

## **Copyright Warning & Restrictions**

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be “used for any purpose other than private study, scholarship, or research.” If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of “fair use” that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

**Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation**

Printing note: If you do not wish to print this page, then select “Pages from: first page # to: last page #” on the print dialog screen

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

## ABSTRACT

Title of Thesis :

**Adaptive Stack Filtering by  
LMS and Perceptron Learning.**

*Yu-chou Huang,* New Jersey Institute of Technology,  
Master of Electrical Engineering, 1991

Thesis directed by : *Dr. Nirwan Ansari*  
Department of Electrical  
and Computer Engineering  
New Jersey Institute of Technology.

Stack filters are a class of sliding-window nonlinear digital filters that possess the weak superposition property(threshold decomposition) and the ordering property known as the stacking property. They have been demonstrated to be robust in suppressing noise. Two methods are introduced in this thesis to adaptively configure a stack filter. One is by employing the Least Mean Square(LMS) algorithm and the other is based on Perceptron learning.

Experimental results are presented to demonstrate the effectiveness of our methods to noise suppression.

2) **Adaptive Stack Filtering by  
LMS and Perceptron Learning**

by

1) *Yu-chou Huang*

Thesis submitted to the Faculty of the Graduate  
School of the New Jersey Institute of Technology  
in partial fulfillment of the requirements for the  
degree of Master of Science in Electrical  
Engineering, 1991



**APPROVAL SHEET**

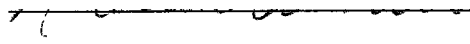
**Title of Thesis :**

**Adaptive Stack Filtering by LMS and  
Perceptron Learning**

**Name of Candidate :** *Yu-chou Huang*

Thesis and Abstract Approved :

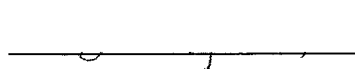
Dr. Nirwan Ansari  
Assistant Professor  
Department of Electrical  
and Computer Engineering

  
12/20/90

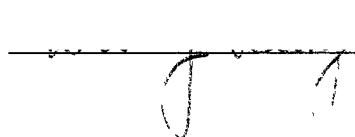
Date

Signature of other members of the thesis committee :

Dr. Edwin S. H. Hou  
Assistant Professor  
Department of Electrical  
and Computer Engineering

  
12/20/90  
Date

Dr. Irving Y. Wang  
Assistant Professor  
Department of Electrical  
and Computer Engineering

  
12/20/90  
Date

## VITA

Name : *Yu-chou Huang*

Permanent Address :

Degree and date to be conferred : M.S.E.E., Jan. 1991

Date of birth :

Place of birth :

Secondary education :

Collegiate institutions attended	Dates	Degree	Date of degree
Chinese Culture University	Sep. 1980	B.S.	June 1984
New Jersey Institute of Technology	Sep. 1988	M.S.	Jan. 1991

Major : **Electrical Engineering**

## ACKNOWLEDGEMENT

I wish to express my sincere appreciation to my advisor, Dr. Nirwan Ansari, for his guidance and encouragement throughout this research effort. Special acknowledgement is extended to Dr. Jean-Hsang Lin, Department of Electrical Engineering, University of Delaware, for his comments and insights on this work. The “Mexican hat” data provided by Dr. Chee-Hung Chu, Center for Advanced Computer Studies, the University of Southwestern Louisiana, are greatly appreciated. I also wish to thank my fellow graduate students in the Department of Electrical and Computer Engineering at New Jersey Institute of Technology for their friendship.

Furthermore, I would like to express my gratitude to other Committee members for serving on the examination committee and for evaluating this research.



# TABLE OF CONTENTS

Acknowledgement	i
Table of Contents	ii
List of Figures	iii
<b>1 Introduction</b>	<b>1</b>
<b>2 Stack Filters</b>	<b>4</b>
2.1 Threshold Decomposition and Stacking Properties . . . . .	6
2.2 Mathematical Descriptions . . . . .	7
2.3 Deterministic Properties of Stack Filters . . . . .	10
<b>3 Linear Discriminant Function, Perceptron and LMS</b>	<b>13</b>
3.1 Concepts . . . . .	13
3.2 The Discriminant Function and Decision Surface . . . . .	14
3.3 LMS and Perceptron Learning . . . . .	16
3.4 Comparison between LMS and Perceptron . . . . .	21
<b>4 Configuring Stack Filters by LMS and Perceptron Learning</b>	<b>23</b>
4.1 Preview . . . . .	23
4.2 General Single-neuron Structure for Configuring Stack Filters . . . . .	24
4.3 Training Procedure . . . . .	27
4.4 Experimental Results . . . . .	29
<b>5 Conclusions</b>	<b>57</b>
<b>Bibliography</b>	<b>59</b>

## List of Figures

Fig. 2.1	Median filter with window width 3. ....	5
Fig. 3.1	Adaline (Adaptive threshold logic element)- Adaptive Filter Structure by LMS and Perceptron rule. ....	15
Fig. 4.1	Additive. ....	24
Fig. 4.2	Single neuron structure during training. ....	26
Fig. 5.1	Original signal (a combination of sinusoids) and zero mean Gaussian noise are shown separately. ....	31
Fig. 5.2	Corrupted signal - a combination of sinusoids + Gaussian noise. ....	32
Fig. 5.3	Output signal obtained by filtering the corrupted signal shown in Fig. 5.2 (a) by LMS rule, window width = 3, (b) by Perceptron rule, window width = 3, respectively. ....	33
Fig. 5.4	Output signal obtained by filtering the corrupted signal shown in Fig. 5.2 (a) by LMS rule, window width = 7, (b) by Perceptron rule, window width = 7, respectively. ....	35
Fig. 5.5	Output signal obtained by filtering the corrupted signal shown in Fig. 5.2 (a) by LMS rule, window width = 11, (b) by Perceptron rule, window width = 11, respectively. ....	37
Fig. 5.6	Original signal (a combination of sinusoids) and $\epsilon$ -mixture Gaussian noise are shown separately. ....	39
Fig. 5.7	Corrupted signal - a combination of sinusoids + $\epsilon$ -mixture of Gaussian noise. ....	40

Fig. 5.8	Output signal obtained by filtering the corrupted signal shown in Fig. 5.7 (a) by LMS rule, window width = 3, (b) by Perceptron rule, window width = 3, respectively. ....	41
Fig. 5.9	Output signal obtained by filtering the corrupted signal shown in Fig. 5.7 (a) by LMS rule, window width = 7, (b) by Perceptron rule, window width = 7, respectively. ....	43
Fig. 5.10	Output signal obtained by filtering the corrupted signal shown in Fig. 5.7 (a) by LMS rule, window width = 11, (b) by Perceptron rule, window width = 11, respectively. ....	45
Fig. 5.11	“Mexican hat” signal and $\epsilon$ -mixture of Gaussian noise are shown separately. ....	47
Fig. 5.12	Corrupted signal - “Mexican hat” signal + $\epsilon$ -mixture of Gaussian noise. ....	48
Fig. 5.13	Output signal obtained by filtering the corrupted signal shown in Fig. 5.12 (a) by LMS rule, window width = 3, (b) by Perceptron rule, window width = 3, respectively. ....	49
Fig. 5.14	Output signal obtained by filtering the corrupted signal shown in Fig. 5.12 (a) by LMS rule, window width = 7, (b) by Perceptron rule, window width = 7, respectively. ....	51
Fig. 5.15	Output signal obtained by filtering the corrupted signal shown in Fig. 5.12 (a) by LMS rule, window width = 11, (b) by Perceptron rule, window width = 11, respectively. ....	53
Fig. 5.16	Mean absolute error between the original “Mexican hat” signal and the signal corrupted by $\epsilon$ -mixture noise. ....	55

Fig. 5.17 Mean squared error between the original “Mexican hat”  
signal and the signal corrupted by Gaussian noise. ....56

# Chapter 1

## Introduction

Artificial neural networks or simply “neural nets [17]” go by many names such as connectionism, parallel distributed processing, and neuromorphic systems. The goal of neural nets is to achieve good performance via dense interconnection of simple computational elements. Instead of performing a program of instructions sequentially as in a von Neumann computer, neural nets explore many competing hypotheses simultaneously using massively parallel nets composed of many computational elements connected by links with variable weights.

Computational elements or nodes used in neural net models are nonlinear. The simplest element sums  $N$  weighted inputs and passes the result

through a nonlinearity. Neural nets are specified by the net topology, node characteristic, and training or learning rules. These rules specify an initial set of weights and indicate how weights should be adapted during use to improve performance. Both design procedures and training rules are the topic of much current research.

The subject of adaptive filters [1][15] has matured to the point where it now constitutes an important part of statistical signal processing. Whenever there is a requirement to process signals that result from operation in an environment of unknown statistics, the use of adaptive filters offers an attractive solution to the problem as it usually provides a significant improvement in performance over the use of a fixed filter designed by conventional methods. Furthermore, it provides new signal processing capabilities that would not be possible otherwise.

All stack filters [2][3][5][10][11][12][19] obey the threshold decomposition property and stacking property. The difference between two stack filters lies solely in the Boolean operation performed on each level. A necessary and sufficient condition for a Boolean operation to preserve the stacking property was proven by Gilbert [13]. He showed the operation must be positive, in which case it has a minimum sum of products representation which is free

of complements of any of the variables. Thus, only the logical AND and OR operations are permitted. A more detailed introduction will be discussed in Chapter 2.

Adaptive filtering approach for stack filters have recently been proposed [4][16]. In this thesis, two new methods based on LMS and Perceptron learning are developed to configure stack filters.

Two adaptive algorithms, LMS [6] and Perceptron, along with their properties and differences will be discussed in Chapter 3. In Chapter 4, the incorporation of the Perceptron and LMS learning to configure stack filters will be developed. Conclusion will be made and future research directions will be recommended in Chapter 5.

# Chapter 2

## Stack Filters

Median filters [2][3][9][10][11] and other rank-order operators [12] possess two properties called threshold decomposition property and the stacking property. The first is a limited superposition property which leads to a new architecture for these filters; the second is an ordering property which allows an efficient VLSI implementation of the threshold decomposition architecture.

Any filter which possesses the threshold decomposition property and stacking property is known as a stack filter [19]. Thus, stack filters form a very large class of easily implemented nonlinear filters which include the rank order operators as well as all compositions of morphological operators.



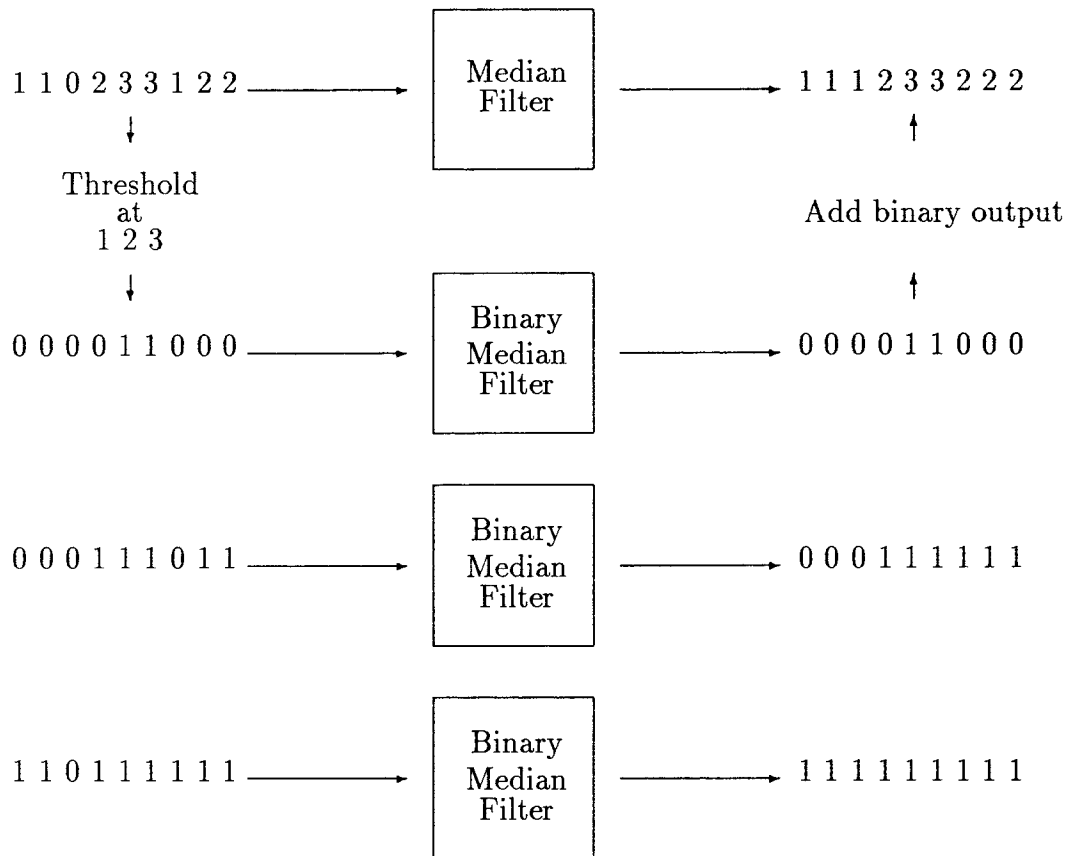


Fig. 2.1 Median filter with window width 3.

Based on the threshold decomposition property and the stacking property, stack filters can be constructed as a “stack” of Positive Boolean functions [13][14], which will be discussed later.

## 2.1 Threshold Decomposition and Stacking Properties

As mentioned above, an entire class of filters which possess the weak superposition property, also called the threshold decomposition, and the stacking property, are called stack filters. These properties can be easily illustrated by a rank order filter such as the median filter with window width three, as shown in Fig. 2.1.

Passing an  $M$ -valued discrete time signal through a rank-order filter is equivalent to the following procedure:

i) Decomposing the  $M$ -valued input signal into a set of  $M-1$  binary signals. The  $k$ th binary signal, where  $k$  is an integer in  $\{1,2,\dots,M-1\}$ , is obtained by thresholding the input signal at value  $k$ . That is, it takes a value of 1 whenever the input signal is greater than or equal to  $k$ , but it is 0 otherwise. Note that summing these  $M-1$  binary signals always provides the original input signal, as illustrated in Fig. 2.1.

ii) Filtering each binary signal independently with its own rank-order filter. Note that each threshold level is performed in parallel. During the process of filtering, each rank-order filter simply adds the number of bits in

the window and compares the result to an integer  $r$ , the desired rank of the filter. The output is 1 when the summation is greater than or equal to  $r$ , and 0 when the summation is less than  $r$ . For example, if the filter's window is  $b=2r+1$ , the filter is a median filter.

iii) Adding the output of each binary rank-order filter one sample at a time, as shown in Fig. 2.1. It is found that the output of the rank-order filter possesses the stacking property. This means that the binary output signals are piled on top of each other according to their threshold levels. It can be seen that a column of 1's always has a column of 0's on top. The desired output value is simply the value of the threshold level where the transition from 1 to 0 takes place.

From Fig. 2.1 and these three steps, we can easily see that "superposition" and "stacking" properties hold. That's why this class of filters is called "stack filters".

## 2.2 Mathematical Descriptions

Consider a vector sequence of length  $n$ ,  $\mathbf{s} = (s_1, s_2, \dots, s_n)$ , where  $s_i$  is the  $i$ th element of this sequence  $\mathbf{s}$ , and  $i = 1, 2, 3, \dots, n$ . Two binary

sequences of length  $n$ ,  $\mathbf{x} = (x_1, x_2, \dots, x_n)$  and  $\mathbf{y} = (y_1, y_2, \dots, y_n)$ , are said to be equal,  $\mathbf{x} = \mathbf{y}$ , if and only if  $x_i = y_i$  for all  $i$ . We say that  $\mathbf{x} \leq \mathbf{y}$  if  $x_i = 1$  implies  $y_i = 1$  for all  $i$ , and in addition, if  $\mathbf{x} \neq \mathbf{y}$ , we say  $\mathbf{x} < \mathbf{y}$ .

Consider an  $M$ -valued sequence  $\mathbf{s}$ . The threshold signals  $\vec{T}^1, \vec{T}^2, \dots, \vec{T}^{M-1}$  of the sequence are defined by

$$\vec{T}^j(i) = \begin{cases} 1 & \text{if } \mathbf{s}(i) \geq j \\ 0 & \text{if } \mathbf{s}(i) < j, \end{cases} \quad (2.1)$$

where  $i$  stands for the  $i$ th element of the appropriate vector and each element of a threshold vector is binary.

Note that these threshold vectors possess the stacking property

$$\vec{T}^1 \geq \vec{T}^2 \geq \dots \geq \vec{T}^{M-2} \geq \vec{T}^{M-1}, \quad (2.2)$$

which implies

$$\vec{T}^1(i) \geq \vec{T}^2(i) \geq \dots \geq \vec{T}^{M-2}(i) \geq \vec{T}^{M-1}(i), \quad (2.3)$$

where  $i$  is the  $i$ th element of the appropriate vector.

Let  $\mathbf{x}$  and  $\mathbf{y}$  be two binary sequences. A filter defined by a function  $F(\cdot)$  is said to have the stacking property if

$$F(\mathbf{x}) \geq F(\mathbf{y}) \quad \text{whenever } \mathbf{x} \geq \mathbf{y}. \quad (2.4)$$

That is, for a filter with window width  $n$ ,

$$F(x_1, \dots, x_n) \geq F(y_1, \dots, y_n) \quad (2.5)$$

when  $x_i \geq y_i, \forall i$ .

Based on Equations (2.2), (2.3), (2.4) and (2.5), the output of this filter should have the following relation

$$F(\vec{T}^1) \geq F(\vec{T}^2) \geq \dots \geq F(\vec{T}^{M-2}) \geq F(\vec{T}^{M-1}). \quad (2.6)$$

The function  $F(\cdot)$  of a rank-order filter, in fact, is a Boolean function. Gilbert [13] called functions satisfying these properties “frontal functions;” elsewhere they are called “Positive Boolean functions.” It means that if filters with function  $F(\cdot)$  have the stacking property, then  $F(\cdot)$  must be a positive Boolean function. The operation of these positive Boolean functions is simply the “max” and “min” operations. For example, a function  $F(\cdot)$  is a positive Boolean function of a stack filter  $S_F$ , and  $F(x_1, x_2, x_3) = x_1 + x_3$ . Thus the operation of the positive Boolean function is  $\max(x_1, x_3)$ .

## 2.3 Deterministic Properties of Stack Filters

Since median filters and rank-order filters are nonlinear, we cannot consider the filtering of an input signal and the filtering of noise separately. Instead, we have to approach the problem syntactically, and see which filter preserves the noisy signal and which eliminates it.

For a median filter, this syntactical approach leads to the idea of the root signal [10], which is any signal that is invariant to filtering by that filter. Essentially, the roots of a median filter are the “passband” signals of the filter. Furthermore, just as a linear filter can estimate a signal within its passband that is corrupted by noise outside the passband, so a linear filter can estimate any root buried in noise. In fact, any signal of finite length will converge to a root in a finite number of passes of a median filter. The output root signal will not be the same as the original uncorrupted signal, but it should be very close.

Stack filters are a generalization of median filters. The structure of a median filter root signal is well known, and the structure and analysis of stack filters are similar. Properties and analysis of stack filters have been covered

in great details [19]. We shall only describe some properties of stack filters without proofs. The following notations which are related to the forthcoming properties of stack filters are adapted in this thesis.

i) Let  $\mathbf{s} = (s_1, s_2, \dots, s_L)$  be an  $M$ -valued signal of length  $L$ . Then for a filter of window width  $2N+1$ , the corresponding appended signal is  $\mathbf{s} = (s_1, \dots, s_1, s_2, \dots, s_{L-1}, s_L, \dots, s_L)$  where  $s_1$  and  $s_{L-1}$  are repeated  $N+1$  times.

ii) The roots of a stack filter are all the appended signals that are invariant under filtering by  $F(\cdot)$ .

iii) We use  $0^m$  and  $1^m$  to denote 0 and 1 repeated  $m$  times.

Properties of stack filters:

*Theorem 1:* Suppose that  $S$  is an  $M$ -level stack filter, based on a nontrivial positive Boolean function  $F(\cdot)$ . Then  $S$  will preserve all constant signals. If  $F(\cdot)$  is identically 0, then  $S$  will preserve only constant zero-valued signals, whereas if  $F(\cdot)$  is identically 1,  $S$  will only preserve signals of value  $M-1$ .

*Theorem 2:* Suppose that an  $M$ -level stack filter  $S$ , based on positive Boolean function  $F$ , has window width  $2N+1$ , then  $S$  preserves all increasing signals if and only if  $F(0^N 1^{N+1}) = 1$  and  $F(0^{N+1} 1^N) = 0$ .

*Theorem 3:* Let  $S$  be  $M$ -level stack filter of window width  $2N+1$ . Then  $S$  preserves the roots of a median filter of window width  $4N+1$  if and only if

S preserves all M-level monotonic signals.



## **Chapter 3**

# **Linear Discriminant Function, Perceptron and LMS**

### **3.1 Concepts**

In this Chapter, we shall be concerned with linear discriminant functions, Perceptron and LMS learning. Linear discriminant functions have a variety of pleasant properties from an analytical point of view and are easy to compute. The problem of finding a linear discriminant function will be formulated as a problem of minimizing a criterion function.

## 3.2 The Discriminant Function and Decision Surface

A linear discriminant function [8] defined by

$$g(\mathbf{u}) = \mathbf{w}^t \mathbf{u} + w_0, \quad (3.1)$$

where  $\mathbf{u}$  is an input pattern vector,  $\mathbf{w}$  is the weight vector, and  $w_0$  is the threshold weight, is used to classify input pattern  $\mathbf{u}$  as one of two possible categories. This two category classification employs the following decision rule: Decide  $c_1$  if  $g(\mathbf{u}) > 0$ , and  $c_2$  if  $g(\mathbf{u}) < 0$ . Thus  $\mathbf{u}$  is assigned to  $c_1$  if the inner product  $\mathbf{w}^t \mathbf{u}$  exceeds the threshold- $w_0$ , and  $\mathbf{u}$  is assigned to  $c_2$  if the inner product  $\mathbf{w}^t \mathbf{u}$  is less than the threshold- $w_0$ . When  $\mathbf{w}^t \mathbf{u}$  is equal to  $w_0$ ,  $\mathbf{u}$  can be assigned to either  $c_1$  or  $c_2$ .

If  $g(\mathbf{u}) = 0$ , it defines the decision surface that separates points assigned to  $c_1$  from points assigned to  $c_2$ . When  $g(\mathbf{u})$  is linear as shown in Equation (3.1), this decision surface is a *hyperplane*. If  $\mathbf{u}_1$  and  $\mathbf{u}_2$  are both on the decision surface, then

$$\begin{aligned} \mathbf{w}^t \mathbf{u}_1 + w_0 &= \mathbf{w}^t \mathbf{u}_2 + w_0 \\ \mathbf{w}^t (\mathbf{u}_1 - \mathbf{u}_2) &= 0. \end{aligned} \quad (3.2)$$

Therefore,  $\mathbf{w}$  is normal to the hyperplane. In general, the hyperplane, we denote it  $\mathbf{H}$ , divides the features space into two halfspaces, the decision region  $R_1$  for  $c_1$ , and  $R_2$  for  $c_2$ . The orientation of the decision surface is decided by  $\mathbf{w}$  and the location of surface is determined by the  $w_0$ .

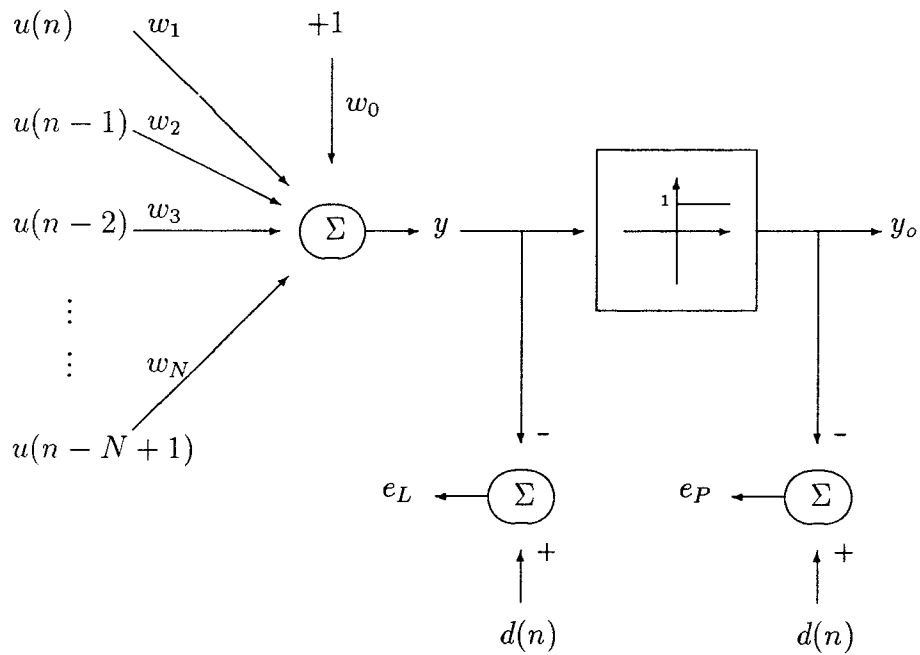


Fig. 3.1 Adaline (Adaptive threshold logic element)–  
Adaptive Filter Structure by LMS and Perceptron rule.

### 3.3 LMS and Perceptron Learning

Before talking about this algorithm, we have to consider the problem of constructing the criterion function. In fact, the Perceptron algorithm is an adaptive algorithm and the structure of the adaptive filter is shown in Fig. 3.1. This structure has two parts: (1) A transversal filter with adjustable tap weights whose values at time  $n$  are denoted  $w_1(n), w_2(n), \dots, w_N(n)$ , and (2) a mechanism for adjusting these tap weights in an adaptive manner.

During the filtering process, an additional signal  $d(n)$ , called the *desired response*, is supplied along with the usual tap input. In fact, the desired signal response provides a frame of reference for adjusting the tap weights of the filter. Denote  $e_L(n)$  and  $e_P(n)$  as the estimation error produced during LMS and Perceptron learning, respectively. Thus as shown in Fig. 3.1,

$$e_L(n) = d(n) - \mathbf{w}^t(n)\mathbf{u}(n), \quad (3.3)$$

where the term  $\mathbf{w}^t(n)\mathbf{u}(n)$  is the inner product of the tap weight vector  $\mathbf{w}(n)$  and the tap input vector  $\mathbf{u}(n)$ , and the superscript  $t$  stands for vector or matrix transpose. Because there are  $N$ -input, that means this filter with window width  $N$ , so the weight vector is  $\mathbf{w}(n) = [w_0(n), w_1(n), w_2(n), \dots, w_N(n)]$ , and input vector is  $\mathbf{u}^t(n) = [1, u(n), u(n-1), \dots, u(n-N+1)]$ .

If the tap input vector  $\mathbf{u}(n)$  and the desired response  $d(n)$  are jointly stationary, then the mean squared error  $J(n)$ , criterion function, at time  $n$  is a quadratic function of the tap weight vector. We may write

$$J(n) = \sigma_d^2 - \mathbf{w}^t(n)\mathbf{p} - \mathbf{p}^t\mathbf{w}(n) + \mathbf{w}^t(n)\mathbf{R}\mathbf{w}(n), \quad (3.4)$$

where  $\sigma_d^2$  is the variance of the desired response  $d(n)$ ,  $\mathbf{p}$  is the cross-correlation vector between the tap-input vector  $\mathbf{u}(n)$  and the desired response  $d(n)$ ,  $\mathbf{R}$  is correlation matrix of the tap-input vector  $\mathbf{u}(n)$ .

It can be found in [10], in contrast to [10], here we only consider real input data, real weights and real desired output data, this criterion function is based on the mean squared error,

$$J(n) = E[(d(n) - \mathbf{w}^t(n)\mathbf{u}(n))(d(n) - \mathbf{u}^t(n)\mathbf{w}(n))]. \quad (3.5)$$

Expanding Equation (3.5) we can get Equation (3.4).

The gradient of the criterion function denoted by  $\nabla$  is simply the derivative of the mean-squared error  $J$  with respect to the tap-weight vector  $\mathbf{w}$ :

$$\nabla = \frac{dJ(n)}{d\mathbf{w}} = -2\mathbf{p} + 2\mathbf{R}\mathbf{w}(n). \quad (3.6)$$

If we let  $\nabla = 0$ , it means that we can get an optimal weight vector such that  $J(n)$  is minimized.

From the above descriptions that  $\mathbf{p}$  is the cross-correlation vector between the tap-input vector  $\mathbf{u}(n)$  and the desired response  $d(n)$ , and  $\mathbf{R}$  is the correlation matrix of the tap-input vector  $\mathbf{u}(n)$ , we can write them as below:

$$\begin{aligned}\mathbf{p} &= E[\mathbf{u}(n)d(n)], \\ \mathbf{R} &= E[\mathbf{u}(n)\mathbf{u}^t(n)].\end{aligned}\tag{3.7}$$

The simplest choice of estimators  $\mathbf{R}$  and  $\mathbf{p}$  is to use the instantaneous estimates that are based on sample values of the tap-input and desired response, as defined by

$$\begin{aligned}\mathbf{R} &= \mathbf{u}(n)\mathbf{u}^t(n), \\ \mathbf{p} &= \mathbf{u}(n)d(n),\end{aligned}\tag{3.8}$$

respectively.

Correspondingly, the instantaneous estimate of the gradient vector is

$$\nabla(n) = -2\mathbf{u}(n)d(n) + 2\mathbf{u}(n)\mathbf{u}^t(n)\mathbf{w}(n).\tag{3.9}$$

According to the method of steepest descent [10], the updated value of the tap-weight vector at time  $n + 1$  is computed by using the simple recursive relation

$$\mathbf{w}(n + 1) = \mathbf{w}(n) + \frac{1}{2}\mu[-\nabla(n)],\tag{3.10}$$

where  $\mu$  is a positive real-valued constant.

Substituting Equation (3.6) into Equation (3.10), we have

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu[\mathbf{p} - \mathbf{R}\mathbf{w}(n)], \quad (3.11)$$

and substituting Equation (3.8) into Equation (3.11), we have the following LMS learning rule:

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) + \mu\mathbf{u}(n)[d(n) - \mathbf{u}^t(n)\mathbf{w}(n)] \\ &= \mathbf{w}(n) + \mu\mathbf{u}(n)e_L(n), \end{aligned} \quad (3.12)$$

where

$$\begin{aligned} e_L(n) &= d(n) - y(n), \\ y(n) &= \mathbf{u}^t(n)\mathbf{w}(n). \end{aligned} \quad (3.13)$$

In Fig. 3.1, the error,  $e_P(n)$ , is generated after passing the linear output,  $y$ , through the hardlimiting function,  $f_H(\cdot)$ . Thus, the output  $y_o$  is

$$y_o(n) = f_H(\mathbf{u}^t(n)\mathbf{w}(n)). \quad (3.14)$$

Replacing  $y$  and  $e_L$  by  $y_o$  and  $e_P$  in Equation (3.12), we having the following Perceptron Learning rule:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu\mathbf{u}(n)e_P(n), \quad (3.15)$$

where

$$e_P(n) = d(n) - y_o(n).$$

Based on the concept of linear discriminant function, the hardlimiting threshold level should be chosen as follow

$$y_o = \begin{cases} 1 & \text{if } y > 0 \\ 0 & \text{if } y \leq 0. \end{cases} \quad (3.16)$$

The single layer perceptron can be used with both continuous valued and binary inputs. This simple net generates much interest when initially developed because of its ability to learn to recognize simple patterns. A perceptron that decides whether an input belongs to one of two classes (we denote  $c_1$  or  $c_2$ ). In Fig. 3.1, the single node computes a weighted sum of input elements and adds a threshold  $w_0$ , then passes the result through a hardlimiting nonlinearity such that the output  $y_o$  is either 0 or 1. The decision rule is to respond Class  $c_1$  if the output is 1 and Class  $c_2$  if the output is 0. A useful technique for analyzing the behavior of nets, such as the Perceptron, is to plot a map of the decision regions created in the multi-dimensional space spanned by the input variables. These decision regions specify which input values result in Class  $c_1$  and which result in Class  $c_2$ . When there are only two inputs, the decision boundary, hyperplane, is a line in the 2-dimentional space, and the boundary line depends on the connection weights and the threshold.



Rosenblatt [18] proved that if the inputs presented from the two classes are separable, then the perceptron convergence procedure converges and positions the decision hyperplane between those two classes. One problem with the perceptron convergence procedure is that decision boundaries may oscillate continuously when inputs are not separable and distributions overlap.

For a linear separable case, LMS or Perceptron learning is good enough to classify the input samples. To classify non-linearly separable samples, it is necessary to use multi-layer networks such as multi-layer Perceptrons [17][20].

### **3.4 Comparison between LMS and Perceptron**

Generally speaking, the Perceptron algorithm and LMS algorithm are almost the same. Both perform weight adaptation based on the estimation error by gradient descent method. However, the estimation error is different from one to the other. This difference, thus, leads to different behavior.

The Perceptron learning rule [18] has been proven to be capable of sepa-

rating linearly separable samples. LMS algorithm is not guaranteed to separate linearly separable samples. If the samples are not linearly separable, the weight vector adapted by the Perceptron rule may oscillate forever and does not converge to a low-error solution. On the other hand, the weight vector obtained by the LMS rule [18] cannot be unreasonable if the samples are not separable. Both learning rules can be generalized to a more general rule by replacing the hardlimiting function by a sigmoid function [6].

# Chapter 4

## Configuring Stack Filters by LMS and Perceptron Learning

### 4.1 Preview

Stack filters possess two properties—threshold decomposition property and stacking property. With these two properties, the implementation of stack filters can be easily realized by a digital circuit. With the technology of VLSI, the stack filters can be designed in a single chip.

In this chapter, we introduce two training algorithms, the LMS algorithm and the Perceptron learning rule, to configure a stack filter.

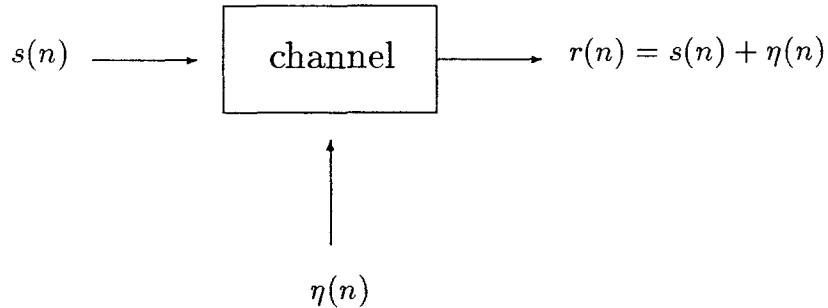


Fig. 4.1 Additive.

## 4.2 General Single-neuron Structure for Configuring Stack Filters

Denote  $s(n)$  as the original signal sequence,  $\eta(n)$  as the noise process, and  $r(n)$  as the resulting sequence. We assume that  $s(n)$  is corrupted by additive noise process, and thus the resulting sequence  $r(n) = s(n) + \eta(n)$ , as shown in Fig. 4.1.

The problem we address in this thesis is to configure a stack filter  $S$  in order to recover the original signal sequence from the corrupted sequence. Since stack filters possess the threshold decomposition and stacking properties, configuring a stack filter is equivalent to first converting the input

signal sequence into sequence of binary signals by threshold decomposition, and then finding the appropriate positive Boolean function used for all level. Now, the input signal sequence is  $r(n)$ . Assume  $r(n)$  is an M-valued sequence, by threshold decomposition we obtain the thresholded binary sequence denoted by  $\vec{T}^{M-1}, \vec{T}^{M-2}, \dots, \vec{T}^2, \vec{T}^1$ , where

$$\vec{T}^1 \geq \vec{T}^2 \geq \dots \geq \vec{T}^{M-2} \geq \vec{T}^{M-1}, \quad (4.1)$$

and

$$\vec{T}^j(n) = \begin{cases} 1 & \text{if } r(n) \geq j, \\ 0 & \text{if } r(n) < j. \end{cases} \quad (4.2)$$

Let N be the window width of the stack filter. At each threshold level, the input sequence is a binary sequence, and the output is a binary number. Thus, the input-output relationship can be realized by a Boolean function. Recall from Chapter 3, some binary Boolean functions can be realized by a linear discriminant function, and thus can be trained like the LMS or the Perceptron discussed in Chapter 3. However, the Boolean function obtained by training the single neuron may not be a positive Boolean function. Heuristics which will be discussed later are introduced to ensure that the resulting Boolean function is a positive Boolean function.

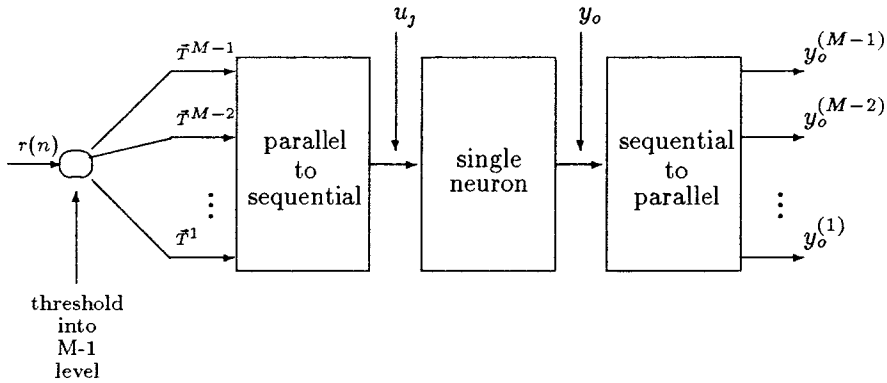


Fig. 4.2 Single neuron structure during training.

The general single-neuron structure for configuring stack filters is shown in Fig. 4.2. The input sequence  $r(n)$  is first converted to threshold binary sequence  $\vec{T}^1, \vec{T}^2, \dots, \vec{T}^{M-1}$ . For each window sample of width  $N$  of the input sequence  $r(n)$ , there are  $(M-1)$  window samples of width  $N$  of the thresholded binary sequences; that is,  $(M-1)$  binary input patterns are presented to the single-neuron. Thus, the weights of the neuron are updated by the  $(M-1)$  binary input patterns  $(M-1)$  times for each sample of  $r(n)$ . The serial of the binary outputs of the neuron are then stacked back into  $(M-1)$  levels. Finally, the  $M$ -valued filtered output signal is reconstructed, by the stacking property, from the binary outputs by a search for level at which a transition from 1 to 0 occurs.

### 4.3 Training Procedure

Denote  $\vec{r}_i$  as the  $i$ th window sample of width  $N$  of the input sequence  $r(n)$ ;  $\vec{T}_i^1, \vec{T}_i^2, \dots, \vec{T}_i^{M-1}$  as the  $(M-1)$  thresholded binary input patterns that result from the  $i$ th input sample  $\vec{r}_i$ . These parallel  $(M-1)$  threshold binary input patterns are transformed into a sequence binary patterns as follow:

$$\vec{u}_j = \vec{T}_i^k \text{ where } j = (M - 1)(i - 1) + k. \quad (4.3)$$

The weights of the neuron are then updated by a learning rule. For the LMS learning, using Equation (3.12) and (3.13), we have

$$\mathbf{w}_{j+1} = \max\{\mathbf{w}_j + \mu \vec{u}_j [d_j - \mathbf{w}^t \vec{u}_j], 0\}, \quad (4.4)$$

where

$$j = (M - 1)(i - 1) + k,$$

$y_j = \mathbf{w}^t \vec{u}_j$  is the  $j$ th binary output of the neuron; i.e., the  $i$ th binary output value corresponding to the  $k$ th level,  $d_j$  is the  $j$ th desired binary output; i.e., the  $i$ th desired binary value corresponding to the  $k$ th threshold level.

Similarly, for the Perceptron learning, we have

$$\mathbf{w}_{j+1} = \max\{\mathbf{w}_j + \mu \vec{u}_j [d_j - f_H(\mathbf{w}^t \vec{u}_j)], 0\}, \quad (4.5)$$

where  $f_H$  is the hardlimiting function.

Note that during training, negative weights except  $w_0$  are set to zero. This heuristics are introduced in order to preserve the stacking property of a stack filter. After training, the final weight vector is used for the remaining inputs. It is easy to show the above heuristics preserve the stacking property.

Denote  $\mathbf{w}^k$  as the weight vector used for the  $k$ th thresholded level signal.

The  $k$ th level output:

$$y_o^k = f_H(y^k),$$

where  $y^k = (\vec{T}^k)^t \mathbf{w}^k$ , and  $k = M - 1, M - 2, \dots, 1$ .

Since

$$\vec{T}^{M-1} \leq \vec{T}^{M-2} \leq \dots \leq \vec{T}^1,$$

and

$$\mathbf{w}^{(M-1)} = \mathbf{w}^{(M-2)} = \dots = \mathbf{w}^{(1)} \geq \mathbf{0},$$

Hence,

$$y_o^{(M-1)} \leq y_o^{(M-2)} \leq \dots \leq y_o^{(1)}.$$



## 4.4 Experimental Results

Two methods to configure stack filters were developed and discussed in details in the last section. We shall demonstrate the effectiveness of our proposed algorithms for noise suppression by experimental results. We have experimented with various types of signals and noise, but we shall only present results for two types of signals and two types of noise. Fig. 5.1 shows the original signal obtained by a linear combination of five sinusoids and the zero mean Gaussian noise. The corrupted signal obtained by adding the signal and noise is shown in Fig. 5.2. The filtered output signals obtained by the LMS and Perceptron rule with various window widths are shown in Fig. 5.3–5.5. Similarly, Fig. 5.6–5.15 show other type of signal, noise and filtered outputs. Fig. 5.16 and Fig 5.17 show the mean absolute error and mean squared error between the desired output and the filtered output by LMS and Perceptron rule using various window widths.

From these results, we can draw the following conclusions:

(1) Perceptron rule outperforms LMS rule in the stack filter.

(2) The noise suppression depends on the signal, noise and filter window width.

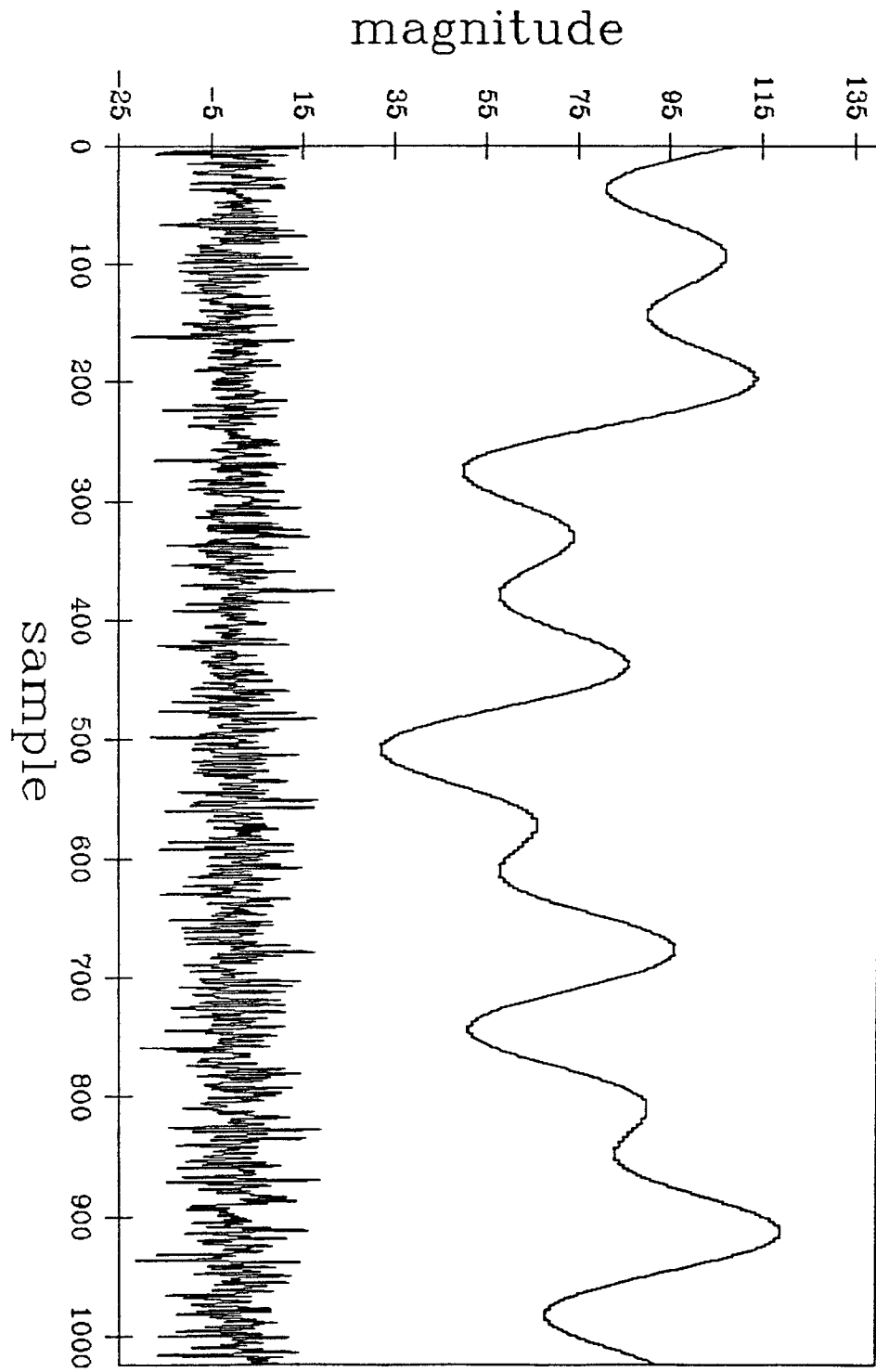


Fig. 5.1 Original signal (a combination of sinusoids) and zero mean Gaussian Noise are shown separately.

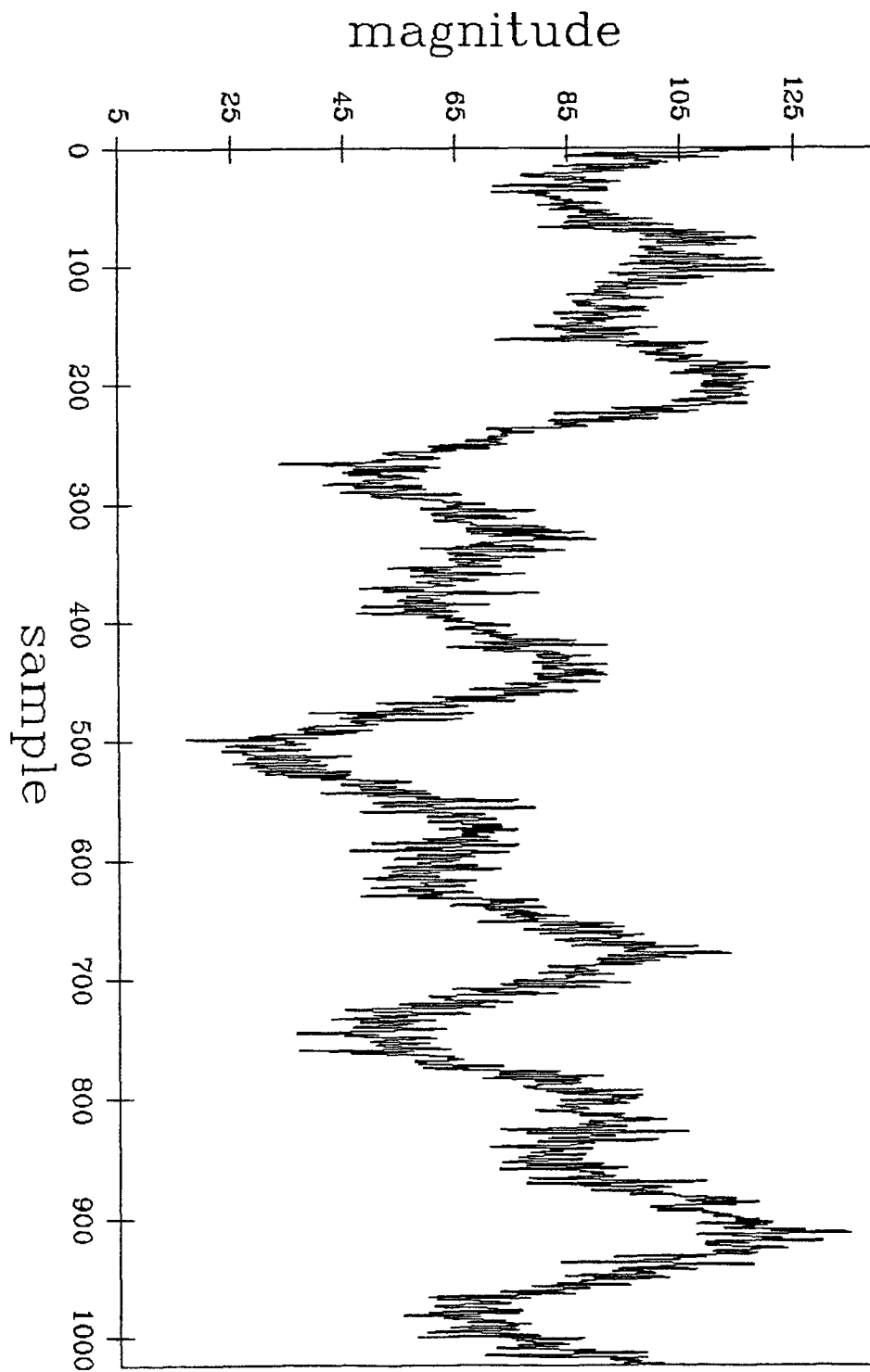


Fig. 5.2 Corrupted signal – a combination of sinusoids + Gaussian noise.

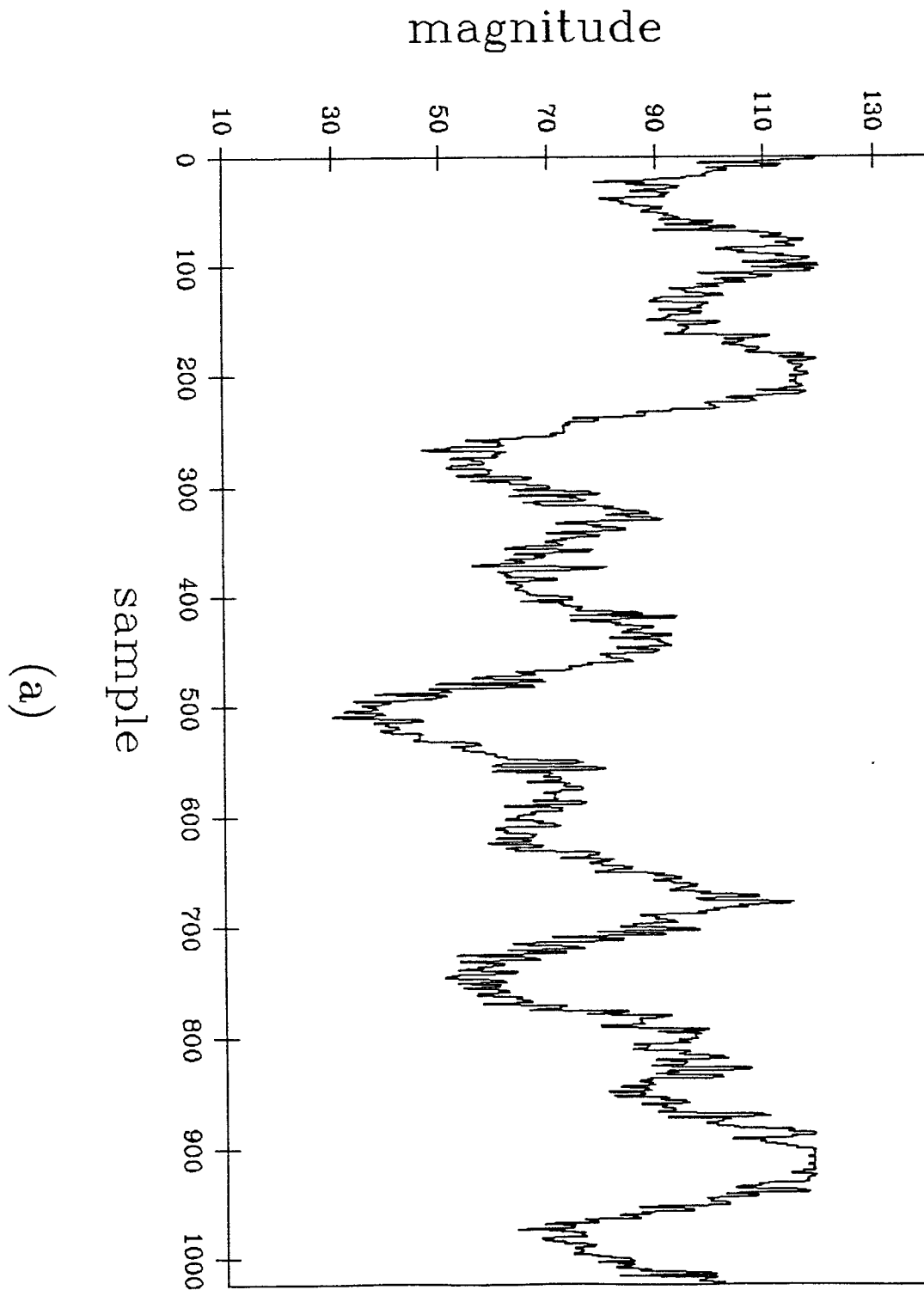


Fig. 5.3 Output signal obtained by filtering the corrupted signal shown in Fig. 2 by (a) LMS rule, window width = 3, and (b) Perceptron rule, window width = 3, respectively.

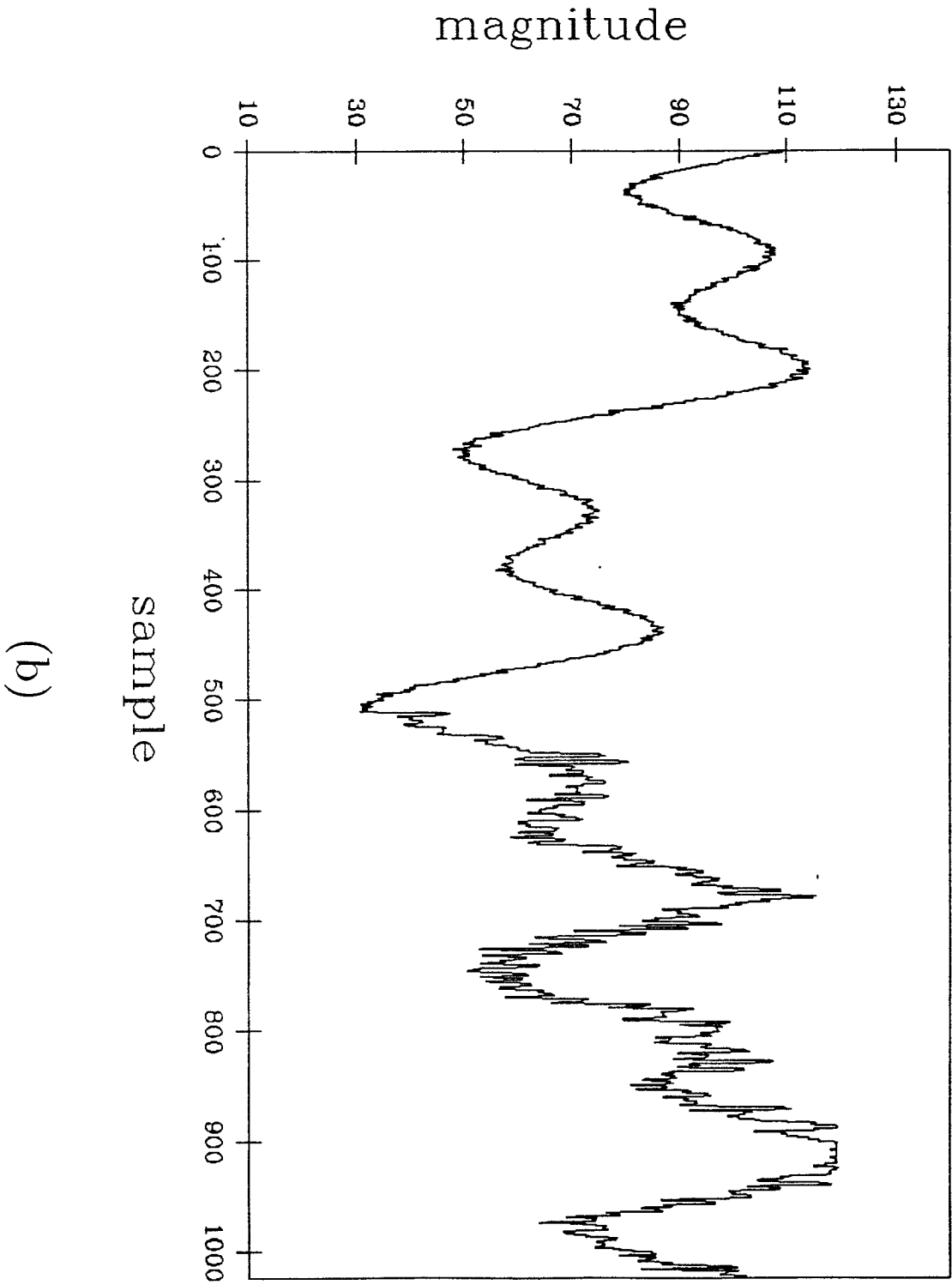


Fig. 5.3, continued.

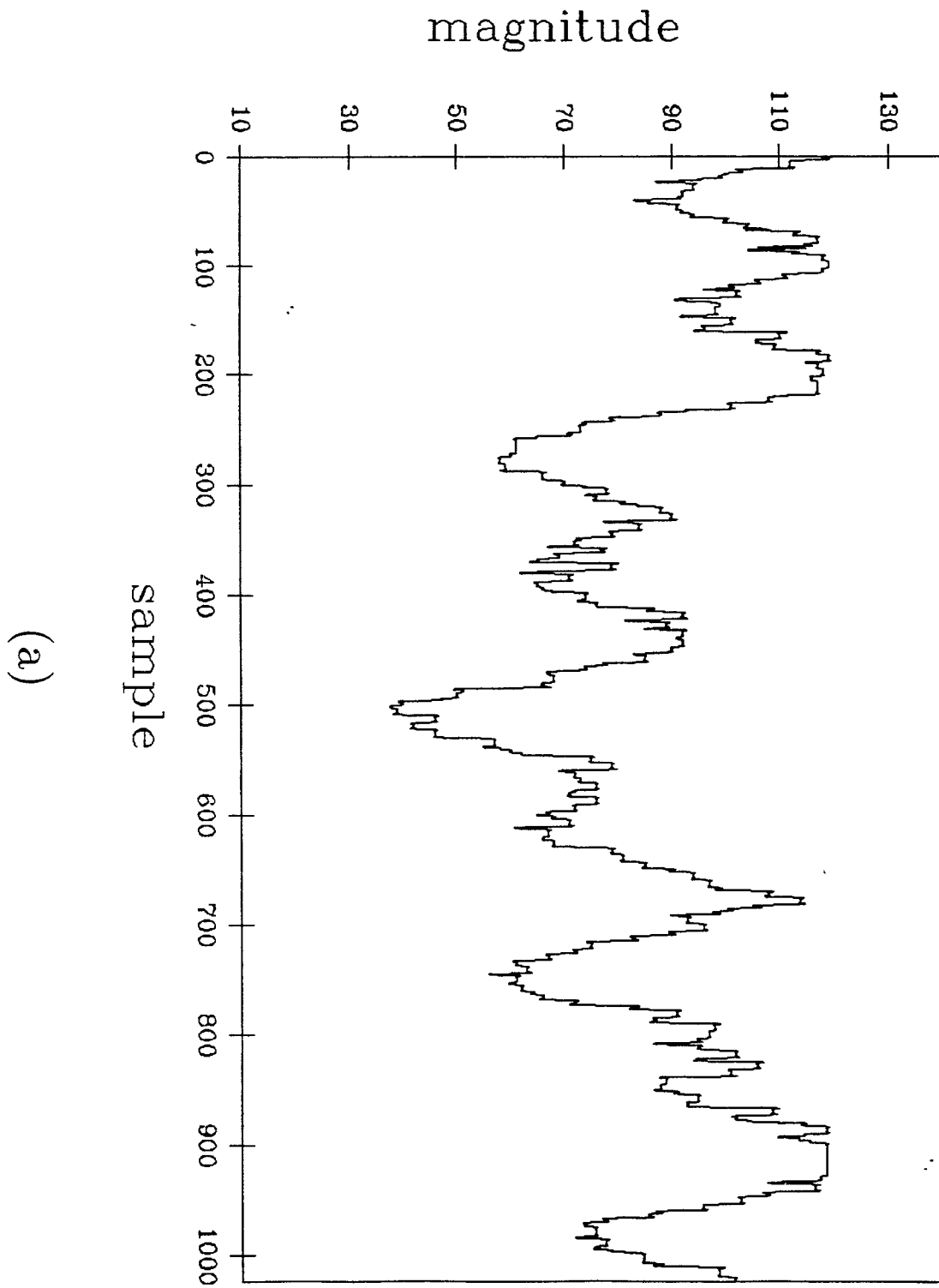


Fig. 5.4 Output signal obtained by filtering the corrupted signal shown in Fig. 2 by (a) LMS rule, window width = 7, and (b) Perceptron rule, window width = 7, respectively.

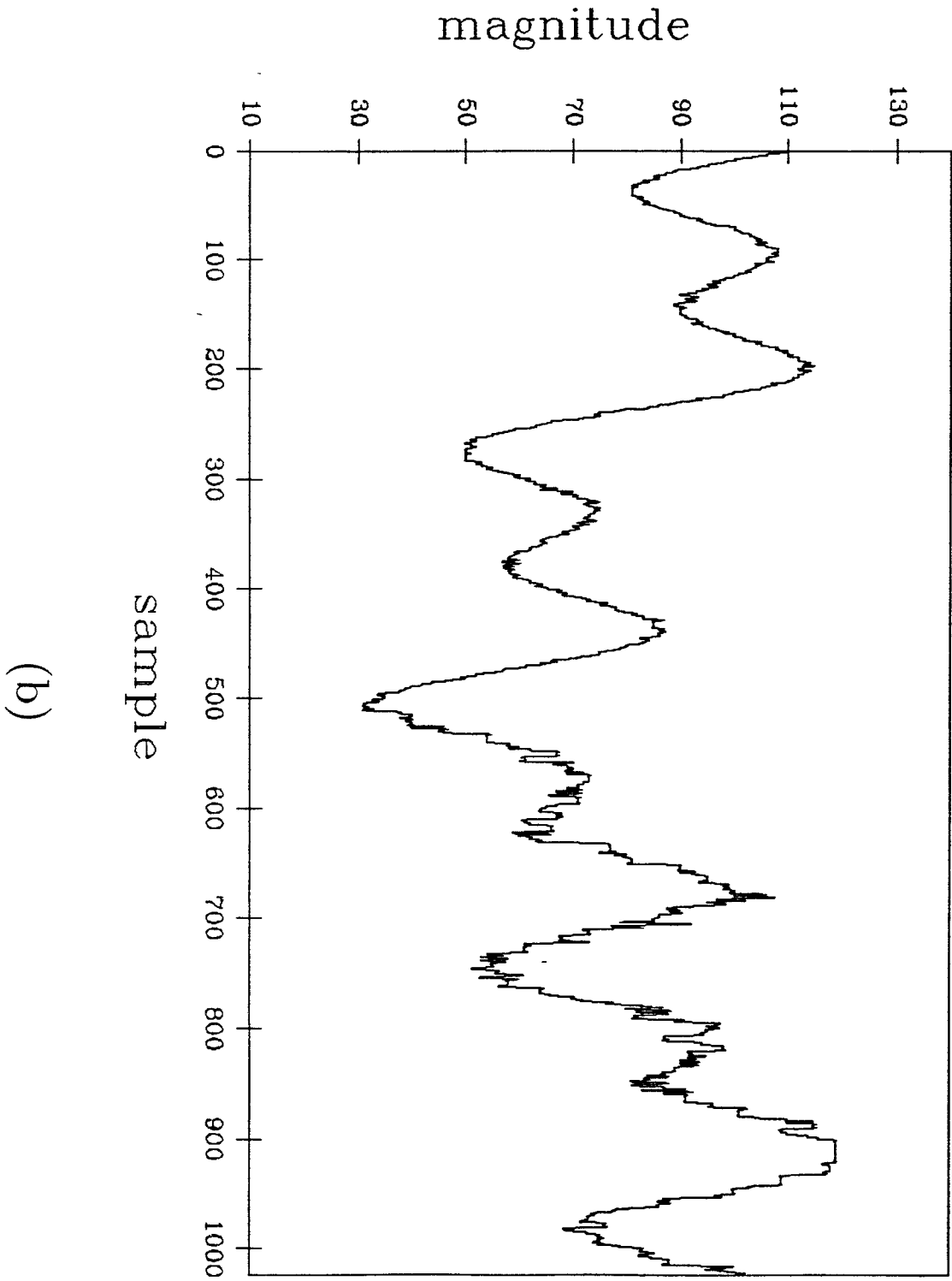


Fig. 5.4, continued.



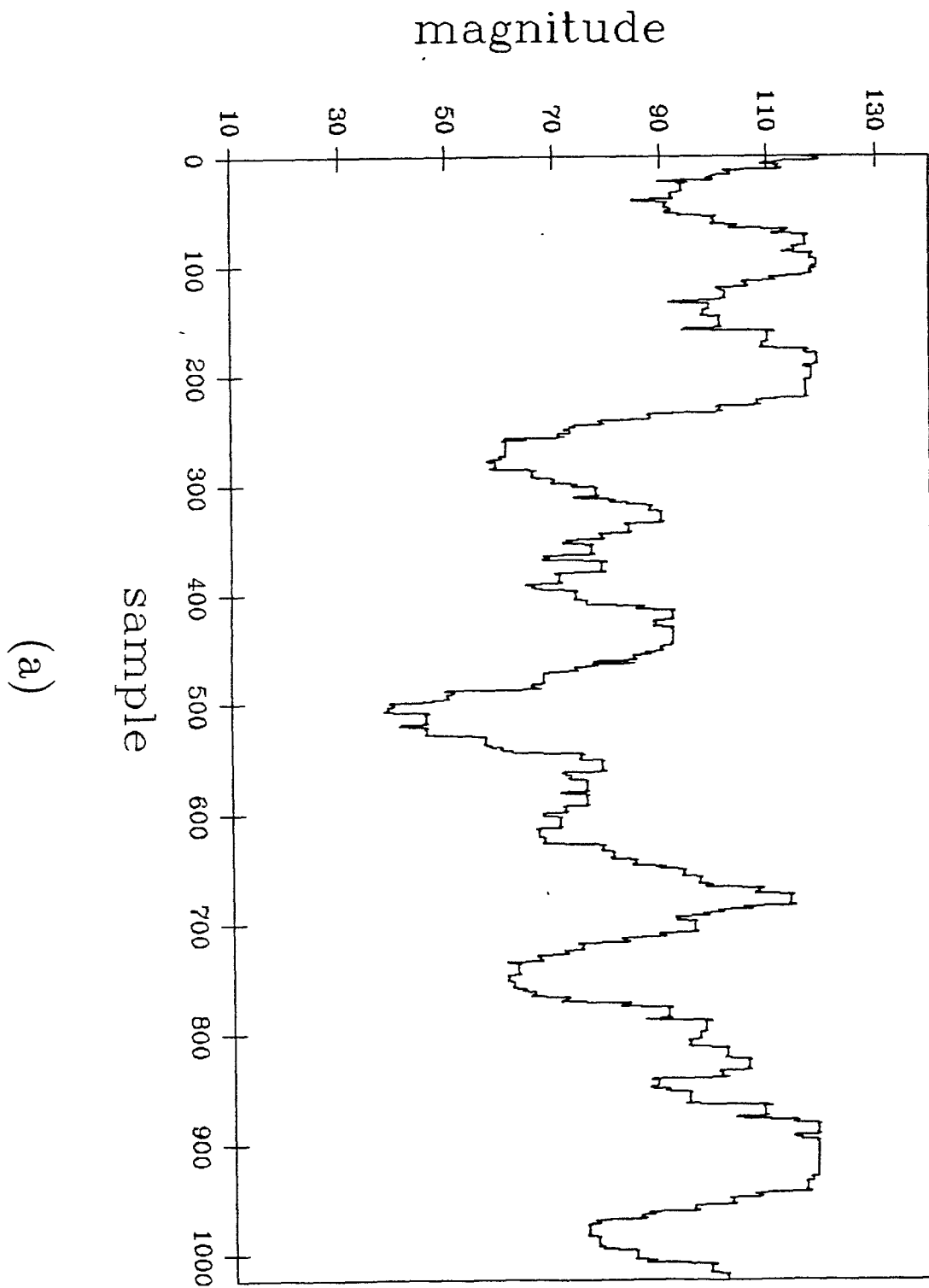


Fig. 5.5 Output signal obtained by filtering the corrupted signal shown in Fig. 2 by (a) LMS rule, window width = 11, and (b) Perceptron rule, window width = 11, respectively.

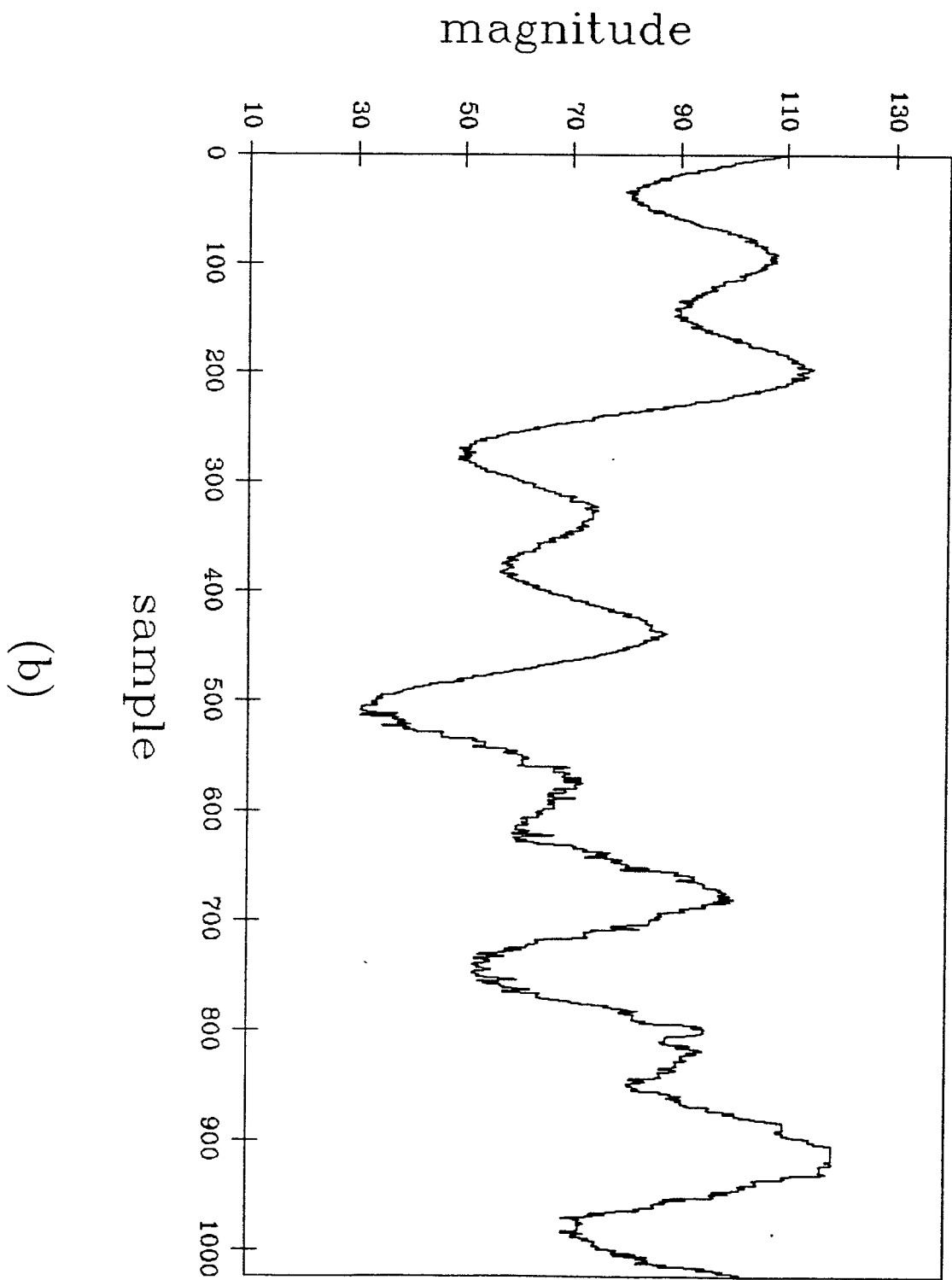


Fig. 5.5, continued.

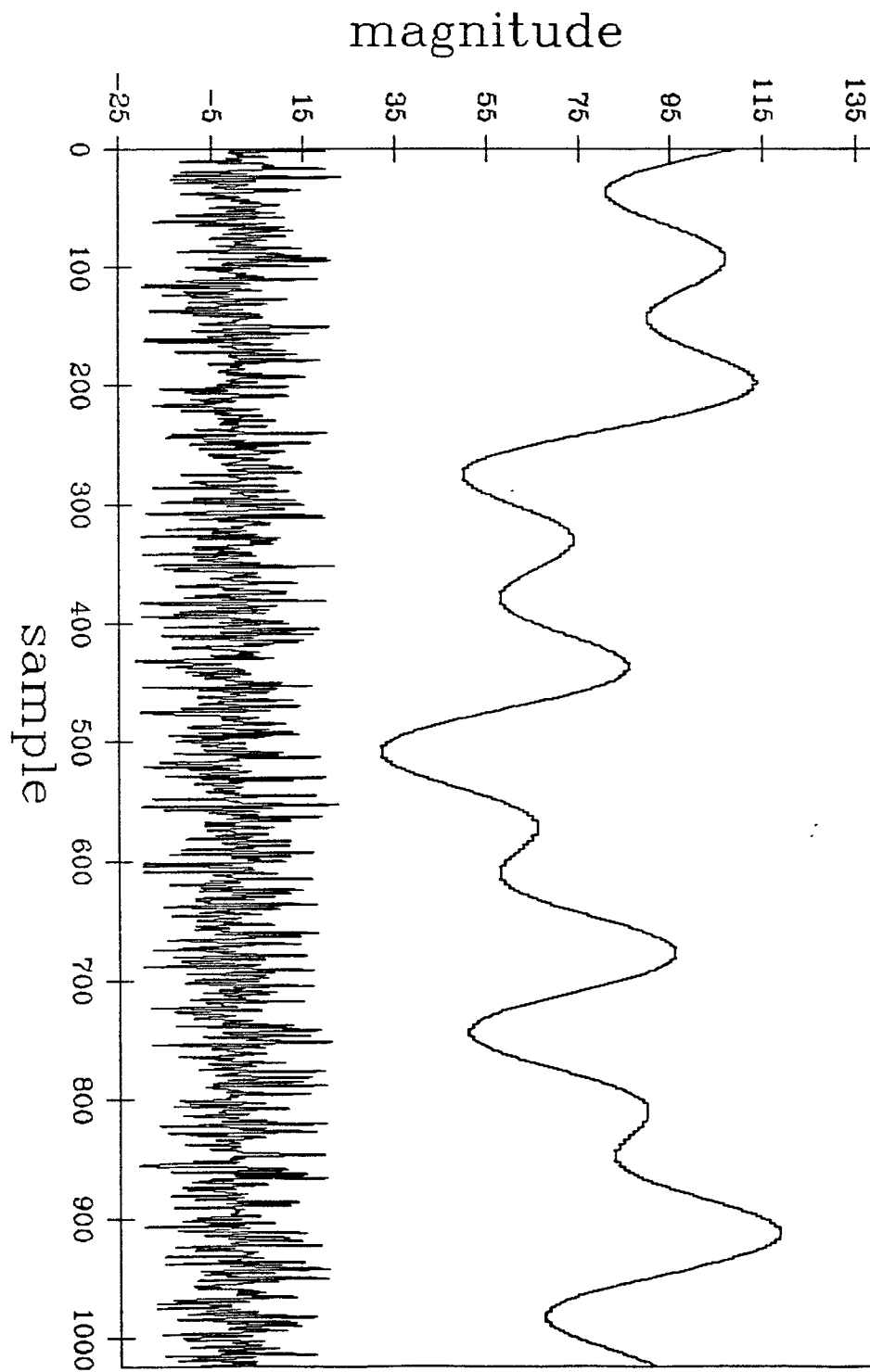


Fig. 5.6 Original signal (a combination of sinusoids) and  $\epsilon$ -mixture Gaussian noise are shown separately.

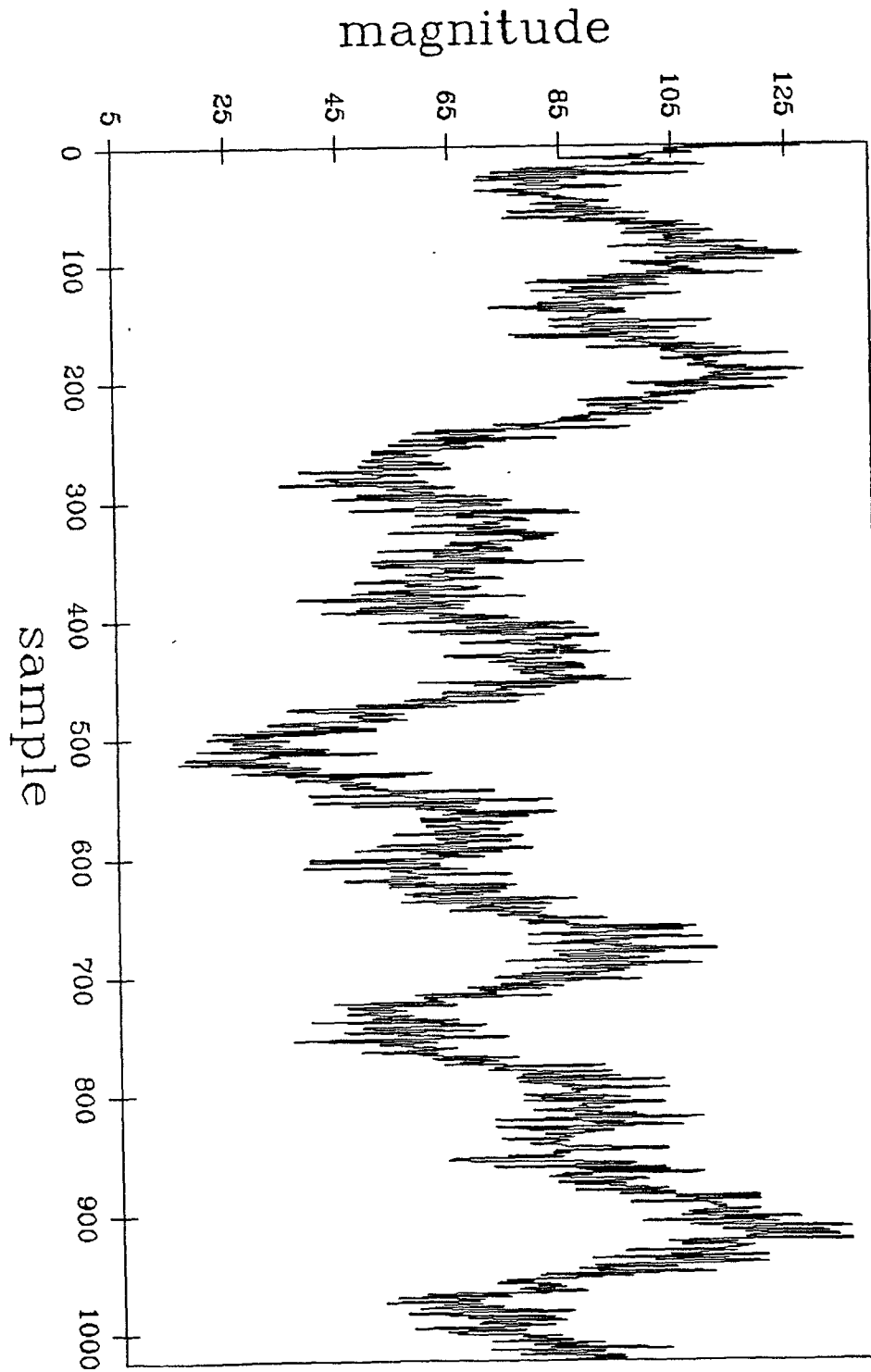


Fig. 5.7 Corrupted signal – a combination of sinusoids +  $\epsilon$ -mixture of Gaussian noise.

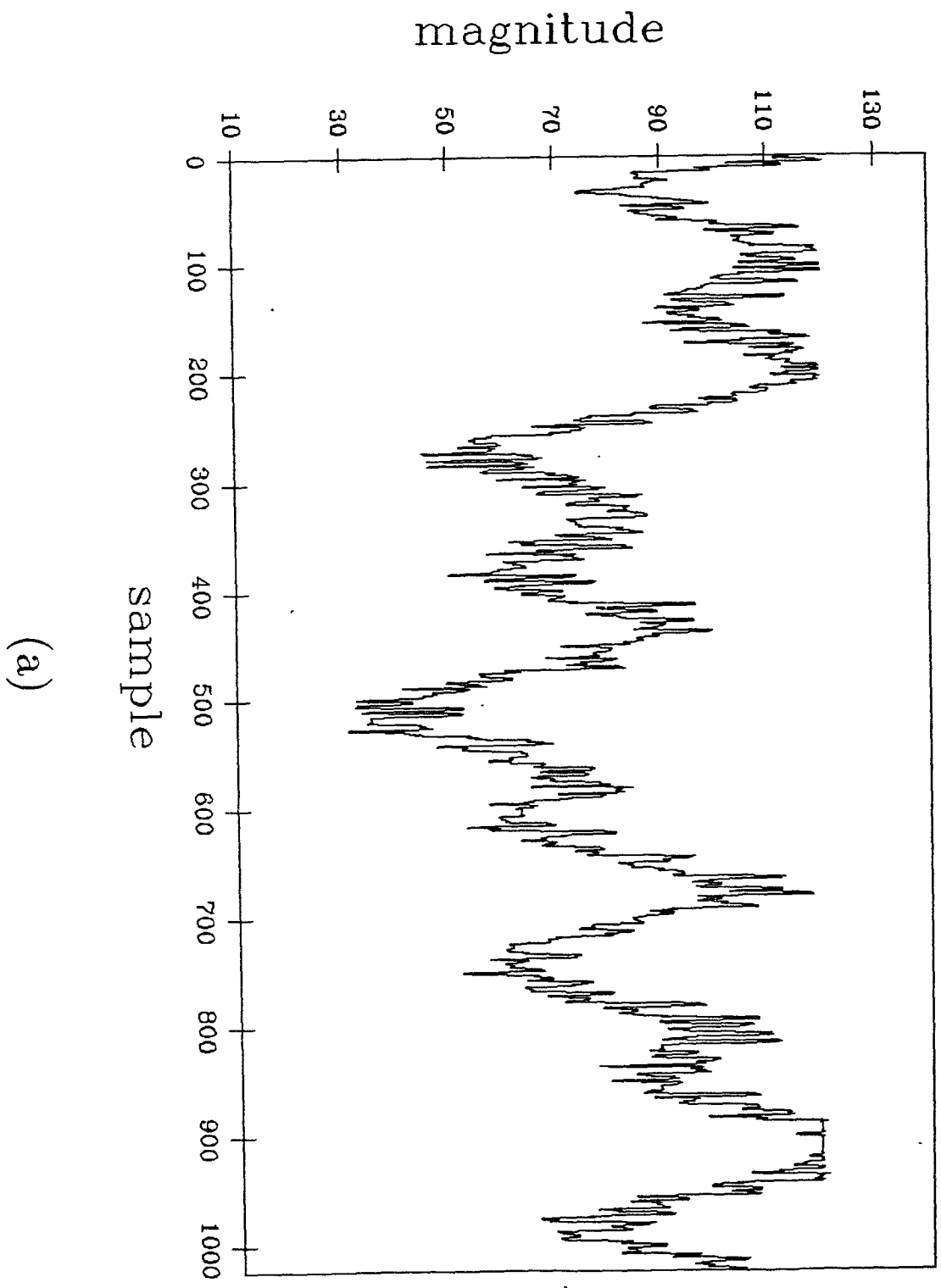


Fig. 5.8 Output signal obtained by filtering the corrupted signal shown in Fig. 7 by (a) LMS rule, window width = 3, and (b) Perceptron rule, window width = 3, respectively.

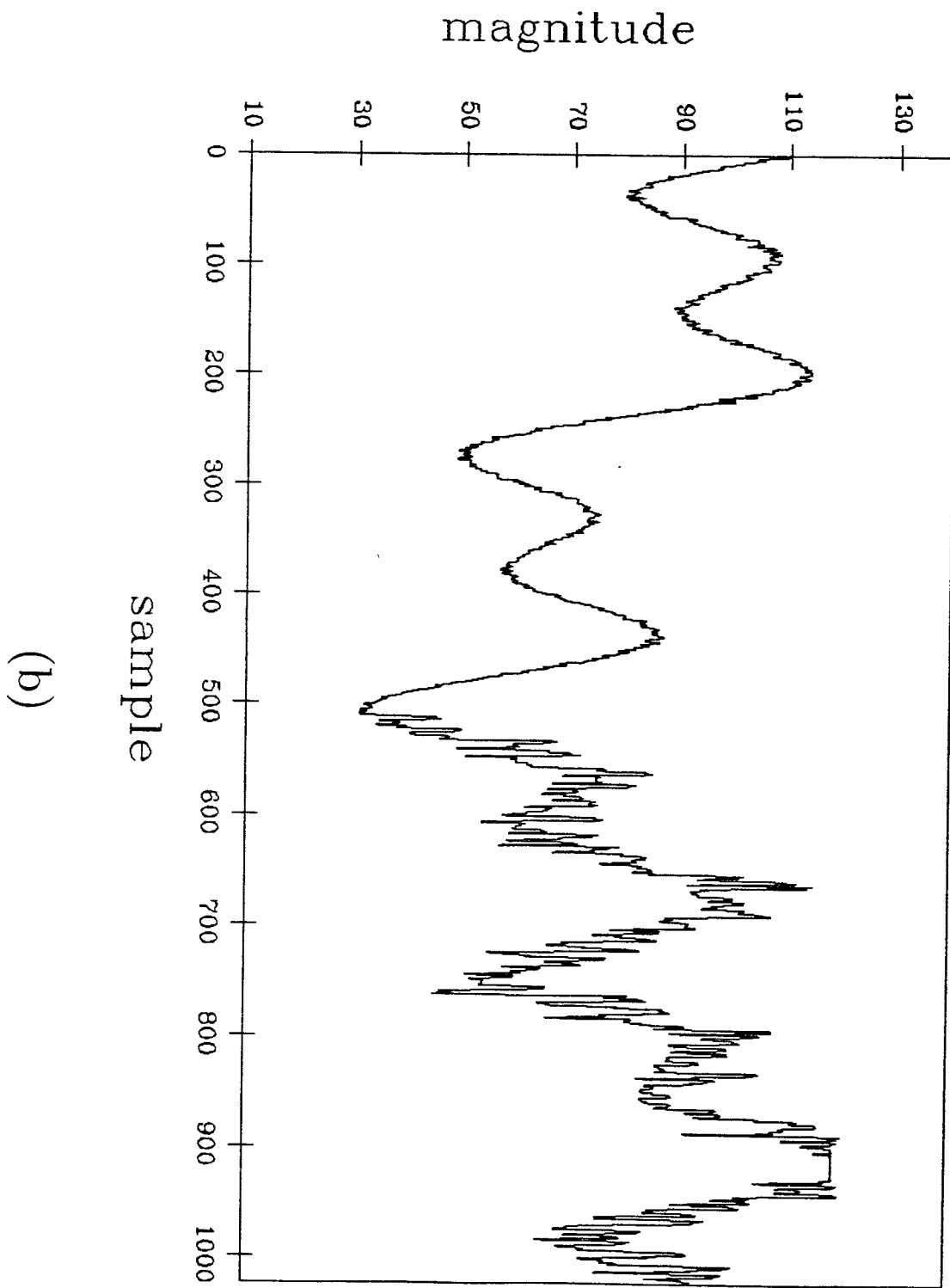


Fig. 5.8, continued.

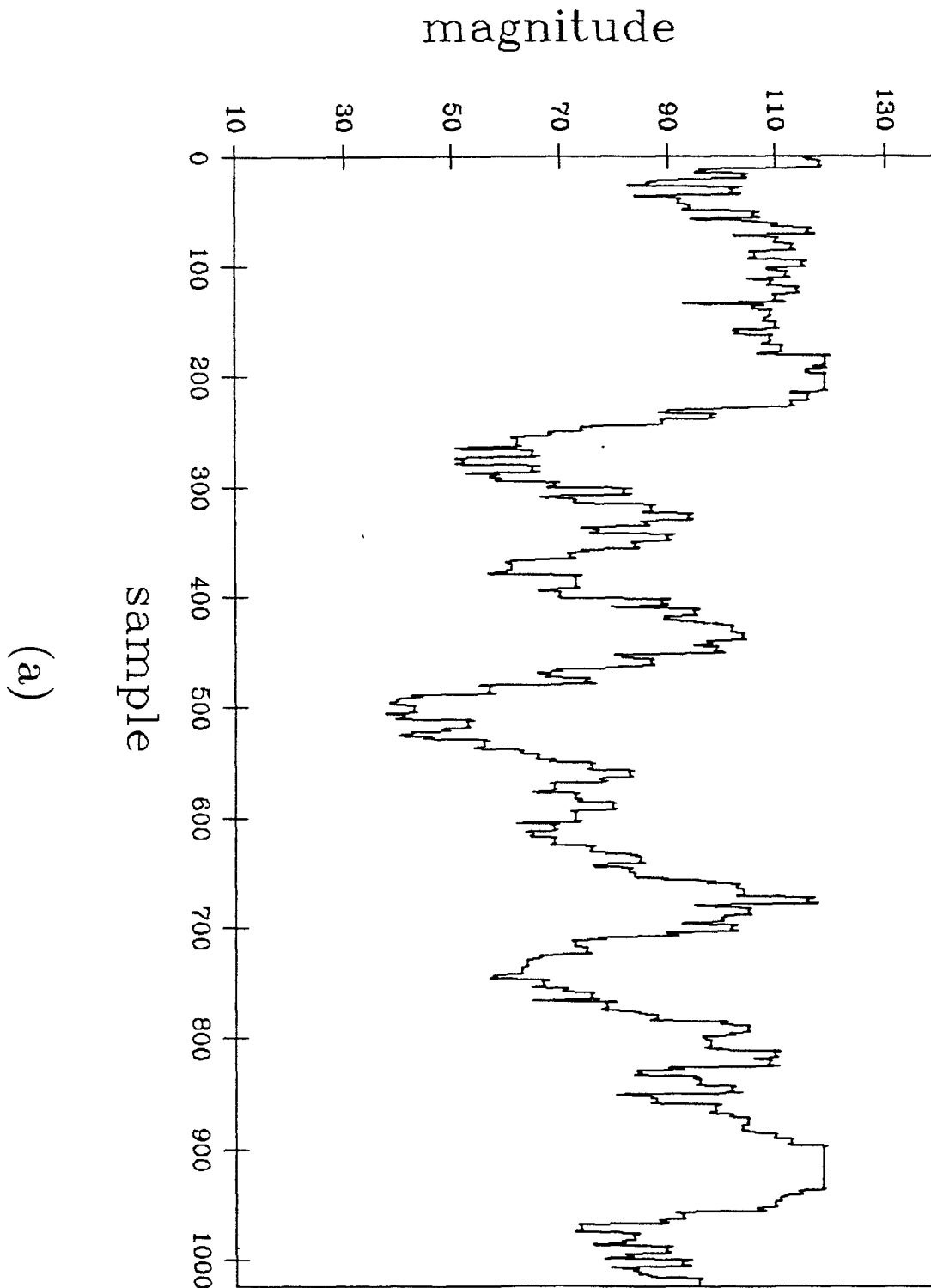
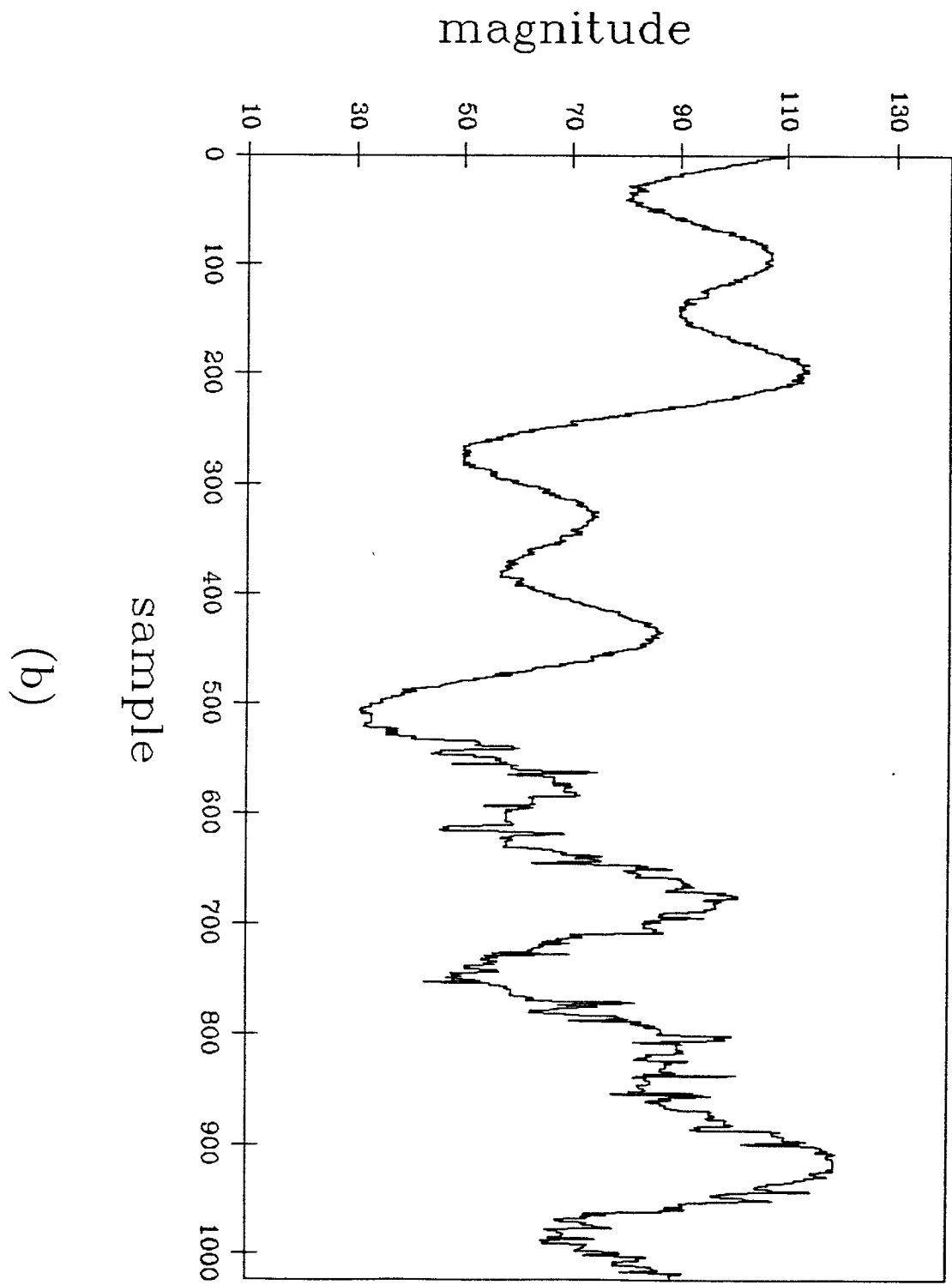


Fig. 5.9 Output signal obtained by filtering the corrupted signal shown in Fig. 7 by (a) LMS rule, window width = 7, and (b) Perceptron rule, window width = 7, respectively.



(b)

Fig. 5.9, continued.



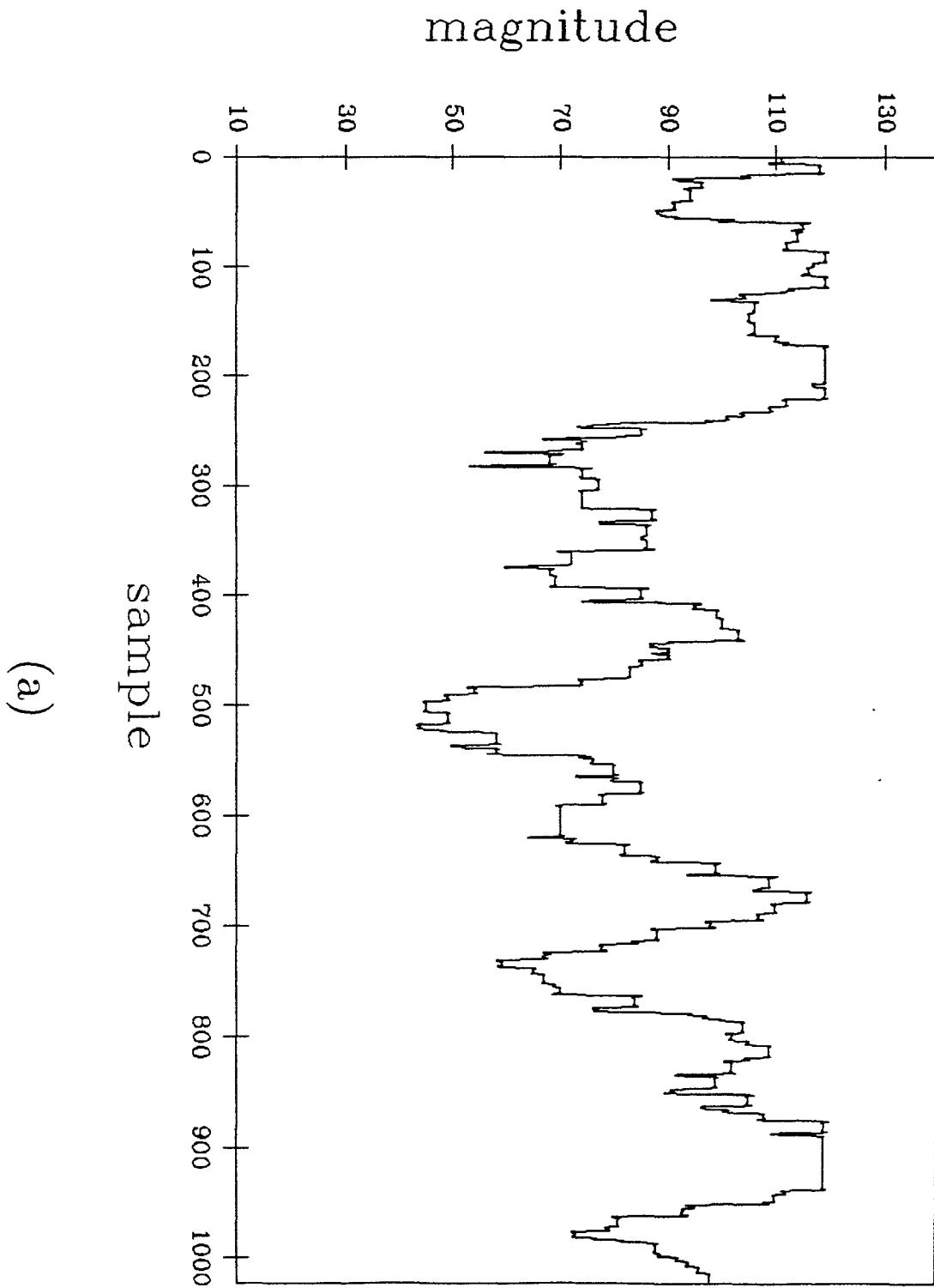


Fig. 5.10 Output signal obtained by filtering the corrupted signal shown in Fig. 7 by (a) LMS rule, window width = 11, and (b) Perceptron rule, window width = 11, respectively.

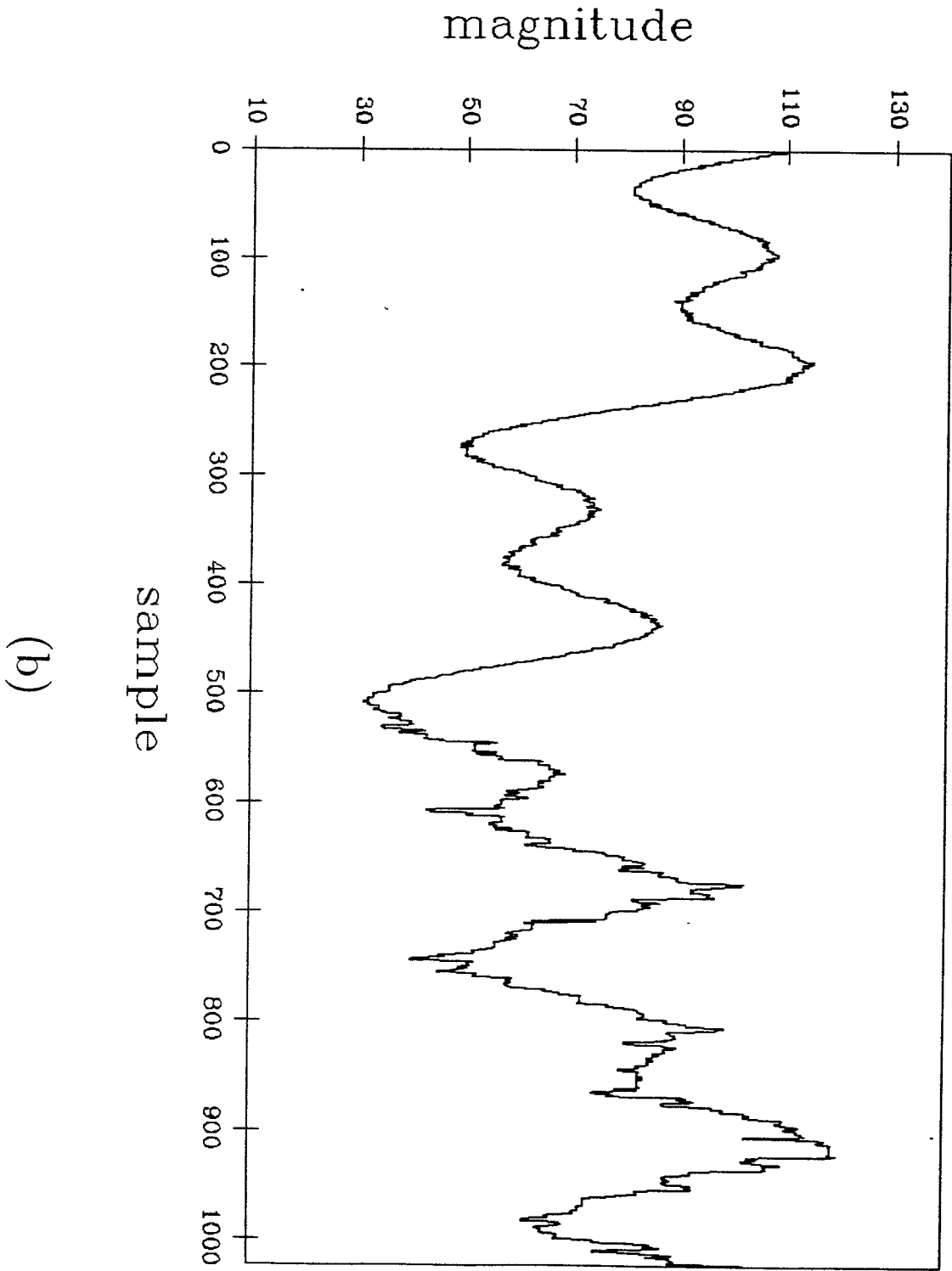


Fig. 5.10, continued.

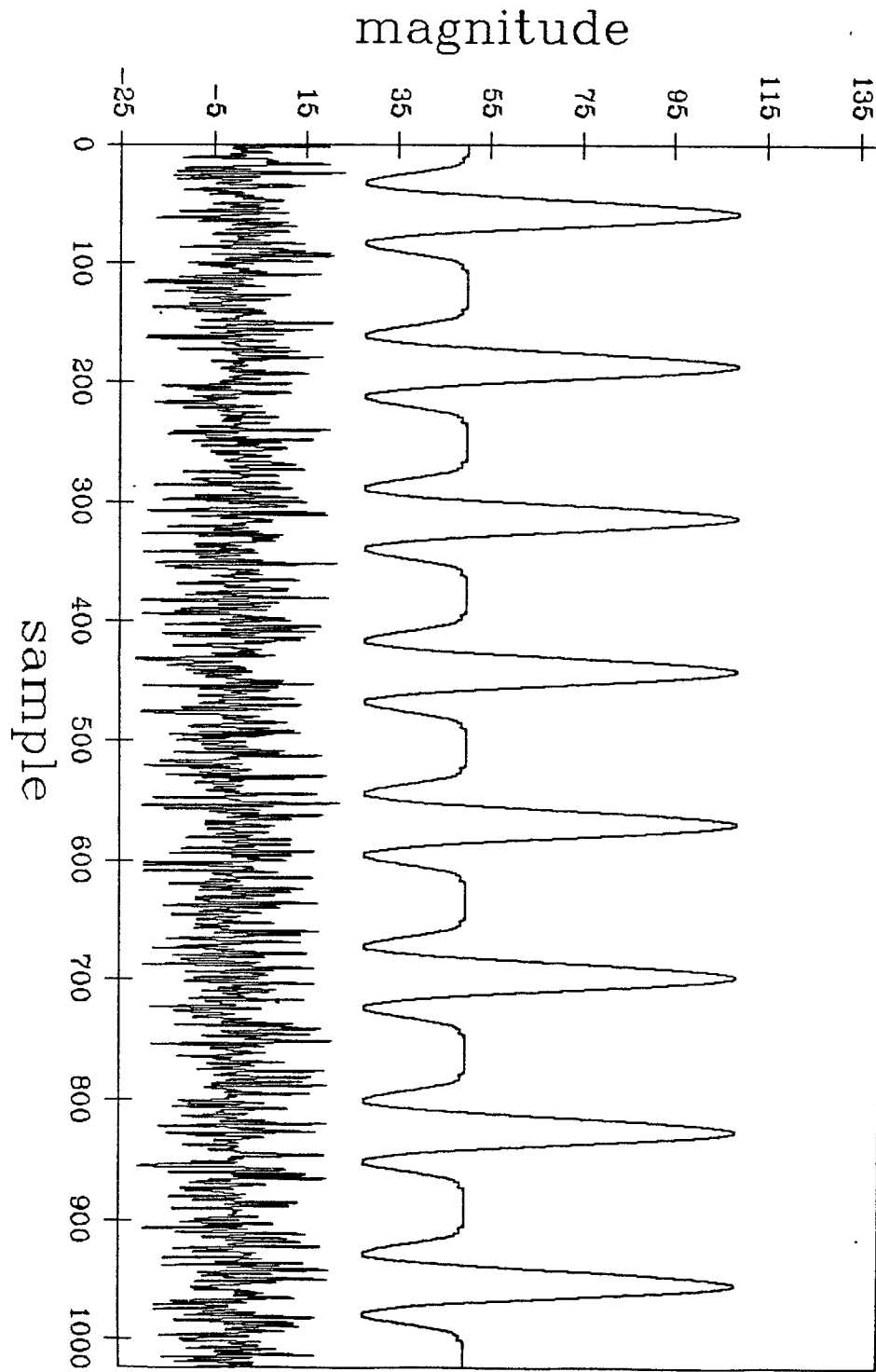


Fig. 5.11 “Mexican hat” signal and  $\epsilon$ -mixture Gaussian noise are shown separately.

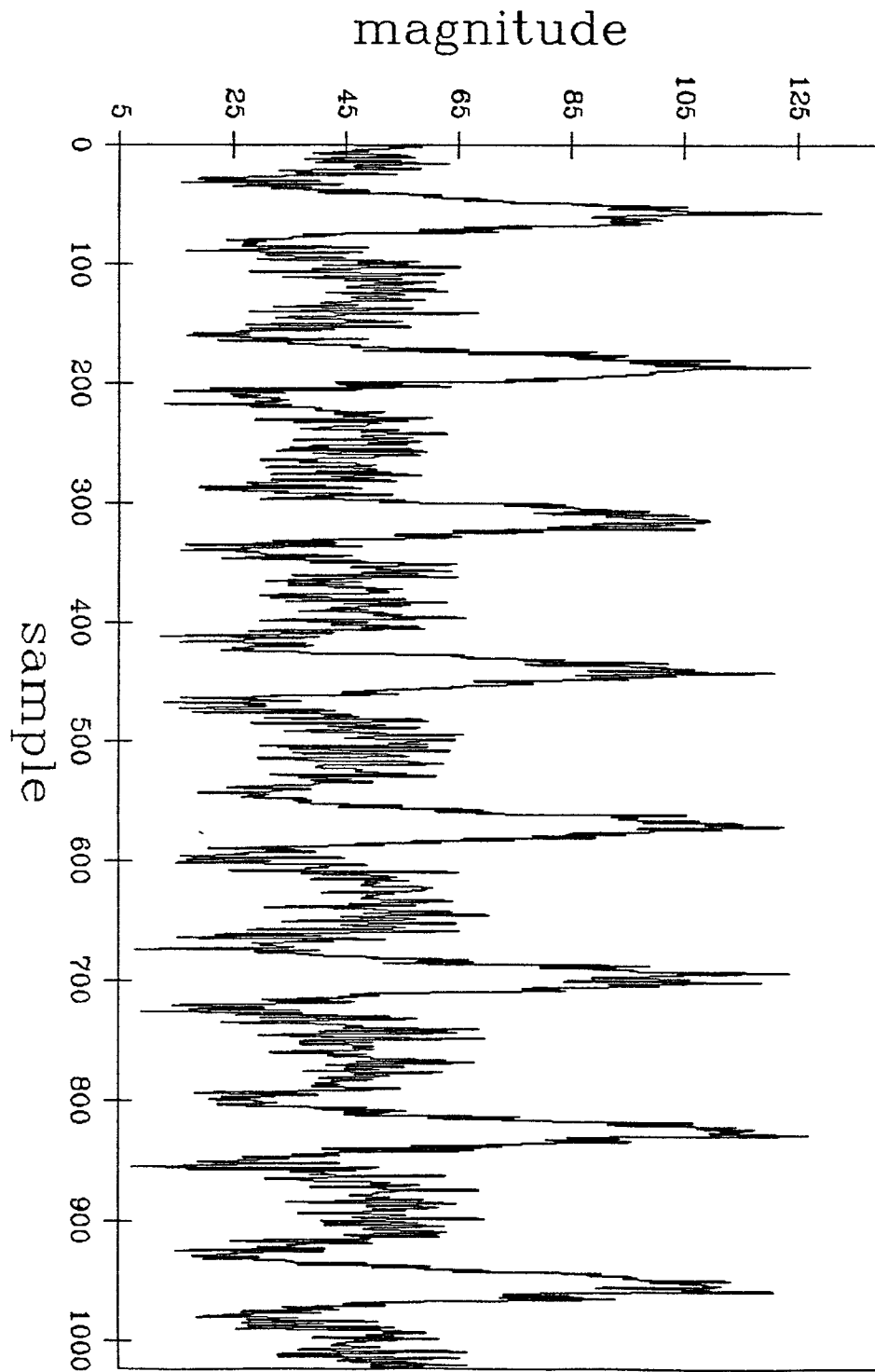


Fig. 5.12 Corrupted signal – “Mexican hat” +  $\epsilon$ -mixture of Gaussian Noise.

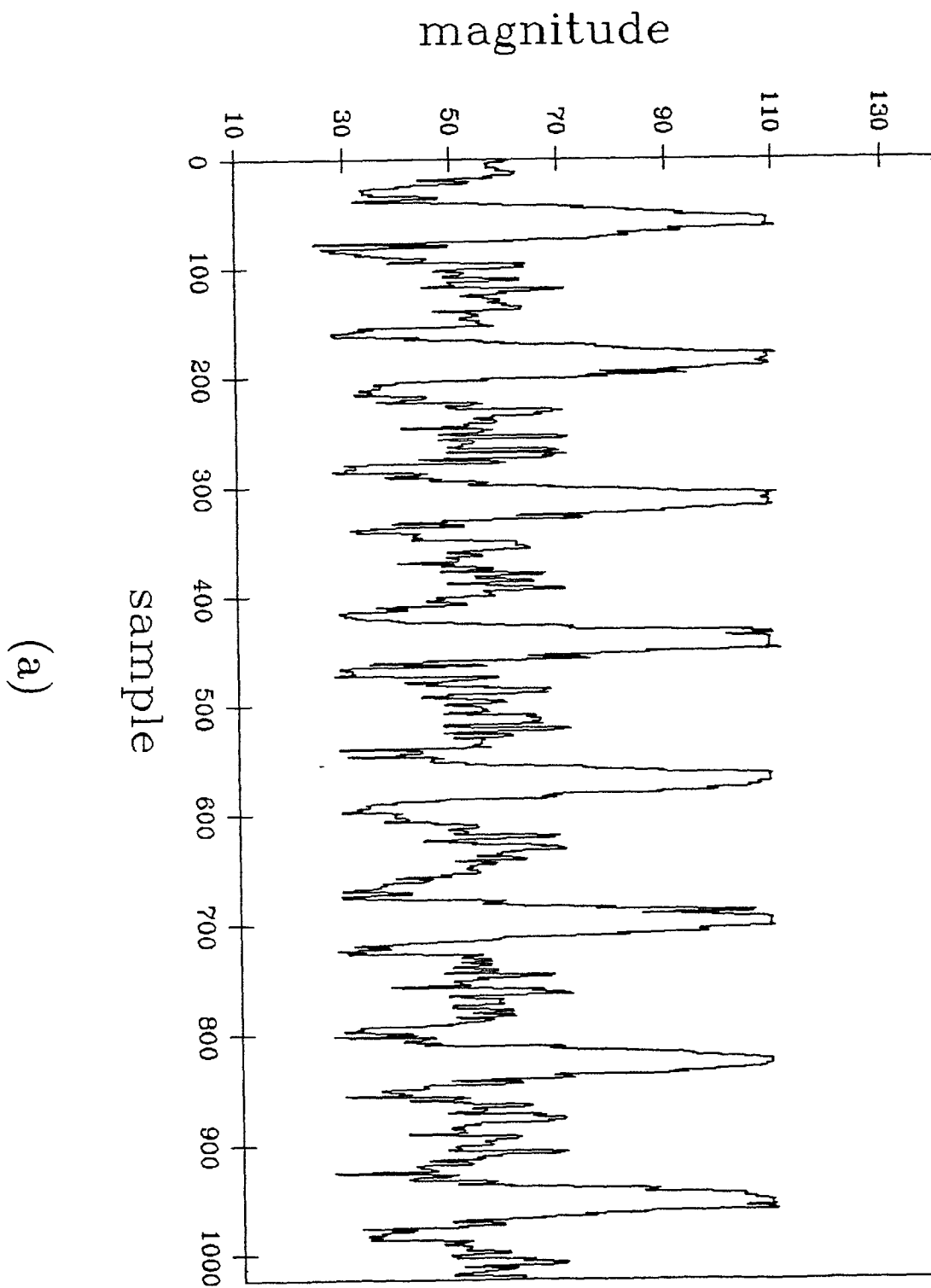


Fig. 5.13 Output signal obtained by filtering the corrupted signal shown in Fig. 12 by (a) LMS rule, window width = 3, and (b) Perceptron rule, window width = 3, respectively.

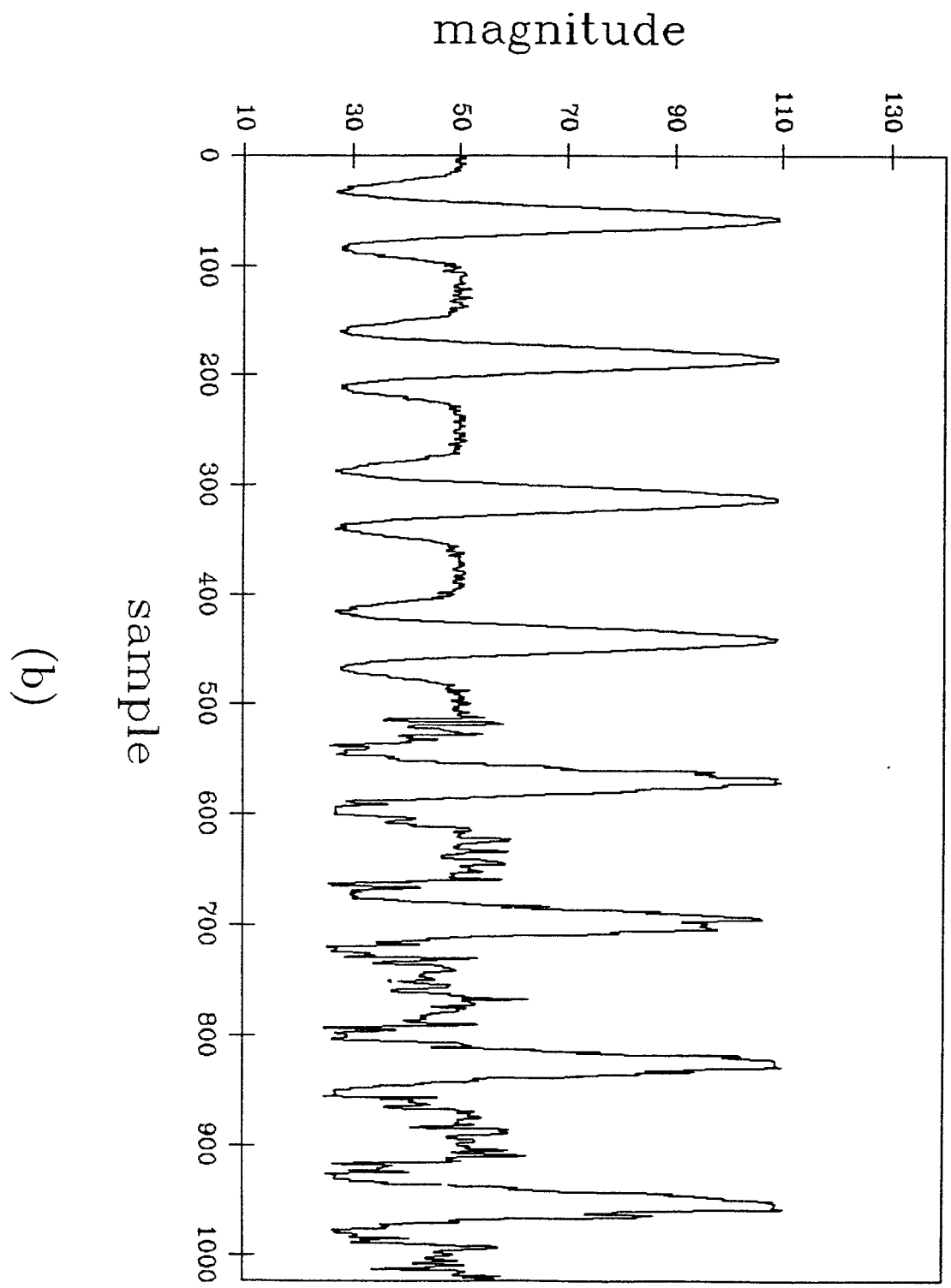
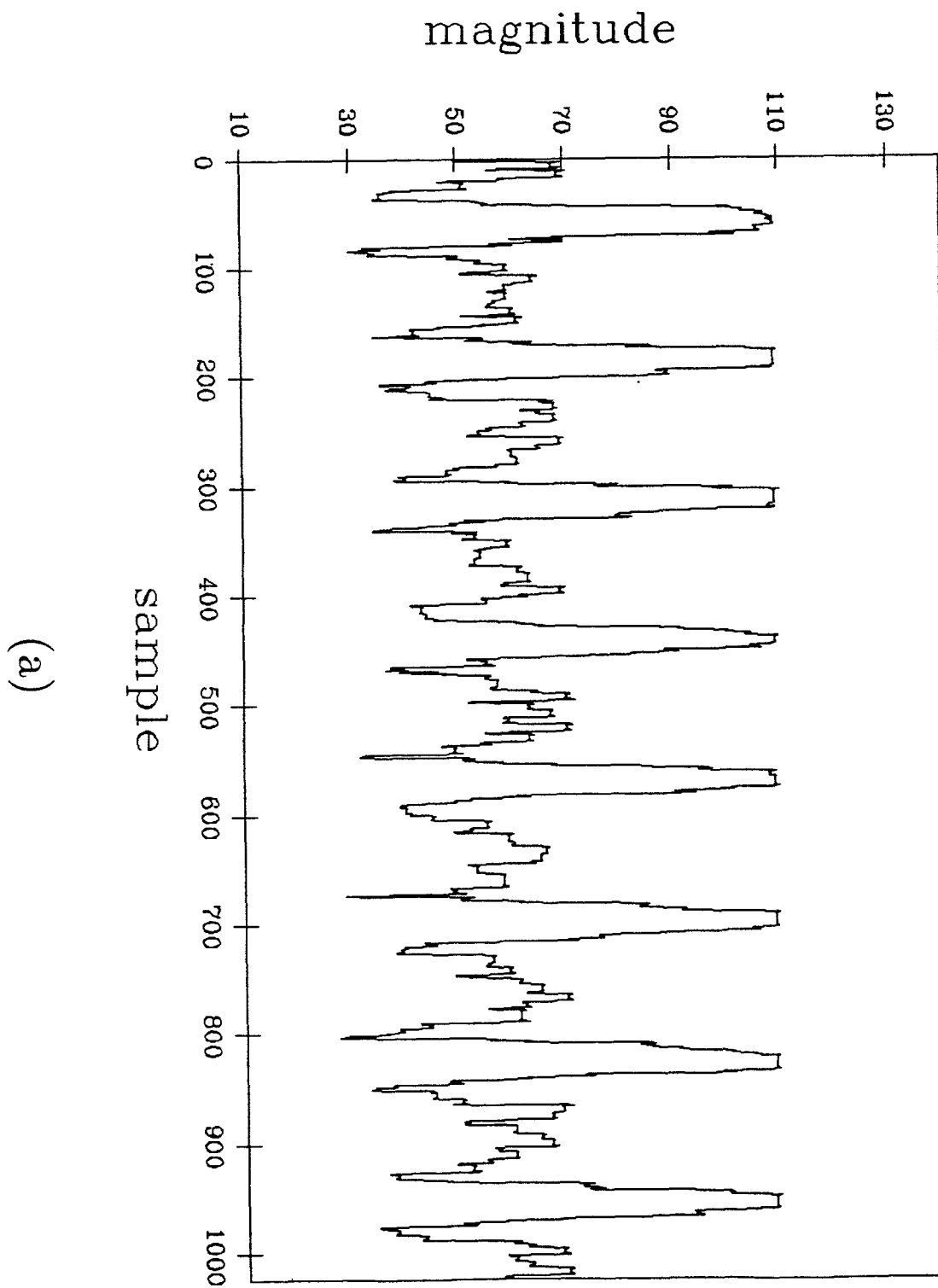


Fig. 5.13, continued.



(a)

Fig. 5.14 Output signal obtained by filtering the corrupted signal shown in Fig. 12 by (a) LMS rule, window width = 7, and (b) Perceptron rule, window width = 7, respectively.

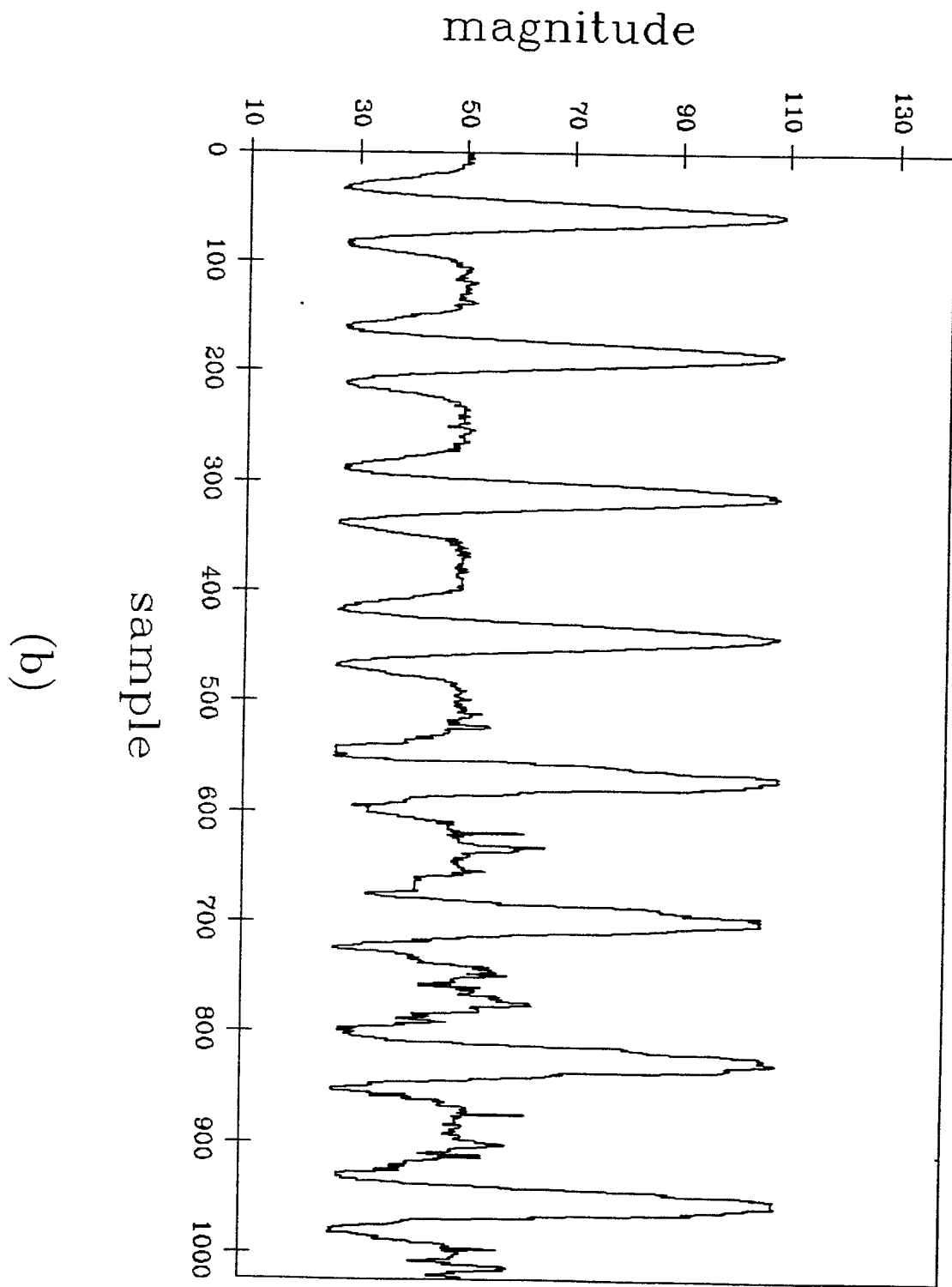


Fig. 5.14, continued.



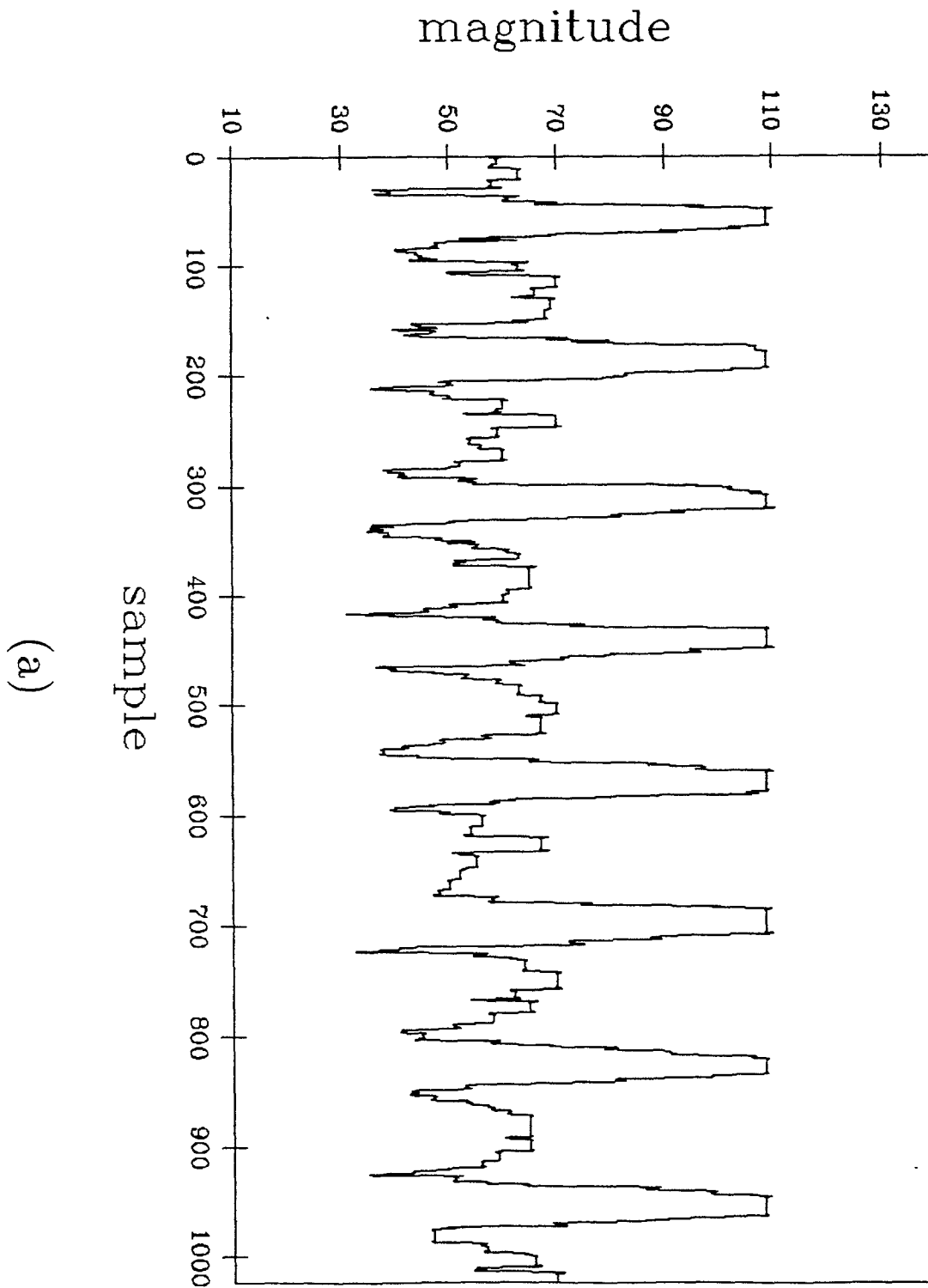


Fig. 5.15 Output signal obtained by filtering the corrupted signal shown in Fig. 12 by (a) LMS rule, window width = 11, and (b) Perceptron rule, window width = 11, respectively.

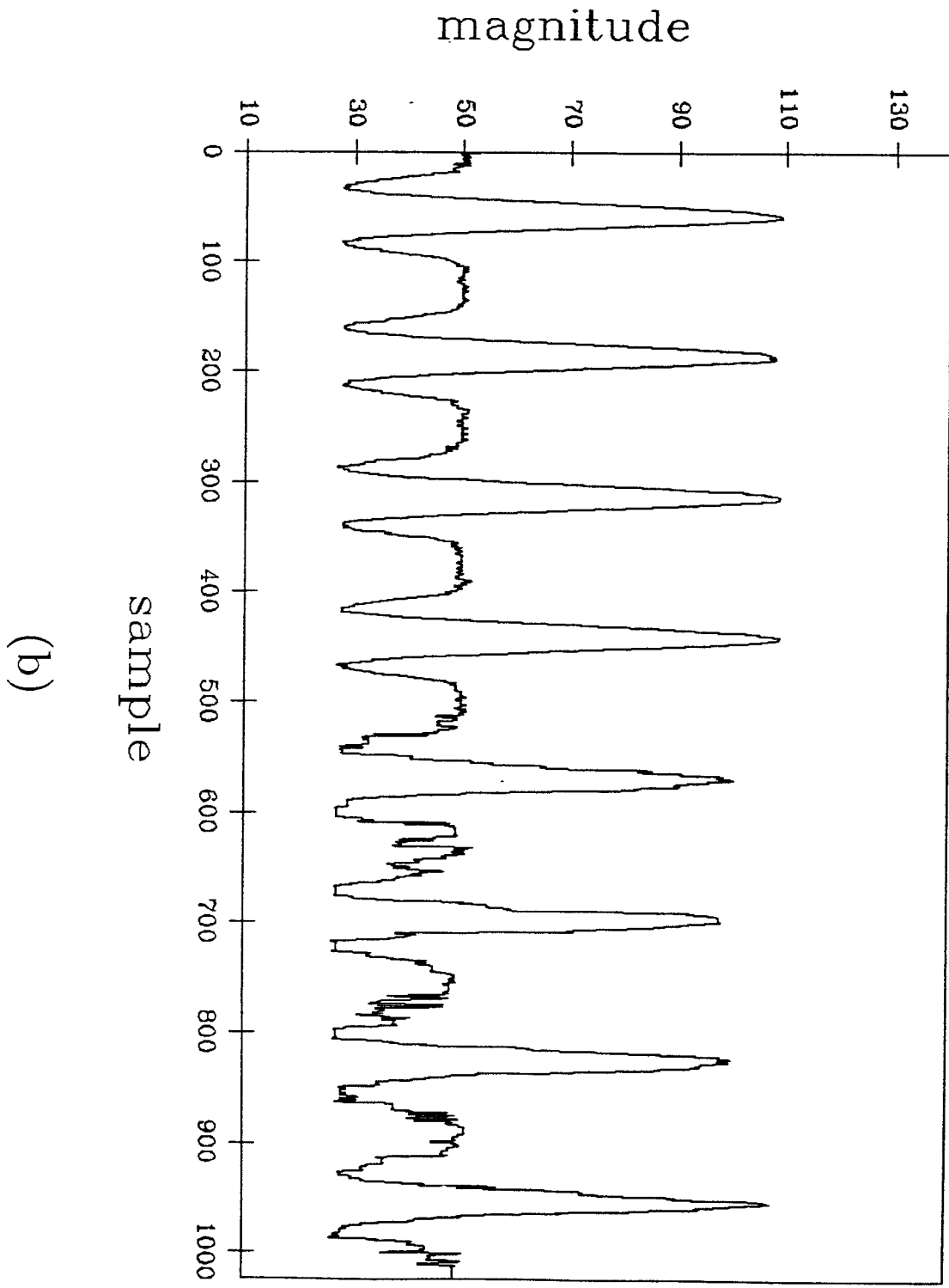


Fig. 5.15, continued.

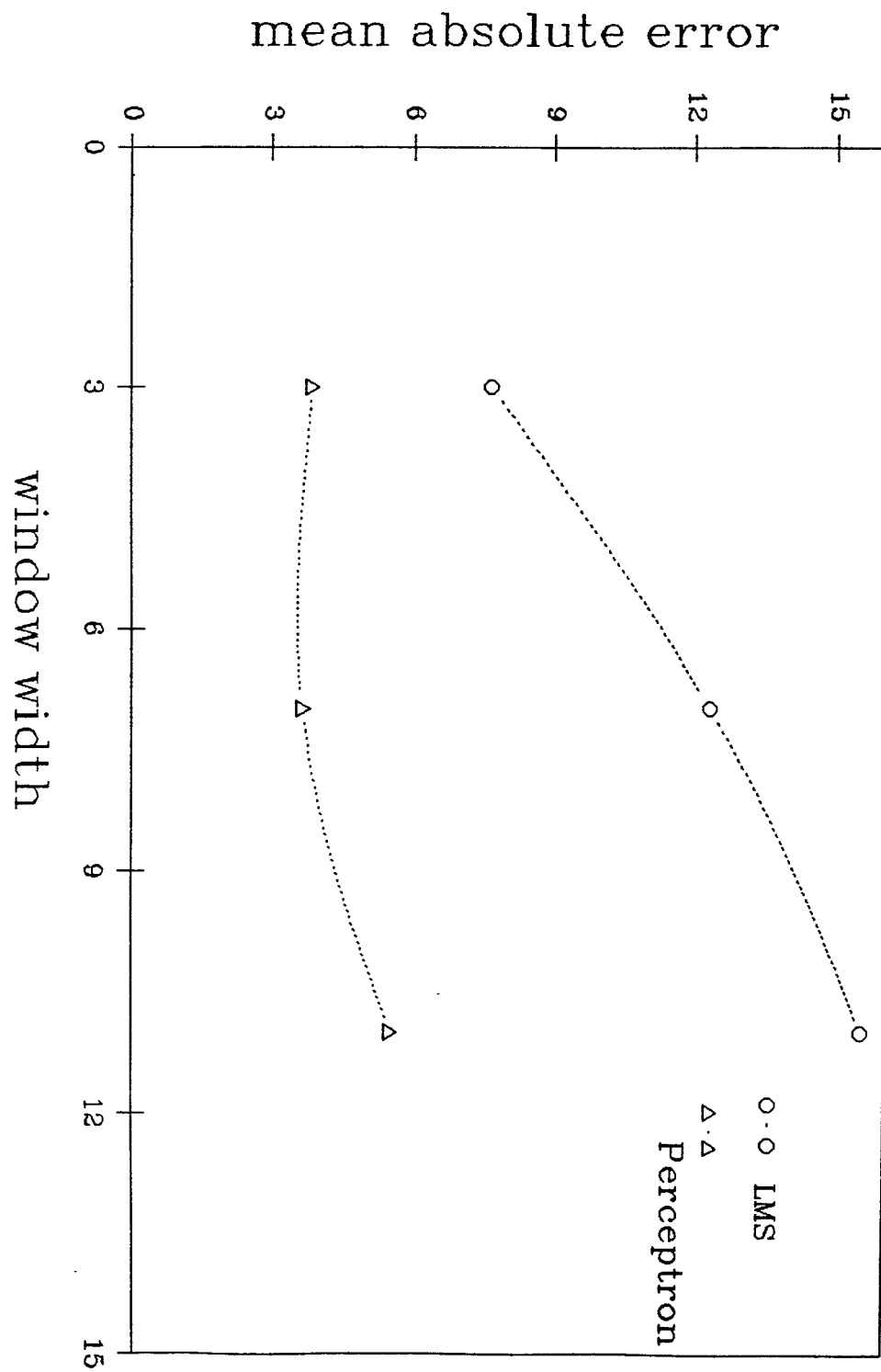


Fig. 5.16 Mean absolute error between the original “Mexican hat” signal and the signal corrupted by  $\epsilon$ -mixture noise.

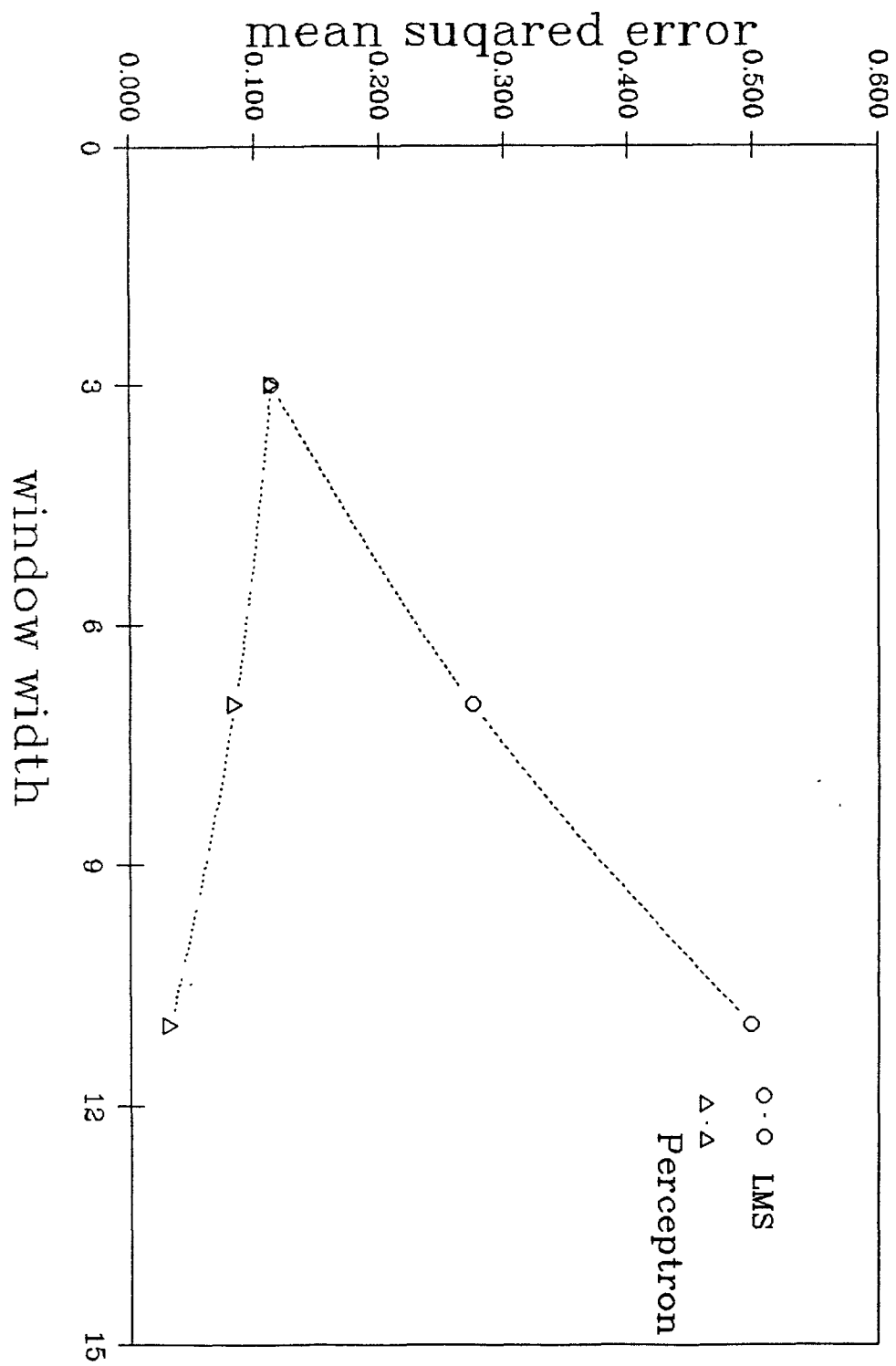


Fig. 5.17 Mean squared error between the original “Mexican hat” signal and the signal corrupted by Gaussian noise.

# Chapter 5

## Conclusions

A framework for configuring stack filters using LMS and Perceptron rules was established and tested. We have demonstrated through experimental results that our proposed algorithms perform the noise suppression task well. The current design only makes use of a simple single neuron. Further improvement is expected if a multi-layer network(multi-layer Perceptron) is employed.

Future research efforts include

- (1) Analyze the proposed adaptive filter structure mathematically.
- (2) Extend a single-neuron structure to a multi-layer neural network.
- (3) Implement adaptive filters using VLSI technology based on the properties

of stack filters.

## Bibliography

- [1] S. T. Alexander, *Adaptive Signal Processing Theory and Application*, New York, NY: Springer-Verlag New York Inc., 1986.
- [2] E. Ataman, V. K. Aatre and K. M. Wong, "Some Statistical Properties of Median Filters," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-29, pp. 1073-1075, Oct. 1981.
- [3] A. C. Bovik, T. S. Huang and D. C. Munson, "A Generalization of Median Filtering Using Linear Combinations of Order Statistics," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-31, pp. 1342-1349, Dec. 1983.
- [4] C. H. Chu, "A Genetic Algorithm Approach to the Configuration of Stack Filters," Proc. Intl. Conf. on Genetic Algorithms, George Mason University, June 4-7, 1989, pp. 218-224.
- [5] E. J. Coyle and J. H. Lin, "Stack Filters and the Mean Absolute Error Criterion," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-36, pp. 1244-1254, Aug. 1988.
- [6] S. C. Douglas and T. H. Y. Meng, "Optimum Error Nonlinearities for LMS Adaptation," ICASSP 90, Albuquerque, New Mexico, April 3-6, 1990.
- [7] S. C. Douglas and T. H. Y. Meng, "An adaptive Edge Detection Method Using a Modified Sigmoid-LMS Algorithm," 23rd Annual Asilomar Conf. on Signal, Syst., Comput., Asilomar, CA, Nov. 1989.
- [8] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, Menlo Park, CA: John Wiley & Sons. Inc., 1973.
- [9] J. P. Fitch, E. J. Coyle and N. C. Gallagher, "Median Filtering by Threshold Decomposition," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-32, pp. 1183-1188, Dec. 1984.
- [10] J. P. Fitch, E. J. Coyle and N. C. Gallagher, "Root Properties and Convergence Rates of Median Filters," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-33, pp. 230-239, Feb. 1985.

- [11] N. C. Gallagher and G. L. Wise, "A Theoretical analysis of the properties of Median Filter," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-29, pp. 1136-1141, Dec. 1981.
- [12] —, "Threshold Decomposition of Multidimensional ranked-order Operations," *IEEE Trans. Circuits Syst.*, vol. CAS-32, pp. 445-450, May 1985.
- [13] E. N. Gilbert, "Lattice-theoretic properties of frontal switching functions," *J. Math. Phys.*, vol. 33, pp. 57-67, Apr. 1954.
- [14] S. E. Hampson and D. J. Volper, "Disjunctive Models of Boolean Category Learning," *Biological Cybernetics*, vol. 56, pp. 121-137, 1987.
- [15] S. Haykin, *Adaptive Filter Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1986.
- [16] J. H. Lin, T. M. Selike and E. J. Coyle, "Adaptive Stack Filtering Under the Mean Absolute Error Criterion," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-38, pp. 938-954, Jun. 1990.
- [17] R. P. Lippmann, "An Introduction to Computing with Neural Nets," *IEEE Trans. Acoust., Speech, Signal Processing Magazine*, pp. 4-22, Apr. 1987.
- [18] R. Rosenblatt, *Principles of Neurodynamics*, New York, Spartan Books, 1959.
- [19] P. D. Wendt, E. J. Coyle and N. C. Gallagher, "Stack Filters," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 898-911, Aug. 1986.
- [20] B. Widrow, R. G. Winter and R. A. Baxter, "Layered Neural Nets for Pattern Recognition," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-36, pp. 1109-1118, Jul. 1988.