2003

The window least mean square error algorithm

Anna Semenovna Degtyarena

Follow this and additional works at: https://scholarworks.lib.csusb.edu/etd-project

Part of the Theory and Algorithms Commons

Recommended Citation
https://scholarworks.lib.csusb.edu/etd-project/2385

This Project is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.
THE WINDOW LEAST MEAN SQUARE ERROR ALGORITHM

A Project
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Anna Semenovna Degtyarena

June 2003
THE WINDOW LEAST MEAN SQUARE ERROR ALGORITHM

A Project
Presented to the
Faculty of
California State University,
San Bernardino

by
Anna Semenovna Degtyarena

June 2003

Approved by:

Dr. George Georgiou, Chair, Computer Science

Dr. Kerstin Voigt

Dr. Owen Murphy

Date 6/2/03
ABSTRACT

In this research basic LMS rule is investigated. The research shows that the basic LMS learning rule has pitfalls - it does not work well and efficiently in all cases.

In order to improve the performance of LMS algorithm by decreasing the amount of calculations this research proposes to make an update on each step only for those elements from the input data set, that fall within a small window W near the separating hyperplane surface. The size of this window is a variable parameter of the classification procedure and may be specified during the learning process. Basically it means to exclude the outlier elements from the input pattern from further consideration.

This work is aimed to describe in detail the results that can be achieved by using the proposed LMS with window learning algorithm in information systems that employs the methodology of neural network for the purposes of classification.

The present research may be divided into several parts. The first part presents the detailed analysis of basic LMS learning algorithm and its use during the
learning process in neural network information models of data classification.

In the second part of research the comparison of efficiency of basic LMS learning algorithm and proposed LMS learning algorithm with window was performed on real data sets.

From analyzing the results it can be inferred that Window algorithm has less number of misclassifications then LMS algorithm. In other words Window algorithm is more efficient then LMS algorithm. Detailed analysis of the results is presented in the third part of the research.
ACKNOWLEDGMENTS

I wish to express my profound gratitude to Dr. Georgiou for the guidance and technical expertise that he provided to me during the course of this research and for the enormous contribution that he made in order to make my thesis a readable document. Also I want to thank him for his patience and understanding that allowed me to complete this work. I am also thankful to Dr. Voigt and Dr. Murphy for the time that they spent reading and correcting my work.

The support of the National Science Foundation under award 9810708 is gratefully acknowledged.
TABLE OF CONTENTS

ABSTRACT ................................................................. iii
ACKNOWLEDGMENTS ................................................... v
LIST OF TABLES ........................................................ viii
LIST OF FIGURES ......................................................... ix
CHAPTER ONE: INTRODUCTION ................................. 1
  1.1 The Elements of Artificial Networks ............. 5
  1.1.1 Models of Artificial Neurons ................. 5
  1.1.2 Types of Activation Functions .......... 9
  1.2 Perceptron Rule .......................................... 12
  1.3 Pocket Perceptron Rule ......................... 17
  1.4 Least Mean Square Error (LMS) Rule .... 18
CHAPTER TWO: DETAILED VIEW ............................... 21
  2.1 Detailed View of LMS .............................. 21
  2.1.1 LMS Learning Algorithm .................. 21
  2.2 LMS Shortcomings .................................. 32
CHAPTER THREE: NEW RULE ................................ 35
  3.1 The New Rule ........................................ 35
  3.2 Advanced LMS with Window Learning Law
       Implementation ........................................ 37
CHAPTER FOUR: METHODOLOGY .............................. 40
  4.1 Methodology of the Research .................. 40
  4.2 The Methodology of the Algorithm
       Implementation ........................................ 42
CHAPTER FIVE: RESULTS

5.1 Testing ................................................. 45

CHAPTER SIX: CONCLUSION ................................. 53

APPENDIX A: PROGRAM SOURCE CODE .................... 56

APPENDIX B: GRAPHICAL REPRESENTATION OF RESULTS .... 65

REFERENCES ...................................................... 216
LIST OF TABLES

Table 1. Number of Misclassified Vectors on Iris Data. Number of Iterations 1000. Class 1 and 2 .................................................. 47

Table 2. Number of Misclassified Vectors on Iris Data. Class 1 and 3. Number of Iterations 1000 ................................. 48

Table 3. Number of Misclassifications for Non Overlapped Randomly Created Classes. Number of Iterations 30000 .................. 49

Table 4. Number of Misclassified Vectors for Overlapped Randomly Created Classes. Number of Iterations 30000 .................. 50
LIST OF FIGURES

Figure 1. Three Basic Graphical Representations of a Single p-Input (p-Synapse) Neuron .................................. 6

Figure 2. The Single Neuron with a Biasing Input ................................................. 8

Figure 3. Dendritic, Signal-Flow, Block-Diagrams .............................................. 11

Figure 4. 1 and 2 Layer Perceptron ................................................................. 15

Figure 5. Block-diagram .......................................................... 30

Figure 6. Dendritic Structure of an Adaline b ................................................. 30

Figure 7. Detailed Structure of an $i_{th}$ Synapse Implementing the LMS Learning ................................................. 31

Figure 8. Graphical Representation of Outliers ................................................. 34

Figure 9. Graphical Representation of Input Classes ........................................... 36

Figure 10. Graphical Representation of the Results. Iris Data. Class 1 and 2 ........... 47

Figure 11. Graphical Representation of the Results. Iris Data. Class 1 and 3 ........... 48

Figure 12. Graphical Representation of the Results. Iris Data. Class 3 and 2 ........... 49

Figure 14. Graphical Representation of the Results. Overlapped Randomly Created Classes ................................................. 50
CHAPTER ONE

INTRODUCTION

The history of Neural Network theory began in 1890 with the publication of a theory on brain function by American psychologist William James [3]. According to his theory the brain was an unstructured randomly connected web of fibers, which propagated (reverberated) electrical currents in all directions in a fashion analogous to a network of metal wires. William James proposed that all thoughts and actions are produced because of these currents flowing from regions (brain-processes) that have an excess of electrical charge to regions with a deficit of electrical charge [1]. The intensity of the all those thoughts and actions are proportional to the current flow rate, which in turn is proportional to the difference in charge between the two regions. Later these reverberating electrical currents would be called engrams [1]. This theory gave to the world the new definition of learning in terms of neural networks: Learning consisted of changing these current paths or forming new paths by using the following rule:
"When two elementary brain-processes have been active together or in immediate succession, one of them, on reoccurring, tends to propagate its excitement into the other." (James 1890, p.566)

According to the theory of William James, the efficiency of the transmission between stimulus and neuron's output should increase with time because the skill is learned but the first experimental test found just the opposite.

This experiment was done by Charles Sherrington who made the next significant breakthrough in 1898. He found that spinal neurons in cats reduce their efficiency with low intensity use instead of increasing their efficiency as expected. He had found a phenomenon known today as habituation.

But there was still no theory that could oppose the theory of William James until 1938 when N. Rashevsky proposed that the brain could be organized around binary logic operations since all actions could be viewed as binary (true or false) values. He even presented the circuit, showing how a binary logic EXCLUSIVE OR operation could be implemented using addition and subtraction operators.
Neural network research really only became possible with the computer age when ideas could be validated by simulation on various types of electronic calculators. The idea for these simulations was provided by Donald Hebb of McGill University in Canada, who in 1949 proposed this unidirectional variation of the William James learning rule [2]:

"Let us assume then that the persistence of repetition of a reverberatory activity (or trace) tends to induce lasting cellular changes that add to its stability. The assumption can be precisely stated as follows: 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 on one or both cells so that A's efficiency as one of the cells firing B is increased" (Hebb 1949, p 62)

According to the modern point of view the Neural Networks are one of a group of intelligence technologies for data analysis that differ from other classical analysis techniques in that they learn about the chosen subject from the data provided to them, rather than being programmed by the user in a traditional sense. Neural networks gather their knowledge by detecting the patterns
and relationships in data, learning from those relationships and adapting their response. Sometimes in order to perform a particular task it is could be necessary to break it down into blocks in order to make it easier to handle. This process may be compared for example to hi-fi stereo system [3] that consists of several blocks, and some of those blocks, like CD players, are dedicated to a particular task and others, like an amplifier, have a more general function. In a analogies way the process of building a software application is much easier if it is broken into a set of smaller parts that can be easily implemented. It is also beneficial to have general-purpose tools that can connect the parts together to produce the application. In many applications a neural network can form one of those blocks. The range of functions that neural networks can perform is very large, and some of them are made possible only by neural networks.

In building an application, the standard method is to break the overall function into blocks of code. These blocks receive input data from outside the system (user input) or from other blocks, perform some processing depending on the code written into the block and then
output data to other blocks or to outside the system (system output). The processing or procedure that is written into each block is often called an algorithm. Programmers have access to many standard algorithms that they can incorporate into their programs for such tasks as computing the difference between two dates or converting one currency into another. However, sometimes there is no algorithm available for functional blocks and creating one may be very time consuming. A key benefit of neural networks is that programmer can use them to build a model of the system or subject he is interested in on the basis of the provided data. One can know only the inputs and outputs that are important but may not know what happens internally [3]. The neural network will model this system from the provided data. Therefore, Neural Networks are powerful solutions to many problems.

1.1 The Elements of Artificial Networks

1.1.1 Models of Artificial Neurons

According to modern theory of artificial neural networks they may be determined as nonlinear information (signal) processing devices, which are built from interconnected elementary processing devices, called
neurons, where an artificial neuron is a p-input single-output signal-processing element, which can be thought of as a simple model of a non-branching biological neuron. Graphically, an artificial neuron is represented in one of the following forms [3]:

\[ v = w_1 \cdot x_1 + \ldots + w_p \cdot x_p = w \cdot x \]

\[ y = \phi(v) \]

a. Dendritic representation

\[ x = [x_1, x_2, \ldots, x_p]^T \]

\[ w = [w_1, w_2, \ldots, w_p] \]

b. Signal flow graph

\[ x = [x_1, x_2, \ldots, x_p] \]

\[ w = [w_1, w_2, \ldots, w_p] \]

Figure 1. Three Basic Graphical Representations of a Single p-Input (p-Synapse) Neuron

Each artificial neuron consists of p synapses arranged either along a dendrite, which aggregates the synaptic activities, or in a layer of branches, which link input nodes with a summing node.
A p-element column vector of input signals represents the pre-synaptic activities.

\[ x = [ x_1 \ldots x_p ]^T \]

In other words to say the space of input patterns is p-dimensional.

Synapses are characterized by adjustable parameters called weights or synaptic strength parameters. The weights are arranged in a p-element row vector:

\[ w = [ w_1 \ldots w_p ] \]

Passing through synapses and a summing node, input signals are combined into the activation potential, which describes the total post-synaptic activity. The activation potential is formed as a linear combination of input signals and synaptic strength parameters, that is, as an inner product of the weight and input vectors:

\[ v = \sum_{i=1}^{p} W_i X_i = w \cdot x = [W_1 W_2 \ldots W_p] \cdot [X_1 \ldots X_p] \]

Consequently, the total post-synaptic activity is passed through so-called an activation function, \( \varphi(\cdot) \), in order to generate the output signal
\[ y = \phi(v) \]

The activation function is typically a saturating function. It normalizes the total post-synaptic activity to the standard values of output signal.

A synapse is called excitatory, if a corresponding weight is positive, \( w_i > 0 \), and inhibitory, if a corresponding weight is negative, \( w_i < 0 \).

Sometimes so-called threshold, \( \theta \) or bias \( b = -q \) may be added as an additional parameter for convenience. Mathematically it can be done by fixing one input signal to be constant. Then we have

\[ x_p = +1 \]
\[ w_p = b = -q \]

After this modification, the activation potential is calculated as follows:

\[ v = \sum_{i=1}^{p} w_i x_i = \hat{v} - \theta, \quad \hat{v} = \sum_{i=1}^{p-1} w_i x_i, \]

where \( [\hat{v}] \) is the augmented activation potential.

![Figure 2. The Single Neuron with a Biasing Input](image-url)
1.1.2 Types of Activation Functions

Typically, the activation function is aimed to generate either unipolar or bipolar signals. There are many types of activation functions. Let us describe several, most often used of them [2].

A linear function

\[ y = v \]

Such linear processing elements (often called ADALINES), are studied in the theory of linear systems. Among different kind of examples the "traditional" signal processing and statistical regression analysis may be presented.

A step function

Unipolar step function

\[ y = \varphi(x) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases} \]

Such kind of processing element is traditionally called Perceptron. It works as a threshold element with a binary output [4].
Bipolar step function

\[
y(x) = \begin{cases} 
+1 & \text{if } v \geq 0 \\
-1 & \text{if } v < 0 
\end{cases}
\]

A step function with bias

In the case when bias or threshold is added to both, unipolar and bipolar step function, we deal with so-called step functions with bias. We then can say that a neuron is "fired", when the synaptic activity exceeds the threshold level, q. The following is the example of unipolar step function with bias:

Unipolar step function with bias

\[
y = \varphi(x) = \begin{cases} 
1 & \text{if } w \cdot x \geq \Theta \\
0 & \text{if } w \cdot x < \Theta 
\end{cases}
\]

Actually, neurons in the most practical cases neurons are arranged into a layer of neurons. Subsequently, it may be thought of as a p-input single layer neural network consisting of m neurons. Similarly
to a single neuron, the network can be represented in all three basic forms: dendritic, signal-flow, and block-diagram [4].

The dendritic representation of the neural network helps to represent a layer of neurons by \( m \times p \) matrix \( W \) of synaptic weights. Each row of the weight matrix is associated with one correspondent neuron.

![Dendritic representation](image)

\[
  x = [x_1, x_2, \ldots, x_p]^T
\]

\[
  W = \begin{bmatrix}
    w_{11} & w_{12} & \cdots & w_{1p} \\
    w_{21} & w_{22} & \cdots & w_{2p} \\
    \vdots & \vdots & \ddots & \vdots \\
    w_{m1} & w_{m2} & \cdots & w_{mp}
  \end{bmatrix}
\]

![Signal-flow graph](image)

\[
  x = \begin{bmatrix}
    x_1 \\
    \vdots \\
    x_p
  \end{bmatrix}
\]

\[
  W = \begin{bmatrix}
    w_{11} & w_{12} & \cdots & w_{1p} \\
    w_{21} & w_{22} & \cdots & w_{2p} \\
    \vdots & \vdots & \ddots & \vdots \\
    w_{m1} & w_{m2} & \cdots & w_{mp}
  \end{bmatrix}
\]

![Block-diagram](image)

\[
  x \xrightarrow[p]{} W \xrightarrow{\phi} y
\]

\( y \xrightarrow{\phi} y \)

Figure 3. Dendritic, Signal-Flow, Block-Diagrams
Mathematically, operations performed by the layer of neurons can be presented as follows:

\[
\begin{bmatrix}
V_1 \\
V_2 \\
\vdots \\
V_m \\
\end{bmatrix} =
\begin{bmatrix}
W_{11} & \cdots & W_{1p} \\
W_{21} & \cdots & W_{2p} \\
\vdots & \cdots & \vdots \\
W_{m1} & \cdots & W_{mp} \\
\end{bmatrix}
\begin{bmatrix}
X_1 \\
X_2 \\
\vdots \\
X_p \\
\end{bmatrix}
\]

or equivalently in a matrix form as:

\[
\begin{bmatrix}
Y_1 \\
Y_2 \\
\vdots \\
Y_m \\
\end{bmatrix} =
\begin{bmatrix}
\varphi(V_1) \\
\varphi(V_2) \\
\vdots \\
\varphi(V_m) \\
\end{bmatrix}
\]

where \( v \) is a vector of activation potentials.

From the signal-flow graph we can see that each synaptic strength parameter \( w_{ij} \) is now related to a connection between nodes of the input layer and the output layer. Therefore, the name connection strengths for the weights is also justifiable.

1.2 Perceptron Rule

The first model of artificial neural network was proposed by Frank Rosenblatt in 1958 [4]. He initiated a new phase in neural network research. The model was
called Perceptron. The basic idea of the Perceptron was to illustrate some fundamental properties of intelligent systems, such as incremental learning from examples - one of the two basic learning paradigms in neural network theory, which sometimes called supervised learning or learning with a teacher. Perceptron consists from one layer of artificial neurons, in which it accepts patterns from the outside world and supplies a result. The rule is presented with several input and output pairs. The list of those pairs is called the training set. When training input is applied to the network, it compares its actual output to the correct output, and then changes the connection strengths among neurons in order to minimize the difference. This is typically done incrementally, making small adjustments in response to each training pair, so that connection strengths converge to a solution in which the actual output matches the desired one.

The general algorithm of the Perceptron rule can be presented in the following way: the rule works by initializing the weights to some low value, not necessarily random. All the patterns are presented and for those patterns, that are incorrectly classified the degree of mismatch between the actual and desired output
is noted. All these mismatches are added together in some fashion (depending on the type of optimization procedure) to give the total error, which is to be minimized, and all the weights on the incorrectly responding templates are changed in proportion to the total error.

Rosenblatt summed up perceptrons in this passage from his 1962 book (page 28):

"Perceptrons are not intended to serve as detailed copies of any actual nervous system. They're simplified networks, designed to permit the study of lawful relationships between the organization of a nerve net, the organization of its environment, and the 'psychological' performances of which it is capable. Perceptrons might actually correspond to parts of more extended networks and biological systems; in this case, the results obtained will be directly applicable. More likely they represent extreme simplifications of the central nervous system, in which some properties are exaggerated and others suppressed. In this case, successive perturbation and refinements of the system may yield a closer approximation." [4]

The Perceptron Learning Rule is of central importance for supervised learning paradigm in neural
network theory. The most neural networks developed later include the Perceptron learning in one form or the other.

![1-layer Perceptron](image1)

![2-layer Perceptron](image2)

Figure 4. 1 and 2 Layer Perceptron

The Perceptron rule can be described by the following formulas [3]: the main idea is to classify the set of externally applied stimuli $x_1, x_2, ..., x_p$ into two classes $c_1$ or $c_2$. These classes are linearly separable if they lie on the opposite sides of a hyperplane and non-separable if they do not lie on the opposite sides of a hyperplane.
For two linearly separable classes $c_1$ and $c_2$ there is a weight vector $W$ such that

$W^T X \geq 0$ if $X \in c_1$ \hspace{1cm} (1)

$W^T X < 0$ if $X \in c_2$ \hspace{1cm} (2)

where $W^T$ - the transpose of the column weight vector $W$;

$X$ - the column input vector.

During Perceptron learning the elements of weight vector $W$ are changed according to a fixed incremental rule such that the weights on a misclassified pattern are changed by a fixed amount, proportional to the total error. This amount calculated by using the following updating formula such that equation (1) and (2) are satisfied.

$W(k+1) = W(k) + \alpha s_k X(k) \hspace{1cm} (3)$

$s_k = d_k - o_k \hspace{1cm} (4)$

where

$W$ - weight vector;

$X$ - input vector;

$\alpha$ - the learning rate - a small real positive value;

$d_k$ - desired output;

$o_k$ - actual output;

$s_k$ - error associated with the $X(k)$ input vector.
The original Perceptron rule has the following two major shortcomings:

- The Perceptron rule in some cases requires a large number of iterations.
- It is not stable in the case of overlapping classes.

1.3 Pocket Perceptron Rule

Certain attempts were made in order to overcome shortcomings of the Perceptron rule. Due to the excessive number of iterations and existence of unstable cases for overlapping classes in input data the Perceptron does not always produce the solution. The Pocket Perceptron algorithm was designed as a modification of Perceptron learning rule [6]. This algorithm produces correct target output (±1) on as many of its input patterns as possible. The basic idea of the Pocket algorithm is to run the Perceptron learning algorithm while keeping an extra set of weights "in its pocket" [6]. It helps to create a set of weights, which has had the longest unmodified run of successes so far. Pocket weights are replaced with Perceptron weights each time it have a better number of correct classification of the training data. The
algorithm stops after all data are completely classified or when some chosen time is expired (the maximum number of iteration are reached).

1.4 Least Mean Square Error (LMS) Rule

One of the most efficient ways to eliminate excessive number of iterations in separation the patterns is to use the delta rule or LMS[3]. Developed by Widrow and Hoff, the delta rule, also called the Least Mean Square (LMS) method, is one of the most commonly used learning rules in practice.

The main idea of this method may be described as follows [10]: For a given input vector, the output vector is compared to the correct answer. If the difference is zero, no learning takes place; otherwise, the weights are adjusted to reduce this difference [3]. The change in weight from $w_i$ to $w_j$ is given by:

$$\Delta w_{ij} = r \cdot a_i \cdot e_j,$$

where

- $r$ - learning rate;
- $a_i$ - activation for input $x_i$;
- $e_j$ - difference between the expected output and the actual output.
If the set of input patterns form a linearly independent set then arbitrary associations can be learned using the delta rule.

It is known that for networks with linear activation functions and with no hidden units (hidden units are found in networks with more than two layers), the error squared vs. the weight graph is a paraboloid in n-space. Since the proportionality constant is negative, the graph of the function is concave upward and has a minimum value. The vertex of this paraboloid represents the point where the error is minimized. The weight vector corresponding to this point is then the ideal weight vector.

The most valuable advantage of the delta learning rule is that these rule not only moves the weight vector nearer to the ideal weight vector, but it does so in the most efficient way. The delta rule implements a gradient descent by moving the weight vector from the point on the surface of the paraboloid down toward the lowest point, the vertex [7].

But there is a problem: the delta rule does not work in all cases. As stated previously, in the case of linear activation functions where the network has no hidden
units, the delta rule will always find the best set of weight vectors. On the other hand, that is not the case for hidden units. The error surface is not a paraboloid and so does not have a unique minimum point. On other words it is possible in the nonlinear case for the cost function to have a local minimum. In this case LMS rule can be stuck at the local minimum. There have been a number of theories in response to this problem. These include the generalized delta rule and the unsupervised competitive learning model.
2.1 Detailed View of LMS

2.1.1 LMS Learning Algorithm

In the previous few sections the decoding part of a neural network was described, assuming that the weight matrix, $W$, is already predefined. If the weight parameters in matrix $W$ are satisfactory determined, during the decoding process the neural network performs some useful task it has been designed to do.

In simple or specialized cases the weight matrix can be pre-computed; but in the most common case it is obtained through the so-called learning process. Thus, learning is a dynamic process of weight matrix modification that gear toward some desirable way [8].

Let us assume that there are two sets of patterns A and B, and this two sets can be separated by a hyperplane in pattern space, then it is possible to classify them using artificial neuron model. Analogically, any division of patterns using artificial neuron model must be such that the input classes may be separated by a hyperplane.
in pattern space. In both cases we can say that the classes are Linearly Separable [3].

The matter of fact, that a hyper-surface can be described in a (p+1)-dimensional space by function of p variables. Therefore, in order to describe the essentials of LMS algorithm we have to discuss several methods of approximating such a surface by a hyperplane using an artificial neuron model. Let us start with a bit more general problem, namely, approximation of such functions using p-input Adalines [3].

Let the function to be linearly approximated be known at N different points, \( x(n) \), and \( d(n) \) being a vector of values of the function.

Let us arrange the N points training patterns, as previously, in the following two matrices:

\[
X = \begin{bmatrix} x(1) & \ldots & x(n) & \ldots & x(N) \end{bmatrix} \text{ is } p \times N \text{ matrix}
\]

\[
D = \begin{bmatrix} d(1) & \ldots & d(n) & \ldots & d(N) \end{bmatrix} \text{ is } m \times N \text{ matrix}
\]

In order to perform an approximation of the stated above function let us consider a p-input m-output Adaline model, characterized by an \( m \times p \) weight matrix, \( W \), each row related to a single neuron.

For every input vector, \( x(n) \), the Adaline calculates the actual output vector as follows
\[ y(n) = W x(n) \quad (1) \]

The set of output vectors is arranged in an output matrix:
\[ Y = \begin{bmatrix} y(1) & \ldots & y(n) & \ldots & y(N) \end{bmatrix} \text{ is } m \times N \text{ matrix} \]

The complete set of the output vectors can also be calculated as
\[ Y = W X \quad (2) \]

Typically, the actual output vector, \( y(n) \), differs from the desired output vector, \( d(n) \), and the error may be calculated as follows:
\[ e(n) = d(n) - y(n) \text{ is a } m \times 1 \text{ vector} \quad (3) \]

where each component being equal to
\[ e_j(n) = d_j(n) - y_j(n) \quad (4) \]

Surface approximation problem, specified by the hyper-plane and set of weight vectors stored in the weight matrix, \( W \), is to find out the optimal weights so that the errors are minimized.

The total measure of the goodness of approximation, or the performance index, can be specified by the mean-squared error
\[ J(W) = \frac{1}{2M} \sum_{n=1}^{N} \sum_{j=1}^{m} e_j^2(n) = \frac{1}{2M} \sum_{n=1}^{N} e^T(n)e(n) = \frac{1}{2M} e^T e \quad , (5) \]
where $M = m \cdot N$ - the total number of all errors $e_j(n)$ over $m$ neurons;

$N$ - training vectors, $\{x, d\}$;

e - $M \times 1$ vector consisting of all errors.

$J(W)$ is called the performance index. It is a non-negative function of $(m \times p)$ weights and may be represented as a quadratic surface in the weight space.

In order to solve the approximation problem, we need to determine the weight matrix which minimizes the performance index, that is, the mean-squared error, $J(W)$.

Let us assume for simplicity the single-neuron (single output) situation, when $m = 1$. The weight matrix, $W$, becomes the $1 \times p$ vector, $w$ and the mean-squared error, $J(w)$, can now be calculated in the following way:

$$J(w) = \frac{1}{2N} (D - Y) (D - Y)^T = \frac{1}{2N} (D \cdot D^T - D \cdot Y^T - Y \cdot D^T + Y \cdot Y^T)$$

where $D$ and $Y = w X - 1 \times N$ row-matrices.

If we take into account that the inner product of vectors is commutative, that is, $u^T \cdot v = v^T \cdot u$, then we have

$$J = \frac{1}{2N} (\|D\|^2 - 2DY^T + YY^T) = \frac{1}{2N} (\|D\|^2 - 2DX^Tw^T + wXX^Tw^T)$$

Let us denote by
\[ q = (D \cdot X^T)/N - 1 \times p \text{ cross-correlation vector (6)} \]

\[ R = (X \cdot X^T)/N - p \times p \text{ input correlation matrix (7)} \]

Then the mean-squared error finally becomes may be presented as follows

\[ J(w) = \frac{1}{2} \left( \|D\|_2^2 / N - 2qw^T + wRw^T \right) \quad (8) \]

In order to find the optimal weight vector which minimizes the mean-squared error, \( J(w) \), we need to calculate the gradient of \( J \) with respect to \( w \)

\[ \nabla J(w) = \frac{\partial J}{\partial w} = \begin{bmatrix} \frac{\partial J}{\partial w_1} & \cdots & \frac{\partial J}{\partial w_p} \end{bmatrix} = \frac{1}{2} \nabla(\|D\|_2^2 / N - 2qw^T + wRw^T) = -q + wR \]

Taking into account that \( R = R^T \) (a symmetric matrix), the gradient of the performance index finally becomes

\[ \nabla J(w) = -q + wR \quad (9) \]

The gradient, \( \nabla J(w) \), becomes zero for

\[ wR = q \quad (10) \]

Equation (10) is very important and known as the normal or Wiener-Hopfield equation. It represents a set of \( p \) linear equations for \( w = [w_1 \ldots w_p] \). The solution, if exists, can be easily found from the following

\[ w = q \cdot R^{-1} = q / R = DX^T(XX^T)^{-1} \quad (11) \]
Using a concept of the pseudo-inverse of a matrix $X$ defined as

$$X^T \overset{\text{def}}{=} X^T (XX^T)^{-1} \quad (12)$$

the optimal in the least-mean-squared sense weight vector can be calculated as

$$w = D \cdot X^T = D/X \quad (13)$$

For the multi-neuron output case, when $D$ is a $m \times N$ matrix, the optimal weight matrix $W$ (which is $m \times p$) can be calculated in a similar way as

$$W = D \cdot X^+ = D/X \quad (14)$$

In order to check that the weight vector from equation (14) really minimizes the performance index $J(w)$, we need to calculate the second derivative of $J$, which is known as the Hessian matrix, presented here

$$H(w) = \frac{\partial^2 J}{\partial w^2} = \frac{\partial}{\partial w} (\nabla J(w)) = R \quad (15)$$

The second derivative is independent of the weight vector $W$ and is equal to the input correlation matrix, $R$, which, as a product of $X$ and $X^T$, can be proved to be a positive-definite matrix. Moreover, it is the fact that if the number of linearly independent input vectors is at least $p$, then the $R$ matrix is of full rank. It means that
the performance index $J(w)$ attains minimum for the optimal weight vector $W$, and that the minimum is unique.

A matrix $A$ is said to be positive-definite if and only if for all non-zero vectors $x$

$$x^T \cdot A \cdot x > 0$$

LMS (Least Mean Square algorithm) is known as one of the most efficient way to separate the linear separable patterns [9]. The main idea of this method may be described as follows. For a given input vector, the output vector is compared to the correct answer. If the difference is zero, no learning takes place. Otherwise, the weights are adjusted to reduce this difference. In the case of neuron model it may be reduced to instantaneous update of the correlation matrices, that is, on the instantaneous update of the gradient of the mean-squared error [7].

To derive the instantaneous update of the gradient vector it is required first to express the current values of the correlation matrices in terms of their previous values (at the step n-1) and the updates at the step n.
First observe that the current input vector $x(n)$ and the desired output signal $d(n)$ are appended to the matrices $d(n-1)$ and $X(n-1)$ as follows

$$d(n) = [d(n-1) + d(n)]$$
$$X(n) = [X(n-1) + x(n)]$$

Now using definitions of correlation matrices, presented in equation (6) and equation (7) we can write the following

$$\begin{bmatrix} q(n) \\ R(n) \end{bmatrix} = \left( \begin{bmatrix} d(n-1) & d(n) \\ X(n-1) & x(n) \end{bmatrix} \begin{bmatrix} X^T(n-1) \\ x^T(n) \end{bmatrix} \right) / n$$

$$= \left( \begin{bmatrix} d(n-1) & X^T(n-1) + d(n) \\ X(n-1) & X^T(n-1) + x(n) \end{bmatrix} \begin{bmatrix} X^T(n-1) \\ x^T(n) \end{bmatrix} \right) / n$$

$$= \mu \begin{bmatrix} q(n-1) \\ R(n-1) \end{bmatrix} + \begin{bmatrix} \Delta q(n) \\ \Delta R(n) \end{bmatrix} \quad (20)$$

where $\mu = [(n-1)/n] > 1$,

and

$$\Delta q(n) = (d(n) x^T(n))/n \quad \text{and} \quad \Delta R(n) = (x(n) x^T(n))/n \quad (21)$$

are the instantaneous updates of the correlation matrices.

The gradient of the mean-squared error at the step $n$ can also be expanded into its previous values and the current update.
From equation (18), we can get the following statements

\[ \nabla J(n) = - \mu (q(n-1) - w(n) R(n-1)) - \Delta q(n) + w(n) \Delta R(n) \]

or

\[ \nabla J(n) = \nabla \hat{J}(n - 1) + \Delta \nabla J(n) \]

where

\[ \nabla \hat{J}(n - 1) = \mu (- q(n - 1) + w(n) R(n - 1)) \]

is the current estimation of the previous (step n-1) value of the gradient vector, and the gradient vector update is

\[ \Delta \nabla J(n) = - \Delta q(n) + w(n) D R(n) \]

\[ = - \frac{1}{n} (d(n) x^T(n) - w(n) x(n) x^T(n)) \]

\[ = - \frac{1}{n} (d(n) - w(n) x(n)) x^T(n) \]

\[ = - \frac{1}{n} (d(n) - y(n)) x^T(n) = - \frac{1}{n} \varepsilon(n) x^T(n) \]

The Least-Mean-Square learning law replaces the gradient of the mean-squared error in equation (17) with the gradient update and can be written in following form

\[ w(n+1) = w(n) + \eta_n \varepsilon(n) x^T(n) \] (22)

where the output error is
\[ \varepsilon(n) = d(n) - y(n) \]

and the learning gain \( \eta_n \) can be either constant or reducible by the factor \( 1/n \).

Figure 5. Block-diagram

Figure 6. Dendritic Structure of an Adaline b
Implementing the LMS Learning Adaline with its error-correcting learning mechanism. The weight update for a single synapse may be presented the following way (see equation (22) for additional details)

\[
\Delta w_i (n) = \eta \varepsilon(n) x_i (n) \quad (23)
\]

The LMS learning law of equation (22) can be easily expanded into the case of the multi-neuron Adaline.

General comments:

- Computationally, the learning process keeps going through all training examples an unpredictable number of times, until a stopping criterion finally is reached.
• Looking at the behavior of mean-squared error function \( J(W(n)) \) the convergence of the process can be monitored.

• The following are the most popular stopping criteria:
  • the mean-squared error is sufficiently small
    \[ J(W(n)) < \varepsilon \]
  • the change rate of the mean-squared error is sufficiently small
    \[ \frac{\partial J(W(n))}{\partial n} < \varepsilon \]

2.2 LMS Shortcomings

As it was already stated the basic LMS learning rule has pitfalls – it does not work well and efficiently in all cases. In practice during the classification procedure it does not inevitable to update correlation matrices for all elements, that have to be classified.

In general the main pitfalls of the LMS are are the followings:

1. It does not guarantee solution (for example in case of overlapped classes)
2. Outliers (see below).

In order to eliminate those pitfalls let us first define outlier element.

In statistics the outlier elements are those elements in data set that lay above or below three standard deviations limit line. These elements may be deleted from the next stage of the analysis because they don’t affect the process of classification in general. They are located so far from the possible classification line, that it is possible to say that they are classified already. If we leave them in the classification process, the number of iterations will increase significantly. That is the main problem with outliers In order to eliminate those pitfalls a new rule was proposed in this thesis. The rule will be discussed in the next chapter.
Figure 8. Graphical Representation of Outliers
CHAPTER THREE

NEW RULE

3.1 The New Rule

In order to improve the performance of LMS algorithm by decreasing the amount of calculations it was proposed to make an update on each step only for those elements from the input data set, that fall within a small window $W$ near the separating hyperplane surface. The size of this window is a variable parameter of the classification procedure and may be specified during the learning process. Basically it means to exclude the outlier elements from the input pattern from further consideration. In our case determination the size of window parameter $W$ will be based upon classified data (see figure 9).
Figure 9. Graphical Representation of Input Classes

That allows the algorithm to reach stable state faster than in the basic case. Another advantage of this algorithm (as will be shown in the next chapter) is the stability of the final convergence that cannot be reached with basic LMS algorithm in finite number of iterations. In case of overlapping classes (see fig.) the basic LMS algorithm doesn't reach stable state in a finite number of iterations, while experiments showed that LMS algorithm with variable window parameter stops at minimum number of unclassified elements and doesn't start to diverge.
In order to implement the idea of ignoring of outlier elements the basic LMS weight-updating rule from equation (23) should be modified as follows

$$\Delta w_j(n) = \begin{cases} 
\eta \sigma(n)x_j(n) , & \text{if } |x_j(n)| \leq W \\
0, & \text{if } |x_j(n)| > W 
\end{cases}$$

(24)

This new LMS algorithm with additional variable $W$ parameter is supposed to achieve the following advantages:

1. The exclusion of outlier elements helps to improve the performance of basic LMS algorithm by sufficiently decreasing the number of iterations;

2. It makes this algorithm more robust;

3. In case of overlapping classes this algorithm reaches stable state in finite number of iterations.

3.2 Advanced LMS with Window Learning Law Implementation

This work is aimed to describe in detail the results that can be achieved by using the proposed LMS with window learning algorithm in information systems that employs the methodology of neural network for the purposes of classification.
In order to obtain the efficiency of the new algorithm in comparison with the basic LMS algorithm we need to investigate the following:

1. Using the Window parameter of this algorithm show its efficiency in terms of number of iterations, required to classify the same sets of data, in comparison with basic LMS learning algorithm;

2. Demonstrate the stability of convergence behavior of the new algorithm in case of overlapped classes as well as non-overlapped;

3. Find out any specific characteristics of its behavior on randomly generated linearly separated sets of data by varying the size of its Window parameter;

4. Demonstrate the efficiency of using this new algorithm in neural network applications.

Improved LMS learning algorithm

with window

Step 1:

Initialize Weight vector, \( W \), to some small random value.

The size of the window will vary.
Step 2:

While ((# of iterations < Iteration _ Max) and (Changes = YES))

if((WINDOW > 0) and (W(n)X(n) < WINDOW))

then

Net(n) = W(n)X(n)

e(n) = d(n) - net(n)

W(n+1) = W(n) + ηe(n)X(n)
CHAPTER FOUR
METHODOLOGY

4.1 Methodology of the Research
The present research may be divided into several parts. The first part presents the detailed analysis of basic LMS learning algorithm and its use during the learning process in neural network information models of data classification. It is known that basic LMS learning algorithm has several pitfalls. Therefore, this analysis is geared towards finding the best efficient way to improve this algorithm. The proposed LMS learning algorithm (Chapter III) with varying external window parameter is supposed to optimize classification procedure in terms of minimization of the number of iterations and convergence properties.

In the second part of research the comparison of efficiency of basic LMS learning algorithm and proposed LMS learning algorithm with window was performed on real data sets. Comparison of these two algorithms is achieved as follows:
1. First, the software module with both
algorithms, was created using C++ programming
language;

2. The specially developed procedure in this
module randomly generates linearly separable,
overlapped as well as none-overlapped sets of
input data for further testing;

3. The results of separating data by applying both
algorithms are kept in the database. Database
for each case keeps the following parameters:
   a. The input data sets;
   b. Convergence time in terms of number of
      iterations;
   c. The behavior of Mean Square Error and
      Number of unclassified vectors through the
      process;
   d. Number of unclassified vectors after
      process complete;
   e. Convergence/divergence.

Comprehensive analysis and discussion of obtained
results presented in the last part of research.
4.2 The Methodology of the Algorithm Implementation

In order to find out the distinctive features of the proposed method’s behavior a specialized program was created using C++ programming language. This program performs the following tasks:

1. The program each time running automatically generates two sets of linearly separable classes of elements (150 elements each for the testing purpose only) using the internal pseudo random generator of real numbers. These two datasets are stored in an external text file and can be retrieved later during further investigations. In order to skip this procedure the name of already created text file with data sets should be provided as the second command line parameter. As the external parameter this subsystem accepts also the overlapping coefficient that can be added to command line parameters list when program starts (Program_Name [-0 value]). This coefficient helps to control the process of input data sets
generation in terms of overlapping area depth. By default the value of this parameter is 0.

2. The next step is graphical representation of the classification process. Already generated sets of output data are loaded in from the external text file. The program that was used for the graphical representation was Origin 6.0. Graphs for the total error and total number of misclassifications help to analyze the results of the program.

3. The program keeps track of performing calculations by storing the Error Value and number of unclassified vectors for each iteration in external files and prints on the screen values of parameters that helps to monitor the converging behavior of both algorithms in each particular case (different data sets and window size parameters) as presented in the following example:

Example:

C:> ProgName [data file] [-O value] [-W value] [-E value]

Iter # 1   Error vectors: 7   Error: 93.0167
Iter # 2  Error vectors: 1  Error: 4.85061
Iter # 3  Error vectors: 0  Error: 4.85061

Result report:
Deviding vector's coords: (-3.67414, 0.696514, 65.064)
Number of Wrong Vectors: 0
Number of iterations: 3

The -W and -E command line parameters are used to modify default values of window size and minimum error value parameters of algorithm correspondently.

Algorithm keeps working until all vectors are classified or the changes of error value become significantly small (the 0.0000001 value was used during testing sessions).
CHAPTER FIVE

RESULTS

5.1 Testing

In order to prove the efficiency of proposed algorithm the series of tests were performed. Tests were done on the set of real data which was acquired from the machine learning databases of the University of California, Irvine archives, and on sets of randomly generated data. Iris data (UCI dataset) are composed of three classes data of Iris plant namely, Iris Setosa(1), Iris Viginica(2), and Iris Versicolor(3). Classes 1 and 2 and 1 and 3 are nonoverlapped classes. Classes 2 and 3 overlap.

Each test for randomly generated data is done in the following consequent steps:

1. Random generation of two sets of linearly separated data;

2. For each pair of generated classes the basic LMS learning algorithm and proposed in this research new LMS learning algorithm with window were applied. The last one was applied to this data set ten times total with the following
window size parameter variations: 0.1, 0.3, 0.5, 0.7, 0.9, 1, 1.2, 1.5, 1.7, 2. The results were stored in the database that keeps information already mentioned above for each particular case.

3. There were total ten data sets processed during the test study. The first half of cases is the none-overlapped cases and the other half of cases is the overlapped cases with different depth of overlapping. The obtained results are collected in the table.

4. The results for real data are collected in the tables 1-4.

Each test for Iris data is done in the same way, excluding step 1 (data generation).

Implementation of the new LMS rule with window was performed by writing the program in C++ language. The program was executed by using real data set (Iris data) and randomly created data set as an input. The number of iterations and the number of misclassified vectors was recorded as an output. The algorithm runs with different window size (form 0.1 to 1.3) and the results were compared to regular LMS rule.
Tables with the obtained results are shown here.

Table 1. Number of Misclassified Vectors on Iris Data.

Number of Iterations 1000. Class 1 and 2

<table>
<thead>
<tr>
<th></th>
<th>Without</th>
<th>(W=0.1)</th>
<th>(W=0.3)</th>
<th>(W=0.5)</th>
<th>(W=0.7)</th>
<th>(W=0.9)</th>
<th>(W=1)</th>
<th>(W=1.2)</th>
<th>(W=1.5)</th>
<th>(W=1.7)</th>
<th>(W=2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>28</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>11</td>
<td>14</td>
<td>16</td>
<td>13</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>46</td>
<td>7</td>
<td>7</td>
<td>1</td>
<td>8</td>
<td>10</td>
<td>10</td>
<td>17</td>
<td>13</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>48</td>
<td>13</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>13</td>
<td>9</td>
<td>10</td>
<td>15</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>9</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>56</td>
<td>6</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>13</td>
<td>11</td>
<td>11</td>
<td>10</td>
<td>17</td>
</tr>
<tr>
<td>6</td>
<td>31</td>
<td>7</td>
<td>8</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>12</td>
<td>9</td>
<td>1</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>45</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>9</td>
<td>5</td>
<td>10</td>
<td>14</td>
<td>15</td>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>37</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>11</td>
<td>16</td>
<td>9</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>60</td>
<td>17</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>25</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>12</td>
<td>15</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>20</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>1</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>Max</td>
<td>60</td>
<td>17</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>13</td>
<td>13</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>18</td>
</tr>
</tbody>
</table>

Average 39.6  7.1  3.7  3  3.2  4.8  9.7  11.9  11  10.6  11.9

Figure 10. Graphical Representation of the Results. Iris Data. Class 1 and 2
Table 2. Number of Misclassified Vectors on Iris Data.

Class 1 and 3. Number of Iterations 1000

<table>
<thead>
<tr>
<th>#</th>
<th>Without W</th>
<th>W=0.1</th>
<th>W=0.3</th>
<th>W=0.5</th>
<th>W=0.7</th>
<th>W=0.9</th>
<th>W=1.2</th>
<th>W=1.5</th>
<th>W=1.7</th>
<th>W=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>49</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>42</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>38</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>39</td>
<td>1</td>
<td>7</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>56</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>61</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>47</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>8</td>
<td>3</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>38</td>
<td>6</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>40</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>9</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>Min</td>
<td>38</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Max</td>
<td>61</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Average</td>
<td>46</td>
<td>2.5</td>
<td>3.4</td>
<td>2.2</td>
<td>3.4</td>
<td>4.7</td>
<td>3.7</td>
<td>4.6</td>
<td>6.2</td>
<td>5.9</td>
</tr>
</tbody>
</table>

Figure 11. Graphical Representation of the Results. Iris Data. Class 1 and 3
Table 3. Number of Misclassifications for Non Overlapped Randomly Created Classes. Number of Iterations 30000

<table>
<thead>
<tr>
<th>#</th>
<th>Without W</th>
<th>W=0.1</th>
<th>W=0.3</th>
<th>W=0.5</th>
<th>W=0.7</th>
<th>W=0.9</th>
<th>W=1</th>
<th>W=1.2</th>
<th>W=1.5</th>
<th>W=1.7</th>
<th>W=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Min</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Max</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Average</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 12. Graphical Representation of the Results. Iris Data. Class 3 and 2
Table 4. Number of Misclassified Vectors for Overlapped Randomly Created Classes. Number of Iterations 30000

<table>
<thead>
<tr>
<th>#</th>
<th>Without W</th>
<th>W=0.1</th>
<th>W=0.3</th>
<th>W=0.5</th>
<th>W=0.7</th>
<th>W=0.9</th>
<th>W=1.2</th>
<th>W=1.5</th>
<th>W=1.7</th>
<th>W=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Min</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Max</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Average</td>
<td>5.2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Figure 14. Graphical Representation of the Results.

Overlapped Randomly Created Classes
From the analysis of the result it’s clear that:

1. The number of misclassified vectors on the runs with the window for is much smaller then with the runs without the window. On Iris data (class 1 and 2) average number of misclassified vectors without window is 37, while the maximum average number with the window is 10, (when window is 1.7). The minimum average number of misclassified vectors was obtained from window 0.5 (3) vectors). The same results were obtained from other data sets.

2. The number of iterations that were needed to separate separable classes is less when using algorithm with window. It is shown in the tables that for example for the randomly created non overlapped classes the same number of iterations classified classes using algorithm with the window, while basic LMS rule left two vectors unclassified.

3. With overlapped classes the results show that with the same number of iterations algorithm with window left on average 4 vectors unclassified, while basic LMS left 6.
4. In case of real data overlapped classes (Iris 1 and 3) the new algorithm left less number of misclassified vectors then the basic LMS. On average 48 vectors were left unclassified using basic LMS and 2 - using new algorithm. Comparison was done with the same number of iterations.

5. It is clear from the analysis of the results, that with the increase of the window size, the number of misclassifications increases. Average numbers of misclassified vectors increased from 3 to 10.6 for the real data set and from 0 to 1.4 for the randomly created data. The best window size therefore is 0.5.

The results of the classifications are presented in the Appendix B. From analysing the results it can be inferred that Window algorithm has less number of misclassifications then LMS algorithm. In other words Window algorithm is more efficient then LMS algorithm. Detailed analysis of the results is presented in the next chapter.
CHAPTER SIX
CONCLUSION

Analyzing obtained results the following conclusions may be made:

1. In all simulated cases proposed LMS algorithm with window works comparably or better than the basic one. In most performed tests ten - was the maximum number of iterations that was needed to classify the input data sets. On the other hand the basic LMS algorithm needed about hundred times more iterations to find the right decision. Graphical representation of the results for the Data Set 1 (non overlapped classes) show that with the basic LMS rule, the number of misclassified vectors reached 2 and with the proposed rule reached 0 in less number of iterations (window size did not matter). All other non-overlapped data set show the same results (less number of iterations). For the overlapped classes window size did play a significant rule: With window size 0.7-1.0, significantly less number of iterations also
was necessary to reach stable state See Figure).

2. In all non-overlapped cases varying the window size parameter does not cause divergence of the new LMS algorithm. It just changes the value of error of the result classification. On the other hand in case of overlapped input data varying window size parameter using the same algorithm causes misclassification of several vectors from the input data set (see graphical representation of the results).

3. In some cases (especially in partially overlapped cases) the basic LMS algorithm leaves unclassified a smaller number of input vectors then the proposed algorithm. Therefore, this algorithm is more sensitive to this kind of data than the LMS with window algorithm, which takes just more time to find the dividing hyper-plane parameters.

4. The exclusion of outliers causes the faster classification of the data for both overlapped and non-overlapped classes, that can be see from the graphs of the misclassified vectors.
For the same number of misclassifications, the LMS with window algorithm takes significantly less number of iterations.

5. Since we do not update the weights every time, this method may be used in some applications where the weights have to be stable.

6. Those advantages can be explained by the fact that the regular rules do not consider the distance, only the classification, but the new Window rule considers both, since it excludes outliers.
APPENDIX A

PROGRAM SOURCE CODE
Basic LMS:
#include <stdio.h>
#include <iostream.h>
#include <string>
#include <string.h>
#include <time.h>
#include <stdlib.h>
#include <math.h>

#define LRATE 0.0001

struct Vector{
    int number;
    double coords[5];
    char name[80];
};

struct Vector Basa [1000];

void main(void)
{
    FILE *fp, *fp1, *fp2;
    char str [100];
    char *p1, *p2;
    int i, classnumber = 0;
    struct Vector v;
    int flag = 1;
    long ErrorVectorNum;

    strcpy(v.name, "");
    v.number = 0;

    srand(time(0));

    if ((fp = fopen("iris.txt", "r+")) == NULL) return;
    i = 0;
    char s[80] = "ttt";
    while(!feof(fp))
    {
        fgets(str, sizeof(str), fp);
    }
p1 = str;
if ((p2 = strchr(p1, ',')) == NULL) break;
*p2 = '0';
Basa[i].coords[0] = atof(p1);
*p2 = '1';
p1 = p2; p1++;  

if ((p2 = strchr(p1, ',')) == NULL) break;
*p2 = '0';
Basa[i].coords[1] = atof(p1);
*p2 = '1';
p1 = p2; p1++;  

if ((p2 = strchr(p1, '\n')) == NULL) break;
*p2 = '0';

if ((p2 = strchr(p1, ',')) == NULL) break;
*p2 = '0';
Basa[i].coords[2] = atof(p1);
*p2 = '1';
p1 = p2; p1++;  

if ((p2 = strchr(p1, ',')) == NULL) break;
*p2 = '0';
Basa[i].coords[3] = atof(p1);
*p2 = '1';
p1 = p2; p1++;  

strcpy(Basa[i].name, p1);
if (strstr(Basa[i].name, s) == NULL)  
{
    strcpy(s, Basa[i].name);
    classnumber++;
}

Basa[i].number = classnumber;
Basa[i].coords[4] = 1;
i++;

fclose(fp);

if ((fp1 = fopen("out.txt", "w")) == NULL) return;
if ((fp2 = fopen("out1.txt", "w")) == NULL) return;
srand(time(0));

for(i = 0; i < 5; i++)
{
    while ((v.coords[i] = rand() / (double) RAND_MAX * 100) == 0);
    cout << v.coords[i] << "n";
}

int iter = 0;

while(iter < 30000)
{
    float error;
    iter++;
    flag = 0;
    ErrorVectorNum = 0;
    double E = 0;

    for(i = 0; i < 1000; i++)
        if(Basa[i].number == 1) break;

    while(Basa[i].number == 1)
    {
        int d = 1, j;
        double actual = 0;

        for (j = 0; j < 5; j++)
            actual += Basa[i].coords[j] * v.coords[j];

        if(actual < 0) ErrorVectorNum++;
        error = (d - actual)*(d-actual);
        E = (E + error) /* (E + error)*/;

        for (j = 0; j < 5; j++)
            v.coords[j] += LRATE * error * Basa[i].coords[j];

        i++;
    }

    for(i = 0; i < 1000; i++)
        if (Basa[i].number == 2) break;
while(Basa[i].number == 2)
{
    int d = -1, j;
    double actual = 0;

    for (j = 0; j < 5; j++)
        actual += Basa[i].coords[j] * v.coords[j];

    if(actual > 0) ErrorVectorNum++;

    error = (d - actual)*(d-actual);
    E = (E + error)/* (E + error)*/;

    for (j = 0; j < 5; j++)
        v.coords[j] += 0.0001 * error * Basa[i].coords[j];

    i++;
}

fprintf(fp1, "%d, %d\n", iter, ErrorVectorNum);
fprintf(fp2, "%d, %f\n", iter, E);

}

cout << "\nNumber of Wrong Vectors: " << ErrorVectorNum << ")\n";
cout << "\nNumber of iterations: " << iter << ")\n";

fclose(fp1);
fclose(fp2);

Window rule:

#include <stdio.h>
#include <iostream.h>
#include <string>
#include <string.h>
#include <time.h>
#include <stdlib.h>
#include <math.h>
#define LRATE 0.0001

struct Vector{
    int number;
    double coords[5];
    char name[80];
};

struct Vector Basa [1000];

void main(void)
{
    FILE *fp, *fp1, *fp2;
    float WINDOW;
    char str [100];
    char *p1, *p2;
    int i, classnumber = 0;
    struct Vector v;
    int flag = 1;
    long ErrorVectorNum;

    cout << "Enter Window Size: ";
    cin >> WINDOW;
    cout << "\n";
    strcpy(v.name, "")
    v.number = 0;
    srand(time(0));

    if ((fp = fopen("iris.txt", "r+")) == NULL) return;
    i = 0;
    char s[80] = "ttt";
    while(!feof(fp))
    {
        fgets(str, sizeof(str), fp);
        p1 = str;
        if ((p2 = strchr(p1, ',')) == NULL) break;
        *p2 = '\0';
    }
}
Basa[i].coords[0] = atof(p1);
*p2 = ' ';  
p1 = p2; p1++;  

if ((p2 = strchr(p1, ',')) == NULL) break;  
*p2 = '0';  
Basa[i].coords[1] = atof(p1);  
*p2 = ' ';  
p1 = p2; p1++;  

if ((p2 = strchr(p1, '
')) == NULL) break;  
*p2 = '0';  

if ((p2 = strchr(p1, ',')) == NULL) break;  
*p2 = '0';  
Basa[i].coords[2] = atof(p1);  
*p2 = ' ';  
p1 = p2; p1++;  

if ((p2 = strchr(p1, ',')) == NULL) break;  
*p2 = '0';  
Basa[i].coords[3] = atof(p1);  
*p2 = ' ';  
p1 = p2; p1++;  

strcpy(Basa[i].name, p1);  
if (strstr(Basa[i].name, s) == NULL)  
{
    strcpy(s, Basa[i].name);  
    classnumber++;  
}

Basa[i].number = classnumber;  
Basa[i].coords[4] = 1;  
i++;  
}

fclose(fp);

if ((fp1 = fopen("out.txt", "w")) == NULL) return;  
if ((fp2 = fopen("out1.txt", "w")) == NULL) return;  
srand(time(0));
for(i = 0; i < 5; i++)
{
    while ((v.coords[i] = rand() / (double) RAND_MAX * 100) == 0);
cout << v.coords[i] << "\n";
}

int iter = 0;

while(iter < 10000)
{
    float error;
    iter++; flag = 0;
    ErrorVectorNum = 0;
    double E = 0;

    for(i = 0; i < 1000; i++)
        if(Basa[i].number == 1) break;

    while(Basa[i].number == 1)
    {
        int d = 1, j;
        double actual = 0;

        for (j = 0; j < 5; j++)
            actual += Basa[i].coords[j] * v.coords[j];

        if((abs(actual) > WINDOW) && (actual > 0)) {i++; continue;}
        else
            if(actual < 0) ErrorVectorNum++;

            error = (d - actual)*(d-actual);
            E += error;

            for (j = 0; j < 5; j++)
                v.coords[j] += LRATE * error * Basa[i].coords[j];

        i++;
    }

    for(i = 0; i < 1000; i++)
        if (Basa[i].number == 2) break;
while(Basa[i].number == 2)
{
    int d = -1, j;
    double actual = 0;

    for (j = 0; j < 5; j++)
        actual += Basa[i].coords[j] * v.coords[j];

    if((abs(actual) > WINDOW) && (actual < 0)) { i++;
        continue; }
    else 
        if(actual > 0) ErrorVectorNum++;

        error = (d - actual)*(d-actual);
        E += error ;

        for (j = 0; j < 5; j++)
            v.coords[j] += 0.0001 * error * Basa[i].coords[j];

    i++; 
}

fprintf(fp1, "%d, %d\n", iter, ErrorVectorNum);
fprintf(fp2, "%d, %f\n", iter, E);

}

cout << "nNumber of Wrong Vectors: " << ErrorVectorNum << "n";
cout << "nNumber of iterations: " << iter << "n";

cfclose(fp1); fclose(fp2);}
APPENDIX B

GRAPHICAL REPRESENTATION OF

RESULTS
“Number of misclassifications” graphs:
The X-axis represents number of misclassifications.
The Y-axis represents number of iterations.

“Total error” graphs:
The X-axis represents total error.
The Y-axis represents number of iterations.
Randomly created data set 1. Non overlapped classes.
Number of misclassified vectors on the randomly created data set 1.
Non Overlapped classes.
Number of misclassified vectors on the randomly created data set 1. Non overlapped classes. Window=0.1
Number of misclassified vectors on the randomly created data set 1. Non overlapped classes. Window=0.3
Number of misclassified vectors on the randomly created data set 1.
Non overlapped classes. Window=0.7
Number of misclassified vectors on the randomly created data set 1.
Non overlapped classes. Window=1.0
Number of misclassified vectors on the randomly created data set 1.
Non overlapped classes. Window=1.3
Total errors on the randomly created data set 1.
Non overlapped classes.
Total error on the randomly created data set 1.
Non overlapped classes. Window = 0.1
Total error on the randomly created data set 1.
Non overlapped classes. Window = 0.3
Total error on the randomly created data set 1.
Non overlapped classes. Window = 0.7
Total error on the randomly created data set 1. Non overlapped classes. Window = 1.0
Total error on the randomly created data set 1.
Non overlapped classes. Window = 1.3
Randomly created data set 2. Non overlapped classes.
Number of misclassified vectors on the randomly created data set 2.
Non Overlapped classes.
Number of misclassified vectors on the randomly created data set 2.
Non overlapped classes. Window=0.1
Number of misclassified vectors on the randomly created data set 2.
Non overlapped classes. Window=0.3
Number of misclassified vectors on the randomly created data set 2.
Non overlapped classes. Window=0.7
Number of misclassified vectors on the randomly created data set 2.
Non overlapped classes. Window=1.0
Number of misclassified vectors on the randomly created data set 2.
Non overlapped classes. Window=1.3
Total errors on the randomly created data set 2.
Non overlapped classes.
Total error on the randomly created data set 2.
Non overlapped classes. Window = 0.1
Total error on the randomly created data set 2.
Non overlapped classes. Window = 0.3
Total error on the randomly created data set 2.
Non overlapped classes. Window = 0.7
Total error on the randomly created data set 2.
Non overlapped classes. Window = 1.0
Total error on the randomly created data set 2.
Non overlapped classes. Window = 1.3
Randomly created data set 3. Non overlapped classes.
Number of misclassified vectors on the randomly created data set 3.
Non Overlapped classes.
Number of misclassified vectors on the randomly created data set 3.
Non overlapped classes. Window=0.1
Number of misclassified vectors on the randomly created data set 3. Non overlapped classes. Window=0.3
Number of misclassified vectors on the randomly created data set 3.
Non overlapped classes. Window=0.7
Number of misclassified vectors on the randomly created data set 3. Non overlapped classes. Window=1.0
Number of misclassified vectors on the randomly created data set 3. Non overlapped classes. Window=1.3
Total errors on the randomly created data set 3.
Non overlapped classes.
Total error on the randomly created data set 3.
Non overlapped classes. Window = 0.1
Total error on the randomly created data set 3.
Non overlapped classes. Window = 0.3
Total error on the randomly created data set 3.
Non overlapped classes. Window = 0.7
Total error on the randomly created data set 3.
Non overlapped classes. Window = 1.0
Total error on the randomly created data set 3.
Non overlapped classes. Window = 1.3
Randomly created data set 4. Non overlapped classes.
Number of misclassified vectors on the randomly created data set 4.
Non Overlapped classes.
Number of misclassified vectors on the randomly created data set 4.
Non overlapped classes. Window=0.1
Number of misclassified vectors on the randomly created data set 4. Non overlapped classes. Window=0.3
Number of misclassified vectors on the randomly created data set 4.
Non overlapped classes. Window=0.7
Number of misclassified vectors on the randomly created data set 4. Non overlapped classes. Window=1.0
Number of misclassified vectors on the randomly created data set 4.
Non overlapped classes. Window=1.3
Total errors on the randomly created data set 4.
Non overlapped classes.
Total error on the randomly created data set 4.
Non overlapped classes. Window = 0.1
Total error on the randomly created data set 4.
Non overlapped classes. Window = 0.3
Total error on the randomly created data set 4.
Non overlapped classes. Window = 0.7
Total error on the randomly created data set 4.
Non overlapped classes. Window = 1.0
Total error on the randomly created data set 4.
Non overlapped classes. Window = 1.3
Randomly created data set 5. Non overlapped classes.
Number of misclassified vectors on the randomly created data set 5.
Non Overlapped classes.
Number of misclassified vectors on the randomly created data set 5.
Non overlapped classes. Window=0.1
Number of misclassified vectors on the randomly created data set 5.
Non overlapped classes. Window=0.3
Number of misclassified vectors on the randomly created data set 5.
Non overlapped classes. Window=0.7
Number of misclassified vectors on the randomly created data set 5.
Non overlapped classes. Window=1.0
Number of misclassified vectors on the randomly created data set 5.
Non overlapped classes. Window=1.3
Total error on the randomly created data set 5.
Non overlapped classes. Window = 0.1
Total error on the randomly created data set 5.
Non overlapped classes. Window = 0.3
Total error on the randomly created data set 5.
Non overlapped classes. Window = 0.7
Total error on the randomly created data set 5.
Non overlapped classes. Window = 1.0
Total error on the randomly created data set 5.
Non overlapped classes. Window = 1.3
Randomly created data set 1. Overlapped classes.
Number of misclassified vectors on the randomly created data set 1. Overlapped classes.
Number of misclassified vectors on the randomly created data set 1. Overlapped classes. Window=0.1
Number of misclassified vectors on the randomly created data set 1.
Overlapped classes. Window=0.3
Number of misclassified vectors on the randomly created data set 1. Overlapped classes. Window=0.7
Number of misclassified vectors on the randomly created data set 1.
Overlapped classes. Window=1.0
Number of misclassified vectors on the randomly created data set 1.
Overlapped classes. Window=1.3
Total errors on the randomly created data set 1.
Overlapped classes.
Total error on the randomly created data set 1.
Overlapped classes. Window = 0.1
Total error on the randomly created data set 1.
Overlapped classes. Window = 0.3
Total error on the randomly created data set 1.
Overlapped classes. Window = 0.7
Total error on the randomly created data set 1.
Overlapped classes. Window = 1.0
Total error on the randomly created data set 1.
Overlapped classes. Window = 1.3
Randomly created data set 2. Overlapped classes.
Number of misclassified vectors on the randomly created data set 2. Overlapped classes.
Number of misclassified vectors on the randomly created data set 2.
Overlapped classes. Window=0.1
Number of misclassified vectors on the randomly created data set 2. Overlapped classes. Window = 0.3
Number of misclassified vectors on the randomly created data set 2. Overlapped classes. Window=0.7
Number of misclassified vectors on the randomly created data set 2. Overlapped classes. Window=1.0
Number of misclassified vectors on the randomly created data set 2.
Overlapped classes. Window=1.3
Total errors on the randomly created data set 2.
Overlapped classes.
Total error on the randomly created data set 2.
Overlapped classes. Window = 0.1
Total error on the randomly created data set 2.
Overlapped classes. Window = 0.3
Total error on the randomly created data set 2.
Overlapped classes. Window = 0.7
Total error on the randomly created data set 2.
Overlapped classes. Window = 1.0
Total error on the randomly created data set 2. 
Overlapped classes. Window = 1.3
Randomly created data set 3. Overlapped classes.
Number of misclassified vectors on the randomly created data set 3.
Overlapped classes.
Number of misclassified vectors on the randomly created data set 3.
Overlapped classes. Window=0.1
Number of misclassified vectors on the randomly created data set 3.
Overlapped classes. Window=0.3
Number of misclassified vectors on the randomly created data set 3.
Overlapped classes. Window=0.7
Number of misclassified vectors on the randomly created data set 3.
Overlapped classes. Window=1.0
Number of misclassified vectors on the randomly created data set 3.
Overlapped classes. Window = 1.3
Total errors on the randomly created data set 3.
Overlapped classes.
Total error on the randomly created data set 3.
Overlapped classes. Window = 0.1
Total error on the randomly created data set 3.
Overlapped classes. Window = 0.3
Total error on the randomly created data set 3.
Overlapped classes. Window = 0.7
Total error on the randomly created data set 3.
Overlapped classes. Window = 1.0
Total error on the randomly created data set 3.
Overlapped classes. Window = 1.3
Randomly created data set 4. Overlapped classes.
Number of misclassified vectors on the randomly created data set 4.
Overlapped classes.
Number of misclassified vectors on the randomly created data set 4. Overlapped classes. Window=0.1
Number of misclassified vectors on the randomly created data set 4.
Overlapped classes. Window=0.3
Number of misclassified vectors on the randomly created data set 4. Overlapped classes. Window=0.7
Number of misclassified vectors on the randomly created data set 4.
Overlapped classes. Window=1.0
Number of misclassified vectors on the randomly created data set 4. Overlapped classes. Window=1.3
Total error on the randomly created data set 4.
Overlapped classes. Window = 0.1
Total error on the randomly created data set 4.
Overlapped classes. Window = 0.3
Total error on the randomly created data set 4.
Overlapped classes. Window = 0.7
Total error on the randomly created data set 4.
Overlapped classes. Window = 1.0
Total error on the randomly created data set 4.
Overlapped classes. Window = 1.3
Randomly created data set 5. Overlapped classes.
Number of misclassified vectors on the randomly created data set 5.
Overlapped classes.
Number of misclassified vectors on the randomly created data set 5.
Overlapped classes. Window=0.1
Number of misclassified vectors on the randomly created data set 5.
Overlapped classes. Window=0.3
Number of misclassified vectors on the randomly created data set 5. Overlapped classes. Window=0.7
Number of misclassified vectors on the randomly created data set 5. Overlapped classes. Window=1.0
Number of misclassified vectors on the randomly created data set 5.
Overlapped classes. Window=1.3
Total error on the randomly created data set 5.
Overlapped classes. Window = 0.1
Total error on the randomly created data set 5.
Overlapped classes. Window = 0.3
Total error on the randomly created data set 5.
Overlapped classes. Window = 0.7
Total error on the randomly created data set 5.
Overlapped classes. Window = 1.0
Total error on the randomly created data set 5. 
Overlapped classes. Window = 1.3
Number of misclassified vectors on Iris data (class 1 and 2).
Number of misclassified vectors on Iris data (class 1 and 2).
Window=0.1
Number of misclassified vectors on Iris data (class 1 and 2).
Window=0.3
Number of misclassified vectors on Iris data (class 1 and 2).
Window=0.7
Number of misclassified vectors on Iris data (class 1 and 2).
Window=1.0
Number of misclassified vectors on Iris data (class 1 and 2).
Window=1.3
Total error on iris data set (class 1 and 2)
Total error on Iris data set (class 1 and 2). $W=0.1$
Number of misclassified vectors on Iris data (class 1 and 3).
Number of misclassified vectors on Iris data (class 1 and 3). Window=0.1
Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3

- Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3
Number of misclassified vectors on Iris data (class 1 and 3). Window=0.7
Number of misclassified vectors on Iris data (class 1 and 3). Window=1.0
Number of misclassified vectors on Iris data (class 1 and 3). Window=1.3
Total error on Iris data set (class 1 and 3).
Total error on Iris data set (class 1 and 3). W=0.3.
Number of misclassified vectors on Iris data (class 1 and 3).
Number of misclassified vectors on Iris data (class 1 and 3). Window=0.1
Number of misclassified vectors on Iris data (class 1 and 3). Window=0.3
Number of misclassified vectors on Iris data (class 1 and 3). Window=0.7
Number of misclassified vectors on iris data (class 1 and 3). Window=1.0
Number of misclassified vectors on Iris data (class 1 and 3). Window=1.3
REFERENCES

1. Mandziuk Jacek, Jagota Arun, 1997, Experimental Study of Perceptron-type Online Local Learning Rule for Hopfield Associative Memory. IEEE.


