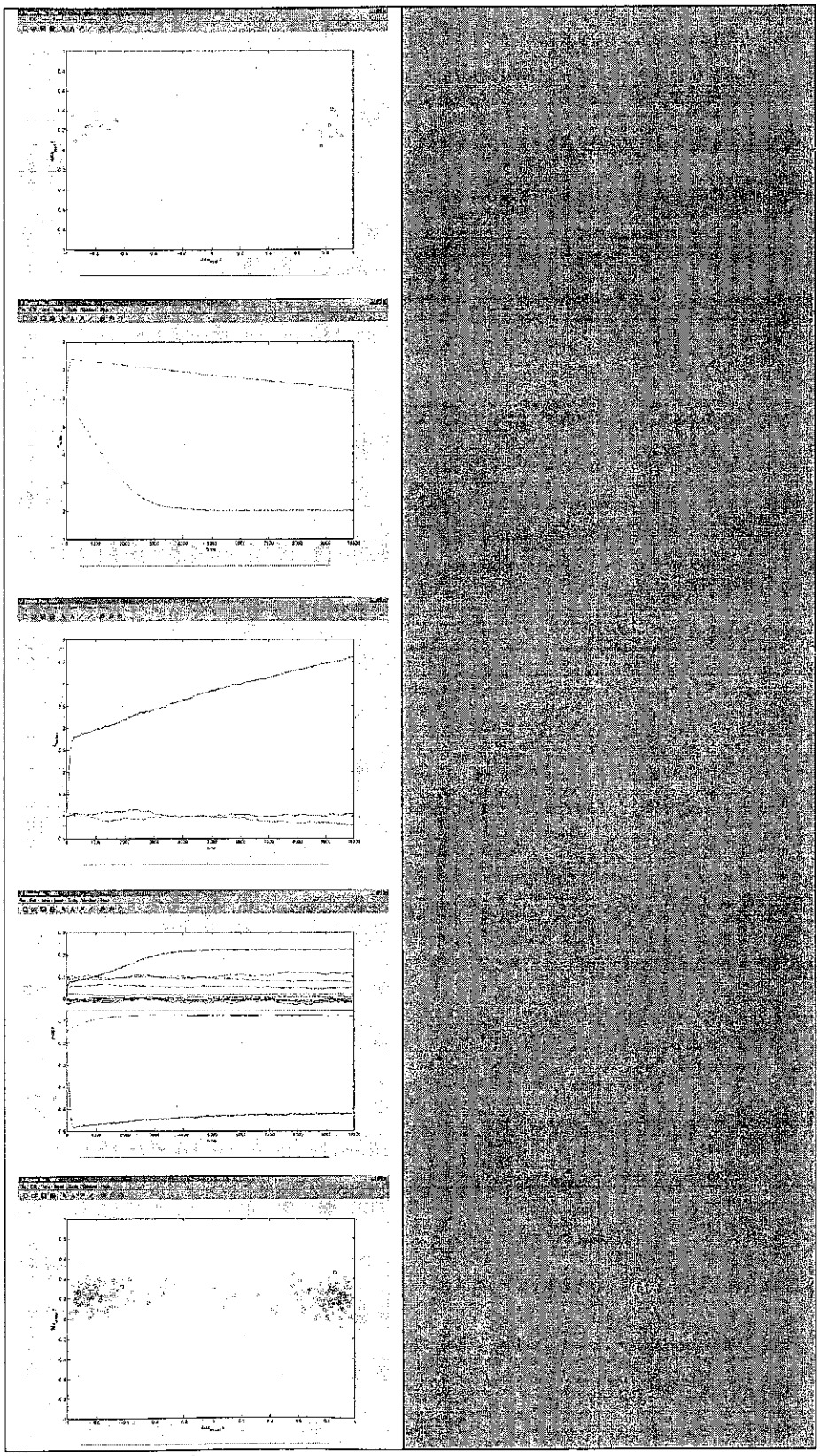
Epoch = 20000 Epoch = 30000



Epoch = 10000 Epoch = 20000



Epoch = 30000 Epoch = 50000



Epoch = 10000