LAGRANGIAN MULTIPLIER IN THE PEARLMUTTER ALGORITHM AND DYNAMIC NEURAL NETWORKS

by

JUAN SUN, B.E.

A THESIS

IN

COMPUTER SCIENCE

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements for
the Degree of

MASTER OF SCIENCE

Approved

May, 1998

ACKNOWLEDGEMENTS

First of all, I would like to express my gratitude to Dr. William J. B. Oldham, my advisor, for his expertise, effort, advice, encouragement and patience throughout my research.

I also want to express my thankfulness to Dr. Susan Mengel, for her willingness to be my committee member and her expertise, help, advice and suggestions.

I want to thank my parents and my parents-in-law, for their love and encouragement. Also, I want to thank Pilar Gomez-Gil, for her help and kindness.

Finally, I would like to express my special thanks to my husband for his love, understanding, and support.

TABLE OF CONTENTS

ACKNOWLEDGES ii

TABLE OF CONTENTS.....iii

LIST OF TABLES.....v

LIST OF FIGURES.....vi

CHAPTER

I. INTRODUCTION.....1

1.1 Neural Networks and Recurrent Neural Networks.....1

1.2 Time Series.....7

1.3 Sine Wave Data.....8

1.4 Electrocardiogram Data (ECG Data).....8

1.5 Mackey-Glass Data.....9

1.6 Neural Networks and Their Training.....10

1.7 Objective.....11

1.8 Organization of the Thesis.....12

II. DESCRIPTION OF PEARLMUTTER ALGORITHM.....13

2.1 The Pearlmutter Algorithm.....13

III. INVESETGATING LAGRANGIAN MULTIPLIER USING SINE WAVE DATA.....17

3.1 Neural Network Architecture in Sine Function Learning.....17

3.2 Sine Data Description.....19

3.3 Case Studies of Sine Wave Learning.....19

3.3.1 Case 1: step size = 0.1.....21

3.3.2 Case 2: step size = 0.05.....35

3.3.3 Case 3: step size =0.01.....48

3.3.4 Comparison of the results of different step size in sine wave.....62

IV. INVESTIGATING LAGRANGIAN MULTIPLIER USING ECG DATA.....64

4.1 Neural Network Architecture in ECG Data Learning.....64

4.2 ECG Data Description.....65

4.3 Case Studies of ECG Data Learning.....65

4.3.1 Case 1: step size = 1/90.....65

4.3.2 Case 2: step size = 1/360.....72

4.3.3 Comparison of ECG data learning using two different step sizes.....77

V. INVESTIGATING LAGRANGIAN MULTIPLIER USING MACKEY-GLASS DATA.....78

5.1 Architecture Description in Mackey-Glass Data Learning.....78

5.2 Mackey-Glass Data Description.....79

5.3 Case Study Using Step Size Δt of 0.9.....79

VI. CONCLUSION AND SUGGESTION.....86

REFERENCES.....91

LIST OF TABLES

3.1 The illustration of a linear function y=2*x, and its first-order difference and the second-order difference.....20

3.2 Comparison of training among three case studies of sine wave learning.....62

4.1 Comparison of training among two case studies of ECG data learning.....77

5.1 Results of Mackey-Glass data learning.....80

LIST OF FIGURES

1.1 An example of feed-forward neural networks.....3

1.2 An example of recurrent neural networks.....4

1.3 An example of fully connected recurrent neural networks.....4

2.1 The error calculation.....15

3.1 Fully recurrent neuron network model used in sine data investigation.....18

3.2 Training result of sine wave data of step size =0.1.....22

3.3 Z value of the output neuron with step size =0.1.....23

3.4 Z value of the hidden neuron 1 with step size =0.1.....24

3.5 Z value of the hidden neuron 2 with step size =0.1.....25

3.6 $Z^{(1)}$ value of the output neuron with step size =0.1.....26

3.7 $Z^{(1)}$ value of the hidden neuron 1 with step size =0.1.....27

3.8 $Z^{(1)}$ value of the hidden neuron 2 with step size =0.1.....28

3.9 $Z^{(2)}$ value of the output neuron with step size =0.1.....29

3.10 $Z^{(2)}$ value of the hidden neuron 1 with step size =0.1.....30

3.11 $Z^{(2)}$ value of the hidden neuron 2 with step size =0.1.....31

3.12 Training result of sine wave data of step size =0.05.....36

3.13 Z value of the output neuron with step size =0.05.....37

3.14 Z value of the hidden neuron 1 with step size =0.05.....38

3.15 Z value of the hidden neuron 2 with step size =0.05.....39

3.16 $Z^{(1)}$ value of the output neuron with step size =0.05.....40

3.17 $Z^{(1)}$ value of the hidden neuron 1 with step size =0.05.....41

3.18 $Z^{(1)}$ value of the hidden neuron 2 with step size =0.05.....42

3.19 $Z^{(2)}$ value of the output neuron with step size =0.05.....43

3.20 $Z^{(2)}$ value of the hidden neuron 1 with step size =0.05.....44

3.21 $Z^{(2)}$ value of the hidden neuron 2 with step size =0.05.....45

3.22 Training result of sine wave data of step size =0.01.....50

3.23 Z value of the output neuron with step size =0.01.....51

3.24 Z value of the hidden neuron 1 with step size =0.01.....52

3.25 Z value of the hidden neuron 2 with step size =0.01.....53

3.26 $Z^{(1)}$ value of the output neuron with step size =0.01.....54

3.27 $Z^{(1)}$ value of the hidden neuron 1 with step size =0.01.....55

3.28 $Z^{(1)}$ value of the hidden neuron 2 with step size =0.01.....56

3.29 $Z^{(2)}$ value of the output neuron with step size =0.01.....57

3.30 $Z^{(2)}$ value of the hidden neuron 1 with step size =0.01.....58

3.31 $Z^{(2)}$ value of the hidden neuron 2 with step size =0.01.....59

4.1 The fully recurrent neuron network model used ECG data investigation.....64

4.2 Training result of the ECG data of step size =1/90.....68

4.3 Z value of the output neuron of ECG data with step size =1/90.....69

4.4 $Z^{(1)}$ value of the output neuron with step size =1/90.....70

4.5 $Z^{(2)}$ value of the output neuron with step size =1/90.....71

4.6 Training result of the ECG data of step size =1/360.....73

4.7 Z value of the output neuron under step size =1 360.....74

4.8 $Z^{(1)}$ value of the output neuron with step size =1 360.....75

4.9 $Z^{(2)}$ value of the output neuron with step size =1 360.....76

5.1 The fully recurrent neuron network model used in Mackey-Glass data investigation.....78

5.2 The training result of the Mackey-Glass data of step size =0.9.....82

5.3 Z value of the output neuron under step size =0.9.....83

5.4 $Z^{(1)}$ value of the output neuron with step size =0.9.....84

5.5 $Z^{(2)}$ value of the output neuron with step size =0.9.....85

CHAPTER I

INTRODUCTION

Time series prediction using dynamic neural networks has been studied by many researchers (Christiansen, 1995; Connor, Atlas and Martin, 1992). Many of them have used recurrent neural networks in time series prediction and the results proved that for certain cases the recurrent neural networks can do a good job. For other cases it has been found to be very difficult to train the network. The Pearlmutter algorithm is one of the learning algorithms in recurrent neural network training, and it has been used by a number of researchers and proven to be successful for some examples. In this paper, we investigated the Lagrangian multiplier equations of the Pearlmutter algorithm. Although as stated, a number of researchers have employed this algorithm, there has not been much reported about the behavior of the Lagrangian multiplier equation and what role it plays in the learning. We looked at the behavior of this equation in cases where learning took place and in cases where it did not.

1.1 Neural Networks and Recurrent Neural Networks

Generally speaking, a neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

1. Knowledge is acquired by the network through a learning process

2. Interneuron connection strengths known as synaptic weights are used to store the knowledge (Haykin, 1994).

A neural network can be used as a predictor which uses past experience to predict future behavior and provide a solution to the temporal problem. Learning to predict is one of the most basic tasks in learning (Haykin, 1994).

Neural networks consist of many connected neurons. We can consider that neurons are nerve cells in the human brain, and that neural networks are networks of these cells. Thus, the brain can be regarded as an example of a natural neural network. The motivation for the classical interest in the subject of artificial neural networks (ANN) was rooted in anatomical and physiological considerations. The computational process envisioned with ANNs is as follows: An "Artificial neuron" (AN) receives inputs from a number of other ANs or from an external stimulus. A weighted sum of these inputs is the input to the neuron and constitutes the argument to an activation function of the neuron. The resulting value of the activation function is the output of the AN. The output connections define the flow of information in the network and are called the architecture of the network. The neurons and their interconnections are the architecture of the network.

Useful architectural configurations of neural networks include single-layer, multi-layer, feed-forward, and feedback. The weighted connections in these architectures play such an important role that these networks are called connectionist models of computation. There are two types of neural networks depending on the connections between nodes in neural networks: feed-forward neural networks and recurrent neural

networks. In feed-forward neural networks the information flow is all in the forward direction, i.e., from the input neurons to output neurons. The connection between pairs of nodes is uni-directional. A three-layer feed-forward neural network is shown in Figure 1.1. In recurrent neural networks there is at lease one feed back loop, that is, in a recurrent network, the connections between nodes can be in the forward direction or backward direction and they can be bi-directional and can also contain self-loops. A three-layer recurrent neural network is shown in Figure 1.2. In recurrent neural network if every neuron connects to any other neuron in the network, it is a fully connected recurrent neural network. A fully connected recurrent neural network consisting of three neurons is shown in Figure 1.3. In a recurrent network, some neurons are output neurons and some are input neurons. It is possible for a neuron to support both input and output functions. Neurons that are neither input nor output neurons are referred to as hidden neurons. Hidden neurons are active in the learning and prediction process, but have no direct connection to the environment.

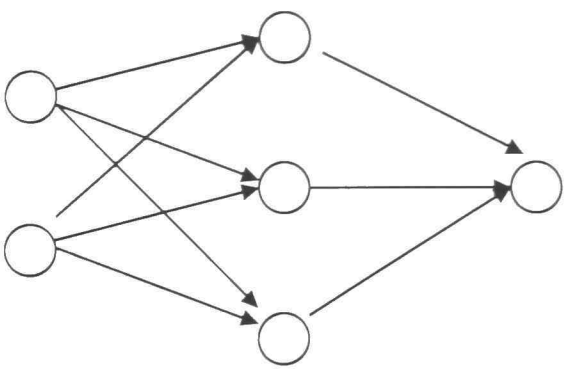

Figure 1.1: An example of feedforward neural networks

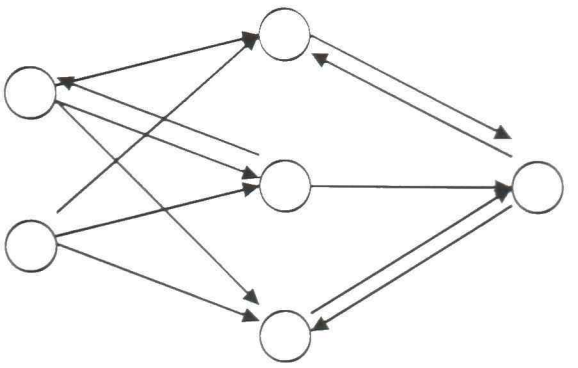

Figure 1.2: An example of recurrent neural networks

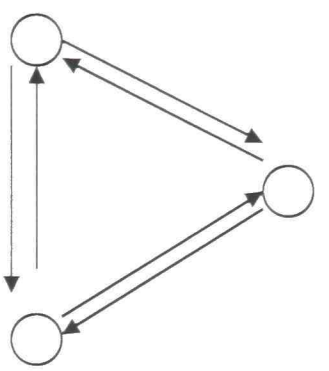

Figure 1.3. An example of fully connected recurrent neural networks

Feed-forward neural networks have been studied by many researchers and have been explained mathematically. One weakness of feed-forward neural network is its incapability of dealing with repetitive dynamics. The potential of recurrent architectures for dealing with temporal behavior is then the motivation for exploring recurrent architectures. Much work has been done by the researchers in the area of recurrent neural networks (Parlo et al., 1994; Kechriotis et al., 1994; Robinson, 1994; Goudreau et al., 1995). One research area is in the use of recurrent neural networks to simulate time series data. In the work of Connor (1992), the author claimed recurrent networks produced performance superior to that of back-propagation when the time series possessed a moving average component, or was state dependent, or had a trend. The author however did not implement a fully connected recurrent network.

In general, the recurrent networks performed well on "smooth" time series. One study suggested that partially recurrent networks performed best on time series that can be modeled by moving average processes (Connor. 1992). In her dissertation. Logar (1992) indicated that fully connected recurrent neural networks are better suited to the prediction of time series data which can be modeled by an auto-regressive process where a positive correlation exists between a point and its predecessor.

The power of neural networks lies in the fact that they can be trained by old data and predict unknown data. In neural networks training, the connection weights are adjusted according to certain criteria, such as mean square errors calculated by the difference between neural network outputs and the desired outputs. Training is critical to the success of every neural network application. Therefore, Neural networks training becomes the center of neural networks research.

The method used to adjust the weights in the process of training the network is called the learning rule. Many learning rules for recurrent neural networks have been proposed. The learning can be either supervised or unsupervised (Vemuri, 1992). An essential ingredient of supervised learning is the availability of an external teacher. which is represented by a set of input-output examples (Haykin, 1994). In the category of supervised learning algorithms, the back-propagation algorithm is the algorithm which is most widely used by researchers in neural network training. The back-propagation algorithm derives its name from the fact that error terms in the algorithm are back-propagated through the network.

The algorithm proposed by Barak A. Pearlmutter in 1990 is an example of back-propagation algorithms. It is a continuous time version of back-propagation through time. Some of the equations in the Pearlmutter algorithm have not been examined in detail by Pearlmutter himself and have not been carefully studied by other researchers. In order to know what the advantages of the Pearlmutter algorithm are and how to improve it in specific cases, fully understanding those equations and their role in the learning process is necessary.

In this research, the equations of the Lagrangian multiplier z in the Pearlmutter algorithm were investigated by using three kinds of data in training a fully connected neural network. The three kinds of data used are the data that the neural network can learn well and the data that the neural network either cannot learn or has a difficult time learning. A sine wave, which has relative simple temporal behavior, was used as the first kind of data since the neural network can learn the time series well. For the data that the neural network either can not learn or has a difficult time learning, we used the electrocardiograph (ECG) data and data obtained form integrating the Mackey-Glass equation. Both of the ECG data and Mackey-Glass data have more complicated temporal behaviors. Next we will discuss the three time series data used in this study.

1.2 Time Series

A time series is a series of numbers resulting from the measurement of an observable quantity sequentially over the time. (Sauer, 1992). Throughout the history of scientific research and industrial applications, time series have laid the foundation for extracting features of an observed system and for predicting its future behavior. The objectives of time series analysis include predicting, modeling and characterization of the system (Gersheufeld, 1993). Predicting means foretelling future events based on previous data series and forecasting system evolution in the short-term. Modeling has the goal to get a precise description of the system features and form a stable model of system behavior in the long-term. Characterization of the system attempts to determine the basic properties of the system without a priori knowledge (Gersheufeld, 1993).

Using neural networks for time series analysis can widen the range of application because the network's nonlinear learning can yield superior prediction of certain types of dynamical systems. In time series prediction we usually use recurrent neural networks because of the incapability of the feed-forward neural networks in this field. Feed forward networks can only model systems with fixed point attractors. Therefore, feed-forward networks cannot model systems whose time dependence is more complicated.

1.3 Sine Wave Data

The sine wave data are generated by sampling a sine function. Those data points are the desired outputs of the neural network. Since sine data has very predictable dynamics, it is easy for neural network to learn as has shown by other researchers (Ding. 1996; Logar, 1995). We used the sine data to investigate the Lagrangian multiplier equation in Pearlmutter algorithm as an example of successful learning.

The domain of sine function used in this research is one cycle. The period of the cycle is 2π (6.28) seconds. We choose 0.8 as the amplitude of the sine wave. Since the hyperbolic tangent function, which ranges from -1 to +1, is used as the activation function in all the neurons, the amplitude must be less than 1. Otherwise, the neural networks can never follow the sine wave precisely. The number of points used for training depends on the step size Δt and is calculated by $2\pi/\Delta t$. The smaller the Δt, the larger is the number of training points.

1.4 Electrocardiogram Data (ECG Data)

Electrocardiogram (ECG) data is "the recording of voltage changes transmitted to the body surface by electrical events in the heart muscle, providing direct evidence of cardiac rhythm and conduction and indirect evidence of certain aspects of myocardial anatomy, blood supply and function" (Dorddner et al. 1994).

The ECG data used in this study came from Gomez-Gil (1997), which is a modified version of file a0310_5a.dat coming from a database produced by the Harvard-

MIT division of Health Science and Technology Biology Biochemical Engineering Center. The original data was recorded using a sampling frequency of 360 Hz. The a03105a.dat has zero mean. One of the data set used in this study was filtered to a passband from 0.5 Hz to 105 Hz, and normalized to values between [-0.3122, 0.75] and it contained 512 points. The other one of the data was filtered to a passband from 0.5 to 45 Hz and down-sampled by a factor of 4. After that, it was normalized to values in the range [-0.2175, 0.75] and it contained 128 points.

The ECG data is used as the kind of data that the neural network either can not learn or has a difficult time learning.

1.5 Mackey-Glass Data

In this thesis, time series data obtained by integrating the Mackey-Glass equation is also used.

The Mackey-Glass differential equation is:

$$\frac{dx}{dt} = \frac{ax(t-\tau)}{1+x^{10}(t-\tau)} - bx(t) \qquad \text{where } a = 0.2, b = 0.1, \tau = 17 .$$

For this set of parameters, x is known as a chaotic trajectory. The Mackey-Glass data is generated by integrating this equation using Euler's method. The Mackey-Glass data is smoother than ECG data which the neural network has a hard time learning. We hoped that we could get a better training result than that of the ECG data. To improve the chances of learning, we increased the number of hidden neurons of the recurrent neural

network in Mackey-Glass data training. However, since Mackey-Glass data still has very complex dynamic behavior, the recurrent neural networks continue to have a difficult time learning the function.

1.6 Neural Networks and Their Training

As mentioned above, three different sets of time series data were used in this research, sine wave data, ECG data and Mackey-Glass data.

The equations that describe the behavior of the neurons are differential equations (the Pearlmutter algorithm and the differential equation are discussed in section 2.1). The method of integration most commonly used is Euler's method. One aspect of the integration is to choose an adequate step size. The main part of this work is to investigate the effect of the step size choices.

In training the network to the sine function, different step sizes are used to compare the different results that received from the training. The fully connected recurrent neural network in sine function learning has one output neuron, two hidden neurons and zero input neurons.

For the ECG data, the number of hidden neurons is 6. The number of output neurons is still one and the number of input neurons is zero.

For training the Mackey-Glass data, we adjust the number of the output neurons to 1 and the total number of neurons to 20. The number of input neurons will still be 0.

1.7 Objective

A large class of problems amendable to neural network solutions is temporally sequential in nature. In order to deal with these problems, Pearlmutter proposed the an algorithm for recurrent neural network training. However, the equations in Pearlmutter algorithm, especially the equations for Lagrangian multiplier, have not been investigated in detail. In this research, a detailed investigation of Lagrangian multiplier equation in the Pearlmutter algorithm is given. The variation of Lagrangian multiplier during the neural network training was closely looked at. The behavior of the solution over time for learned and unlearned examples was studied. We also investigate the first order difference of the multiplier and the second order difference of the multiplier in order to evaluate the choice of the step size. The reason that we investigate the first order difference and the second order difference of the Lagrangian multiplier is that by looking at these values, we can decide if the step size is small enough for the training. In a small interval of a linear section in the sine wave, we expect the Lagrangian multiplier to be linear. The reason is that the neural network is approximate to a linear system when learning this small interval of a linear section. The Lagrangian multiplier, which is the order derivative of total error with respect to neuron outputs, is also linear in this small interval. We further expect the first order difference of the Lagrangian multiplier to be a constant, and the second order difference of the Lagrangian multiplier to be zero. If they are not, it indicates that in this small interval the simulated curve is not linear and thus, the learning is not good. It is known that the smaller the step size, the better the neural network can learn the time series data. Therefore, we need to decrease the step size for better learning.

Through the investigation of Lagrangian multiplier, the first order difference and the second order of difference of the Lagrangian multiplier, we can get the information when the step size is small enough for good learning.

1.8 Organization of the thesis

This thesis is composed of the following chapters.

Chapter I explains some basic concepts about neural networks and recurrent neural networks. The time series data used in this research are also introduced. In addition, the structure of neural networks used in the study and initialization and training of the neural networks are also introduced. Chapter II gives the Pearlmutter algorithm and its derivation. In Chapter III, results of investigating the Lagrangian multiplier equation using sine wave data are given. Chapter IV gives the experimental results of investigating the Lagrangian multiplier equation using ECG data, and Chapter V gives the experimental results of investigating the Lagrangian multiplier equation using Mackey-Glass data. Conclusions and suggestions for future research are discussed in Chapter VI.

CHAPTER II

DESCRIPTION OF PEARLMUTTER ALGORITHM

In this chapter, the Pearlmutter equations and algorithm are given.

2.1 The Pearlmutter Algorithm

Back propagation through time is a popular learning algorithm and it is a very powerful tool in training recurrent neural network to learn dynamic data. Its application includes pattern recognition, dynamic modeling, sensitivity analysis, and the control of systems over time, among others. It can be applied to neural networks, to economics models and to almost any system built up from elementary subsystems or calculations (Werbos, 1990).

Pearlmutter defined a neuron network as a connected architecture in which differential equations are used to describe the evolution of the neuronal activation. The networks discussed here are fully connected networks. In a fully connected architecture all nodes are identical and only through designation are specific nodes marked as input, hidden and output nodes. Every node is connected to every other node and the output from each node is fed back as an input to itself and to all other nodes.

Ding (1996) used three methods to derive the Pearlmutter algorithm: the Werbos method, the Pearlmutter method and the Variational method. The first two methods both unfold the continuous-time NN model into discrete NN model and employ the conception

of ordered derivatives (Werbos, 1988). In this study, we used the Pearlmutter algorithm as published by Pearlmutter.

The Pearlmutter algorithm is based on the Pearlmutter model, which is governed by the following equations:

$$x_i(t) = \sum_j w_{ji} y_j(t) \quad . \qquad (2.1)$$

$$\frac{dy_i(t)}{dt} = -y_i(t) + \sigma(x_i(t)) + I_i(t) \quad . \qquad (2.2)$$

where i, j = 1, 2, 3…,n, and n is the total number of neurons, w_{ij} is the weight of the connection from neuron j to neuron i. Y_i is the state of neuron i. X_i is internal input of the network while I_i is external input of the neural network. The nonlinear function is usually a squashing function. In this thesis, we use the squashing function $\sigma(x) = \tanh(x)$.

The Pearlmutter Algorithm is one of the gradient descent methods. It uses the continuous time version of back-propagation through time. In the algorithm, learning is performed as a gradient descent in the weights of a continuous recurrent network to minimize an error function E of the temporal trajectory of the states. It can be thought of as an extension of recurrent back-propagation to dynamic sequences (Hassoun, 1995).

In this method, the trajectory is to be approximated as a continuous function. The error is calculated by integrating the square of the difference between the desired trajectory and the network's trajectory. Making this integral small will cause the network trajectory to be close to the desired one (Corwin, 1995). The error calculation of the sine function training is shown in Figure 2.1.

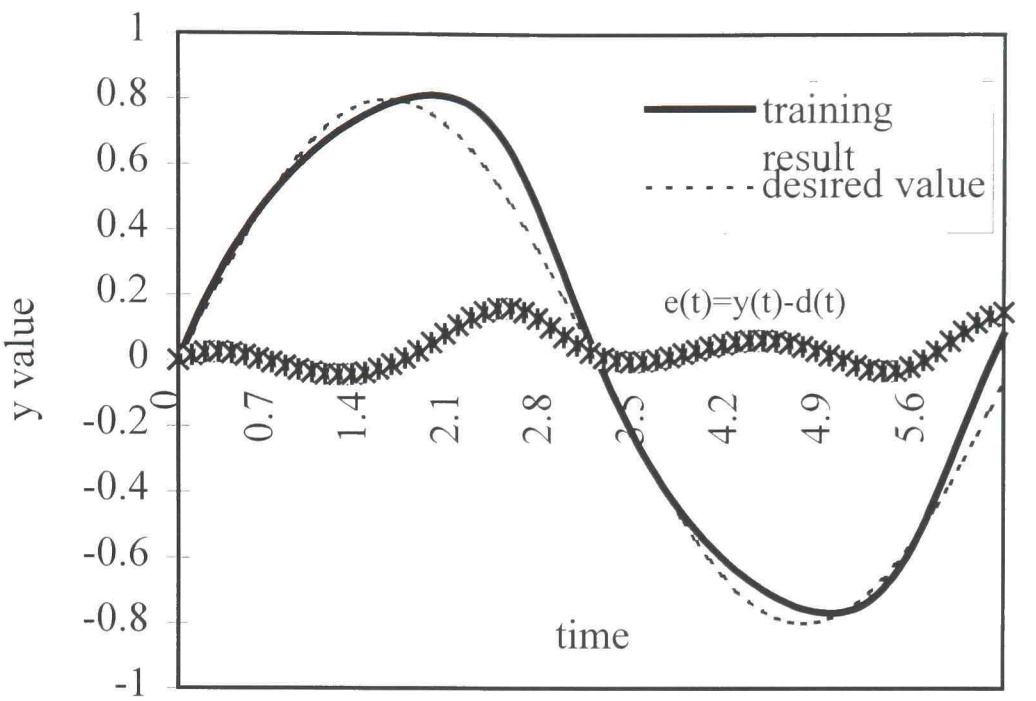

Figure 2.1 The error calculation

According to Pearlmutter algorithm, the Lagrangian multiplier is given as follows:

$$z_i(t) = \Delta t * e_i(t) + (1-\Delta t)\,\tilde{z}_i(t+\Delta t) + \sum_j w_{ij}\sigma'(\tilde{x}_j(t))\Delta t \cdot \tilde{z}_j(t+\Delta t)\,. \qquad (2.3)$$

From this equation, we can get the differential equation of z (throughout the research, z stands for Lagrangian multiplier) if we use the gradient equation:

$$\frac{dy_i}{dt}(t) \cong \frac{y_i(t+\Delta t) - y_i(t)}{\Delta t}\,. \qquad (2.4)$$

The differential equation of z is:

$$\frac{dz_i}{dt} = z_i - e_i - \sum_j w_{ij}\sigma'(\tilde{x}_j)\cdot z_j\,. \qquad (2.5)$$

The Euler's method was used to integrate these equations as it is the simplest method in numerical integration. Given the previous data point (t_n, y_n) and assuming that

the slope (dy/dt) remains constant for a small distance Δt, the value of $y(t_n+\Delta t)$ can be estimated as

$$y_{n+1} = y_n + \frac{dy}{dt} \cdot \Delta t \ . \qquad (2.6)$$

In the z equations, the Euler's method is reduced to:

$$z_i(t+\Delta t) = z_i(t) + \frac{dz_i}{dt} \cdot \Delta t \ . \qquad (2.7)$$

where Δt is the step size.

The equation for the weights update is:

$$\frac{\partial E}{\partial W_{ij}} = \int_{t_0}^{t_1} y_i \sigma'(x_j) z_j dt \ . \qquad (2.8)$$

Since Euler's method has a local truncation error on the order of Δt^2, the step size Δt has significant effect on the precision of Euler's method. Theoretically, the smaller the step size, the more precise the Euler's method is. However, with decrease of step size, data points used in training increase, the training becomes more computationally intensive, and the propagation error increases. Therefore, the step size should comprise of the truncate error and the propagation error.

CHAPTER III

INVESTIGATING STEP SIZES AND LAGRANGRAN MULTIPLIER USING SINE WAVE DATA

In this chapter, we present results of our investigation of the effect of using different step sizes in the Pearlmutter algorithm on neural network training when using the sine wave data set. We will investigate the Lagrangian multiplier in the Pearlmutter algorithm by looking at how it changes along the training procedure. Analysis will be given throughout these chapters when problems are encountered, and comparison among the data sets is given.

The main task in this chapter is to adjust the step size Δt to make the training result better. Since Euler's method is used in the integration, the step size has a great effect on the precision of Pearlmutter algorithm, and consequently on the training results.

3.1 Neural Network Architecture in Sine Function Learning

In training the sine function, the architecture of the neural network is as follows:

The number of hidden neurons is 2; the number of output neurons is 1; the number of input neurons is 0.

The model used in training the sine function is shown in Figure 3.1.

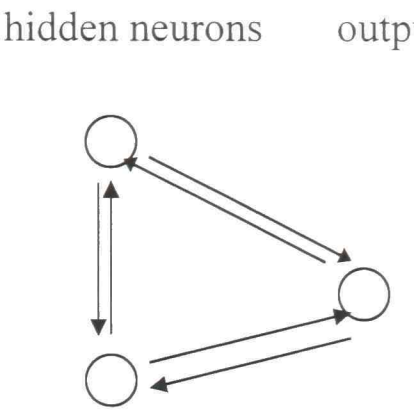

Figure 3.1. Fully connected recurrent neuron network model used in sine data learning

During the training of the network to the sine function, the training is carried out in two ways. One is to train all the data points in one cycle at one time. The weights are initialized by a random number generator in C++. The other one is using a sequential strategy (Ding, 1996) to train the neural network. The sequential strategy is based on the fact that it is difficult for the neural network to learn a large number of sample points at one time. Thus, at first, a small portion of sample points are used to train the network. The weights are initialized with small random numbers. After the training converges, the weights are stored for further training. Then a larger portion of sample points are used to train the network using the weights obtained from last training. This step-by-step procedure continues until all the sample points are used in training. This procedure is proved to be very effective in training. Hence, it is used here.

3.2 Sine Data Description

In training the neural network, different step sizes are used in order to decide the effect of step size on training results. The domain of sine function used in this research is one cycle. The period of the cycle is 2π (6.28) seconds and the amplitude can be varied. In this study we use 0.8 as the amplitude. The number of points used for training is determined by the step size Δt, calculated by $2\pi/\Delta t$. For a smaller Δt, larger number of training points is used. Data points are generated from a cycle of a perfect sine wave before the training starts and are used as the desired output of the neural network. The neural network in this research has one output neuron, two hidden neurons and zero input neuron.

3.3 Case Study of Sine Wave Learning

In this section, z value and the first order difference of z ($z^{(1)}$) as well as the second order difference of z ($z^{(2)}$) will be investigated using different step size Δt . According to the theory, in a small linear interval of sine wave, the $z^{(1)}$ value should approach to a constant and the $z^{(2)}$ value should approach to zero. If the result is not as what we expected, it means that the step size is too big and that a smaller step size is needed. By investigating the $z^{(1)}$ and $z^{(2)}$ value, we can find an appropriate step size and get the desired results. The theory can be explained in Table 3.1 as follows. In this table, we used a linear function, y=2*x. We calculated the first-order difference of y, which is y_1 in this case and the second-order difference of y, which is y_2 in this case. The results show that y_1 is a constant 0.2 and y_2 is 0.

Table 3.1 The illustration of a linear function y=2*x, and its first-order difference and the second-order difference

x	y	y_1	y_2
0	0		
0.1	0.2	0.2	
0.2	0.4	0.2	0
0.3	0.6	0.2	0
0.4	0.8	0.2	0
0.5	1	0.2	0
0.6	1.2	0.2	0
0.7	1.4	0.2	0

In this study, we will compare the training results of the $z^{(1)}$ and $z^{(2)}$ value using different step sizes and find out the effect of step size on neural network learning. We will also look closely at the trends of the change of z in the training procedure. The maximum number of iteration 5000 is used as a stopping criteria here.

From the error of each training, we can see that recurrent neural network with two hidden nodes proposed by Pearlmutter can predict the sine wave very well and the explanation of this has been given by Ding (1996). So in this study we do not focus on whether the recurrent neural network with 2 nodes can train sine wave, rather we focus on the z equation in the Pearlmutter algorithm to see how it behaves. The behavior of z , $z^{(1)}$ and $z^{(2)}$ during the training is our main concern in this research.

3.3.1 Case 1: step size $\Delta t = 0.1$

For a the step size of 0.1, the number of points we need to generate for the training is $2\pi/0.1$. These points are used in the training as the desired outputs of the neural network. The predictions of the neural network after training are shown in Figure 3.2. The sum of square errors using the step size of 0.1 is 0.69, equivalent to an error of 0.0133 on average per point, which is small considering the number of data points we used.

The z values and the first order difference of z ($z^{(1)}$) and the second order difference of z ($z^{(2)}$) are shown in the following figures from Figure 3.3.1 to Figure 3.5.3. In order to compare the results among all the neurons, each neuron is represented by one curve in every figure. We can see from the figures that different neurons behave differently during the training procedure.

Also in the sine function training, data corresponding to an iteration number are drawn in the same figure in order to keep track of the changing tendency of the z , $z^{(1)}$ and $z^{(2)}$ values during the training procedure.

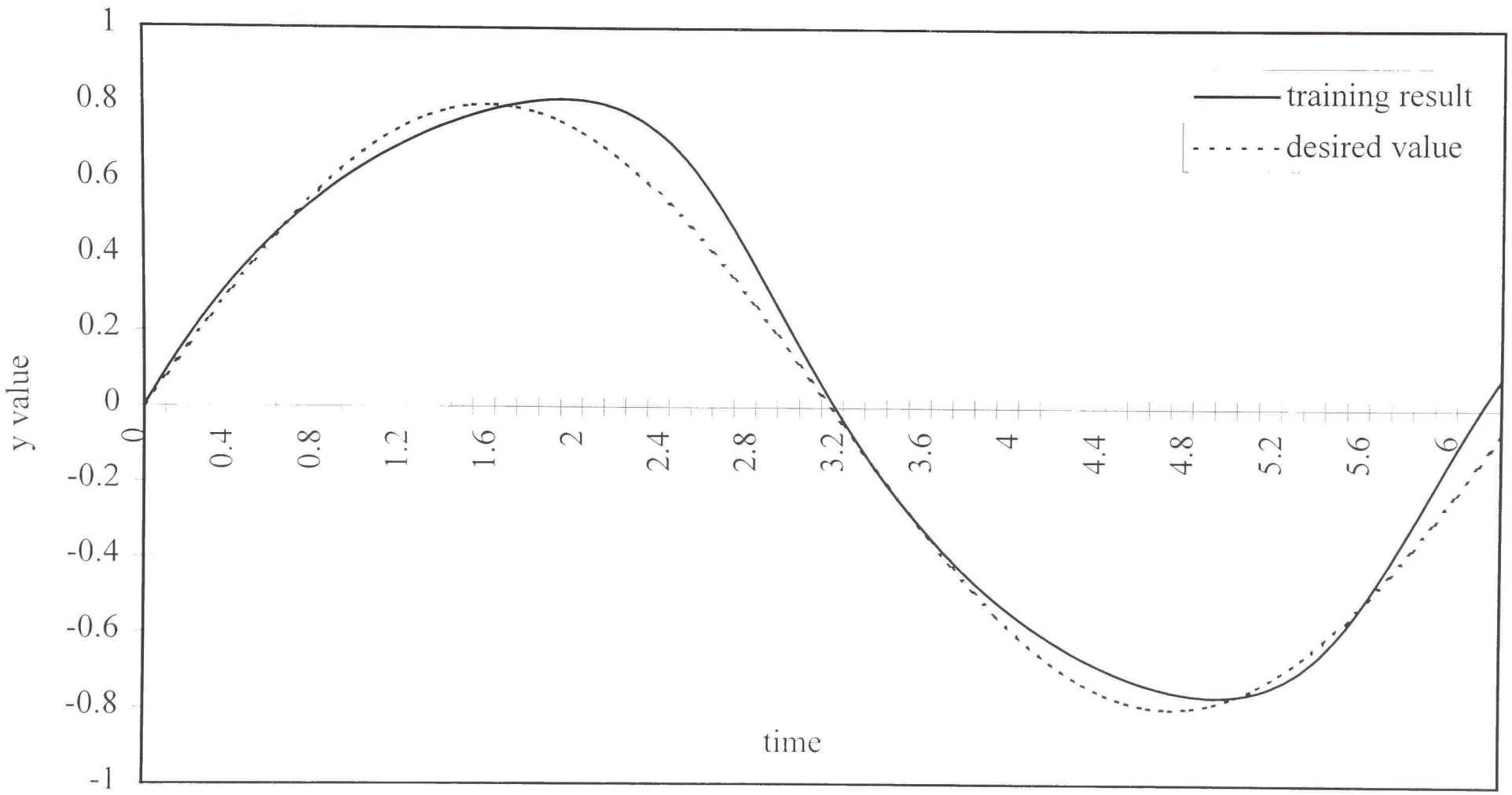

Figure 3.2. Training result of sine wave using step size = 0.1, epoch number =5000

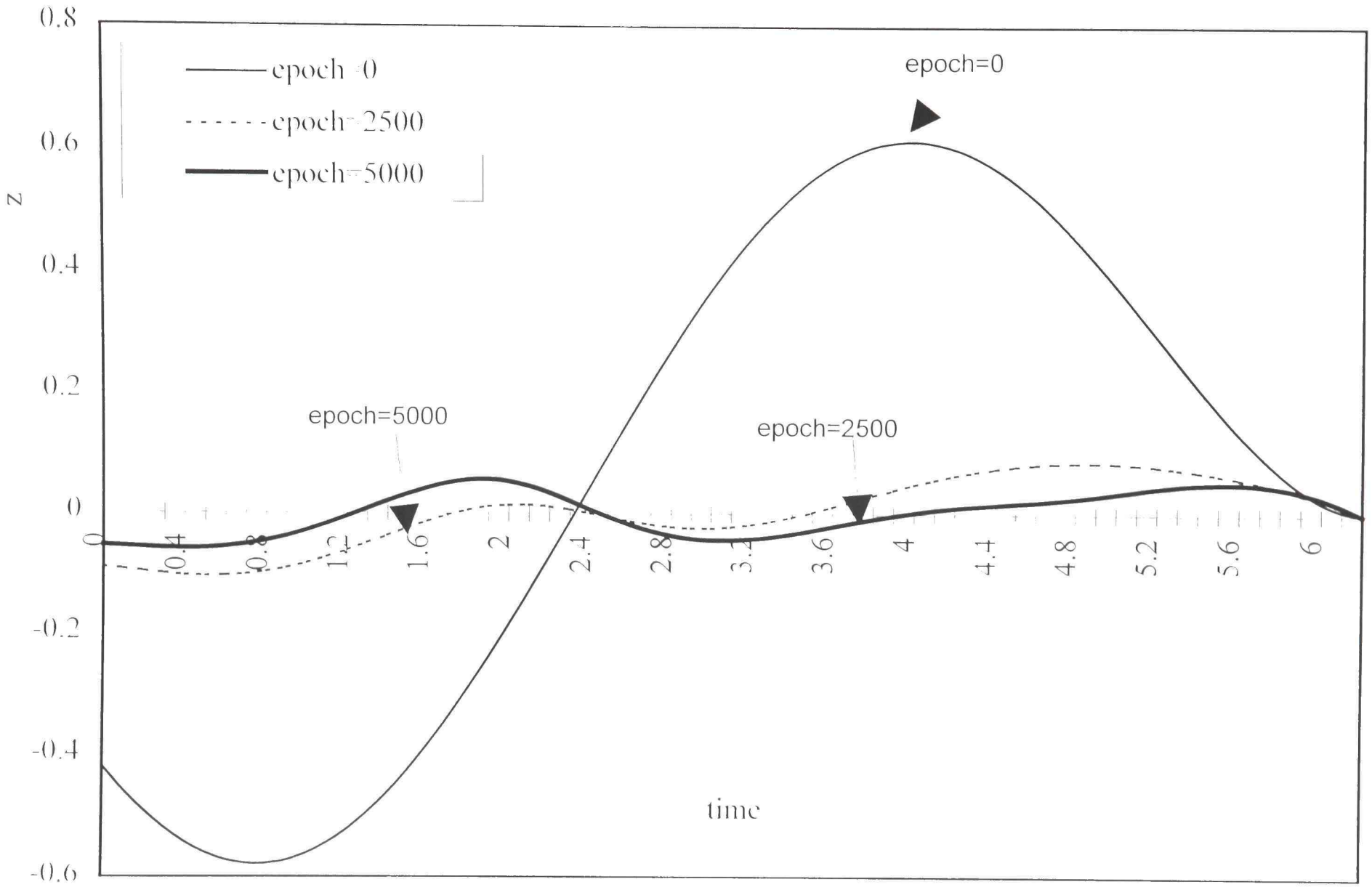

Figure 3.3. z value of the output neuron with step size 0.1

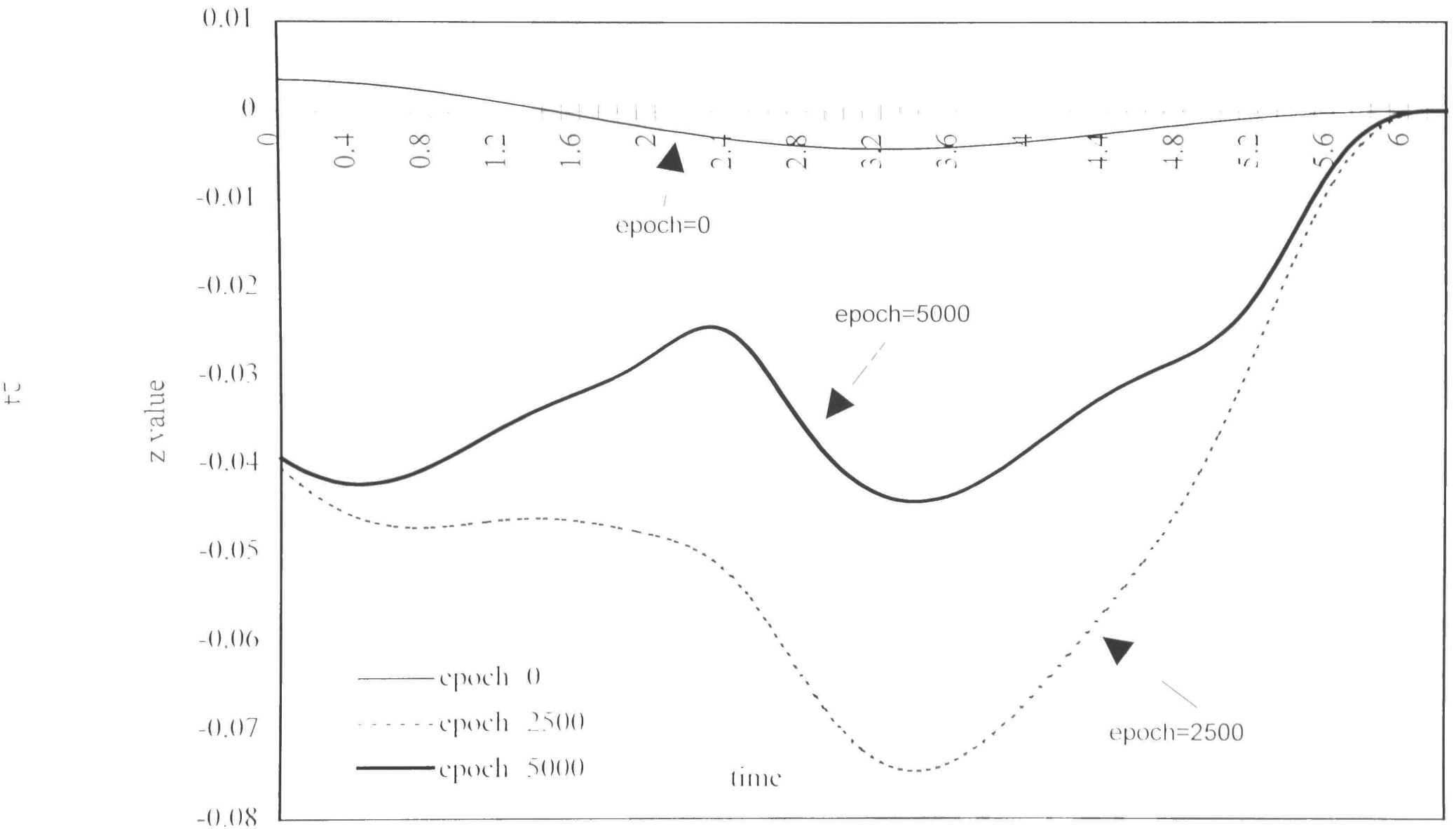

Figure 3.4. z value of hidden neuron 1 with step size= 0.1

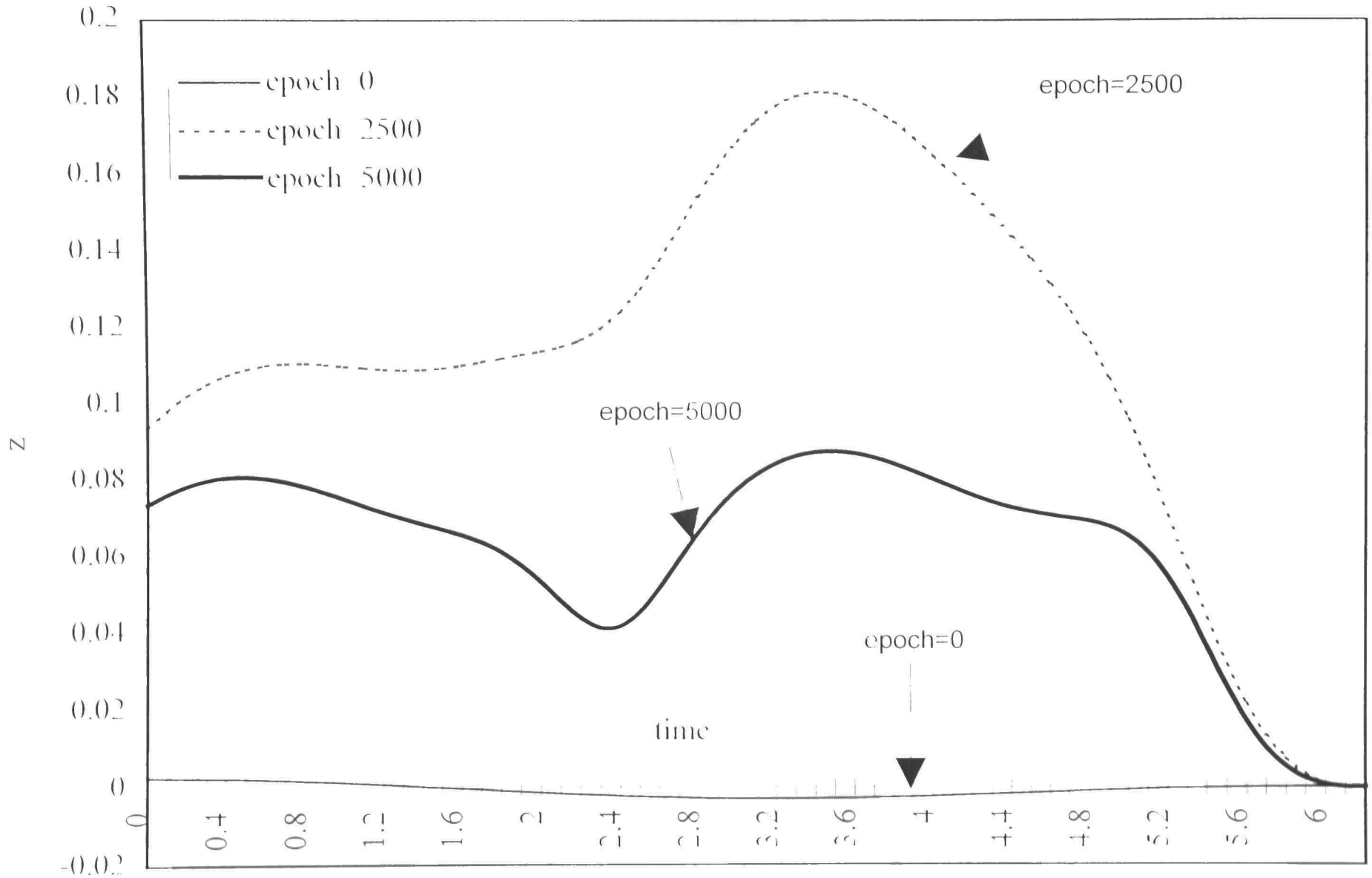

Figure 3.5. z value of hidden neuron 2 with step size 0.1

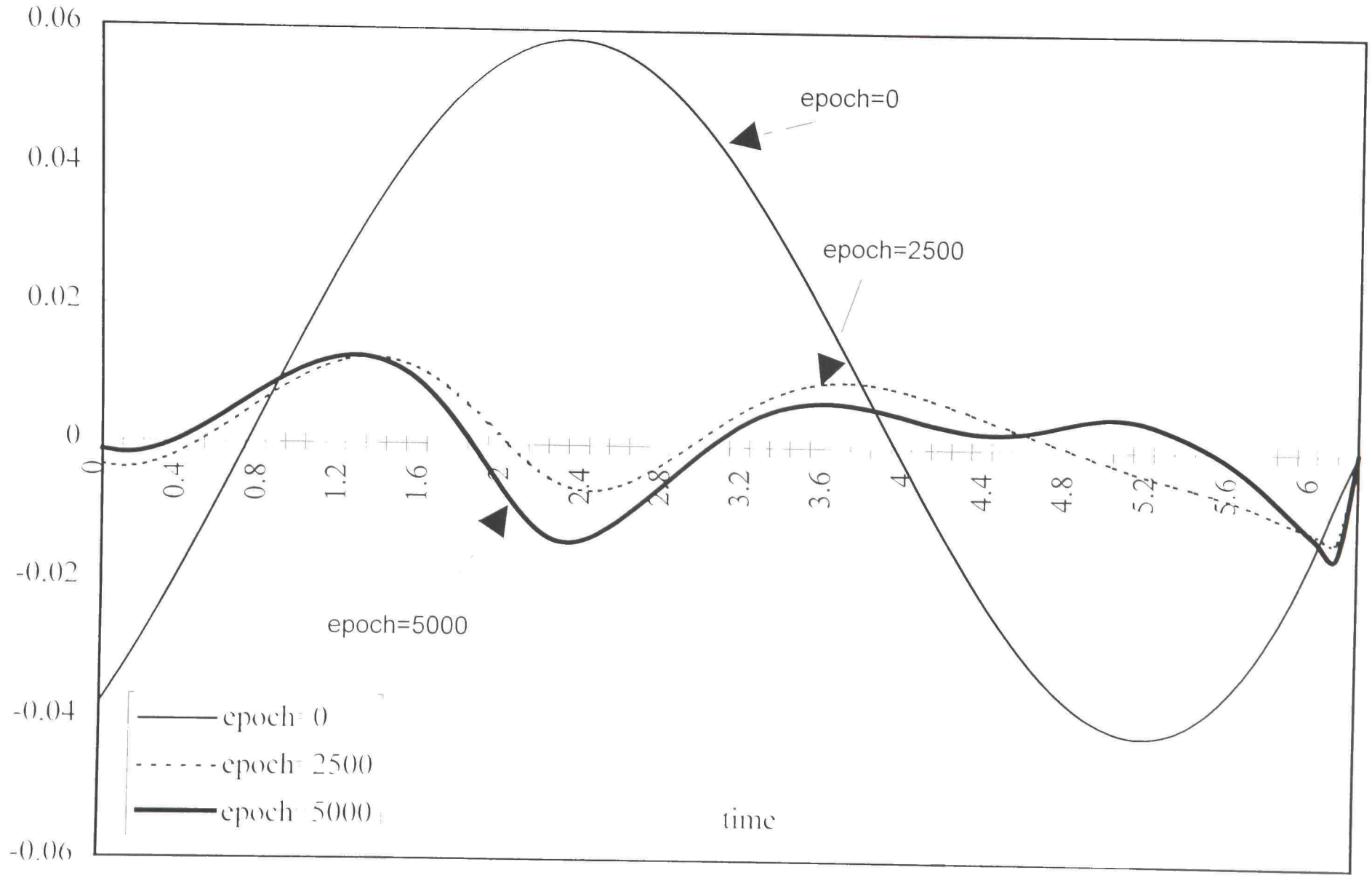

Figure 3.6. z1 value of the output neuron with step size 0.1

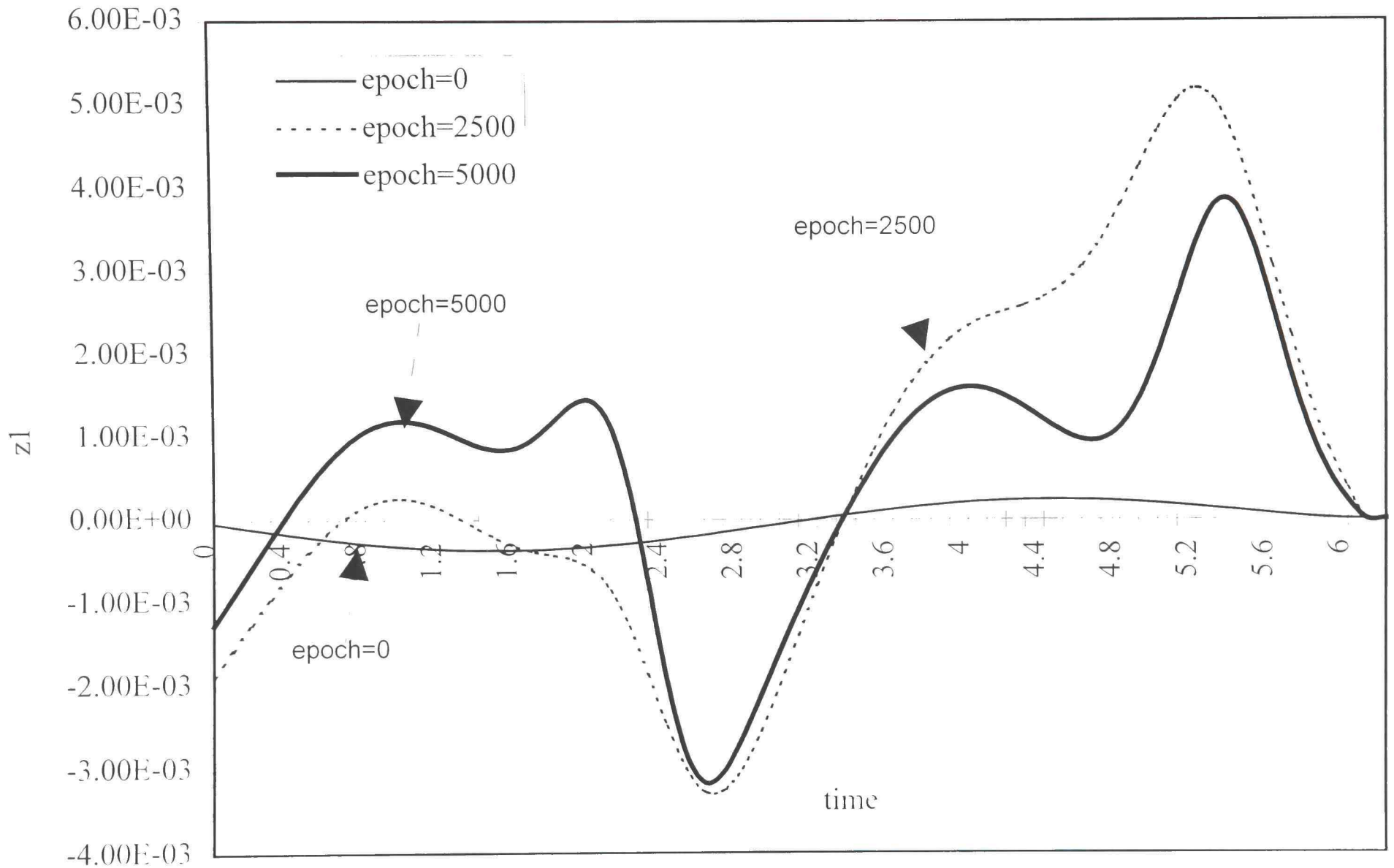

Figure 3.7. z1 value of hidden neuron 1 with step size =0.1

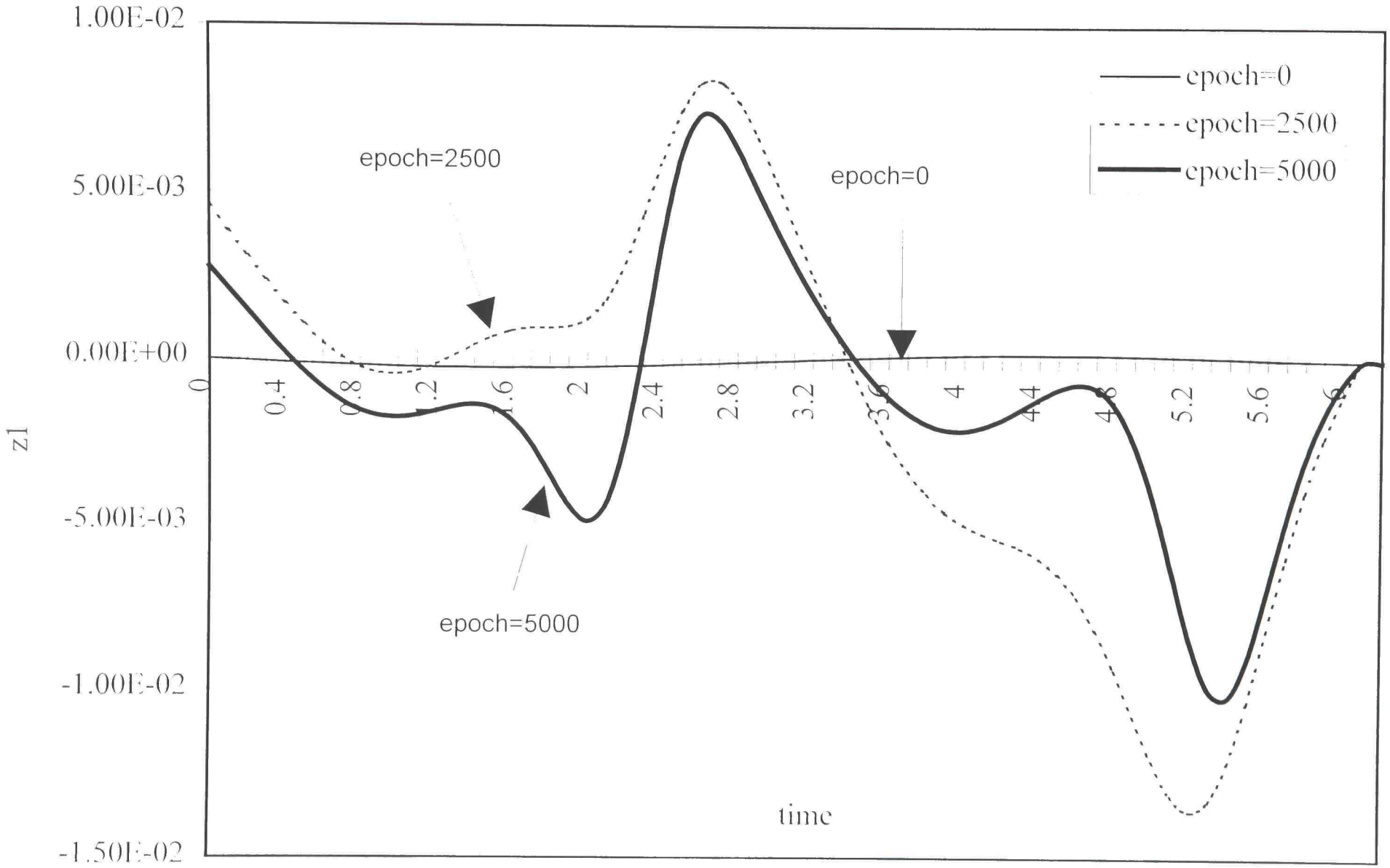

Figure 3.8. z1 value of the hidden neuron 2 with step size=0.1

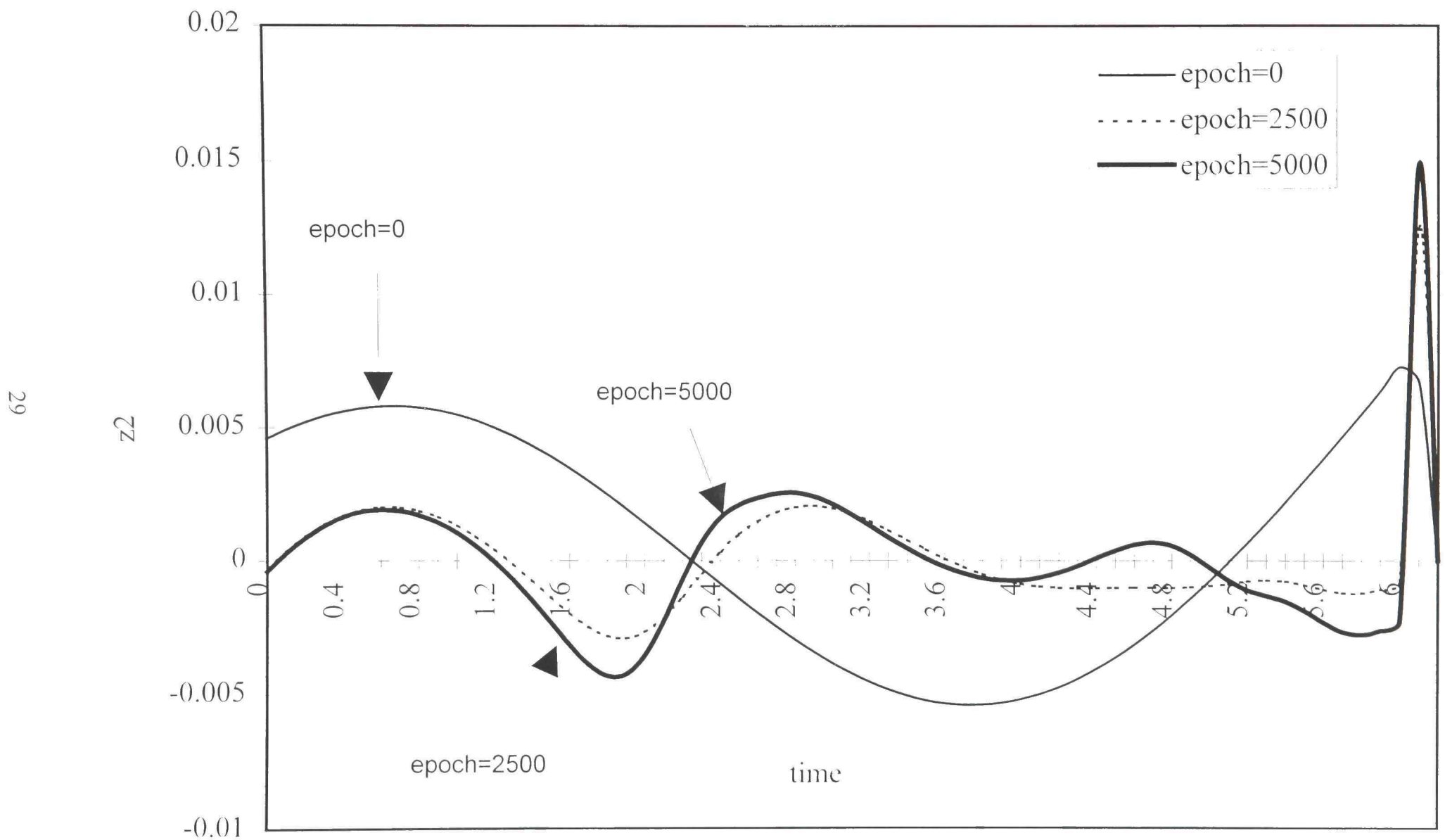

Figure 3.9. z2 value of the output neuron with step size=0.1

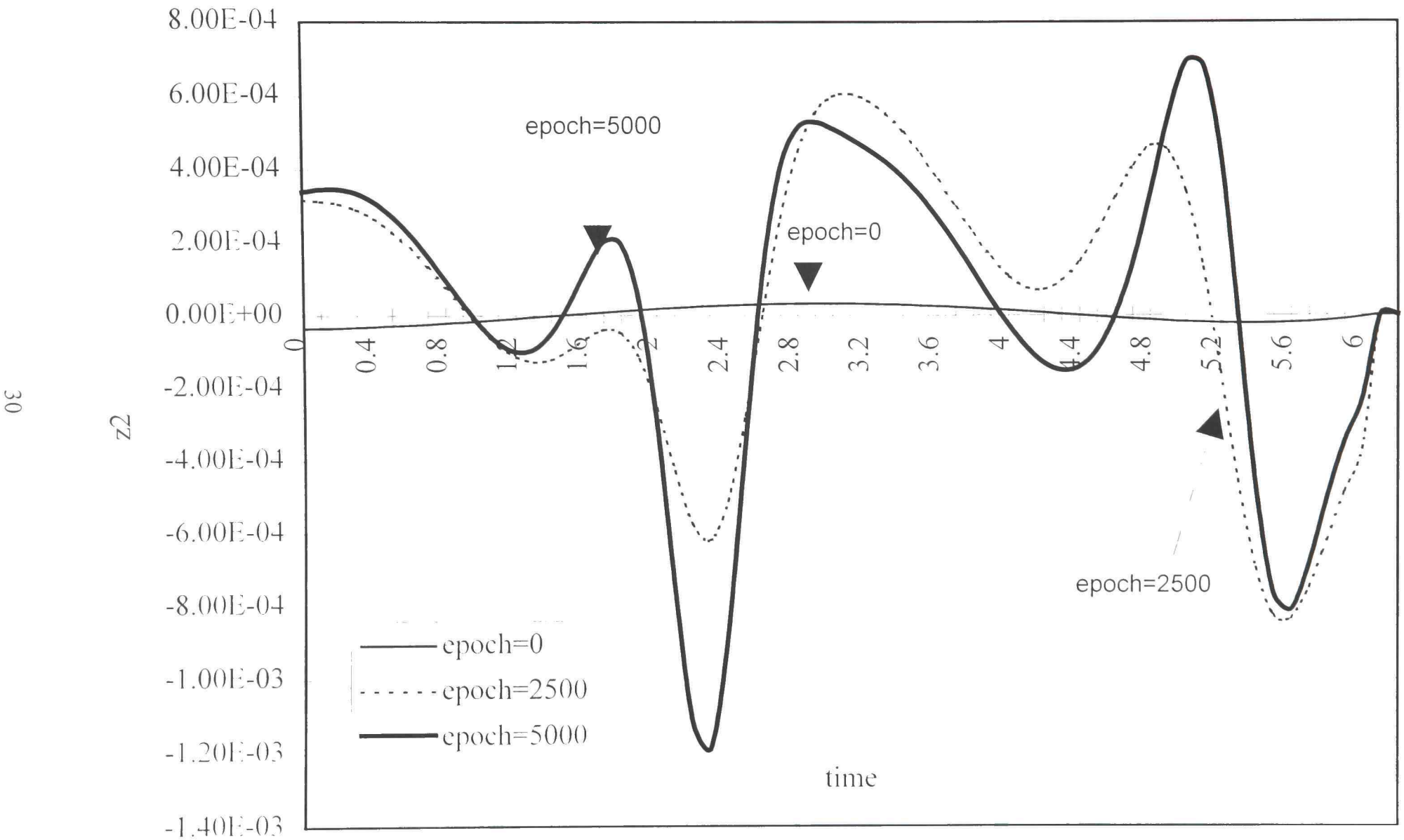

Figure 3.10. z2 value of hidden neuron 1 with step size=0.1

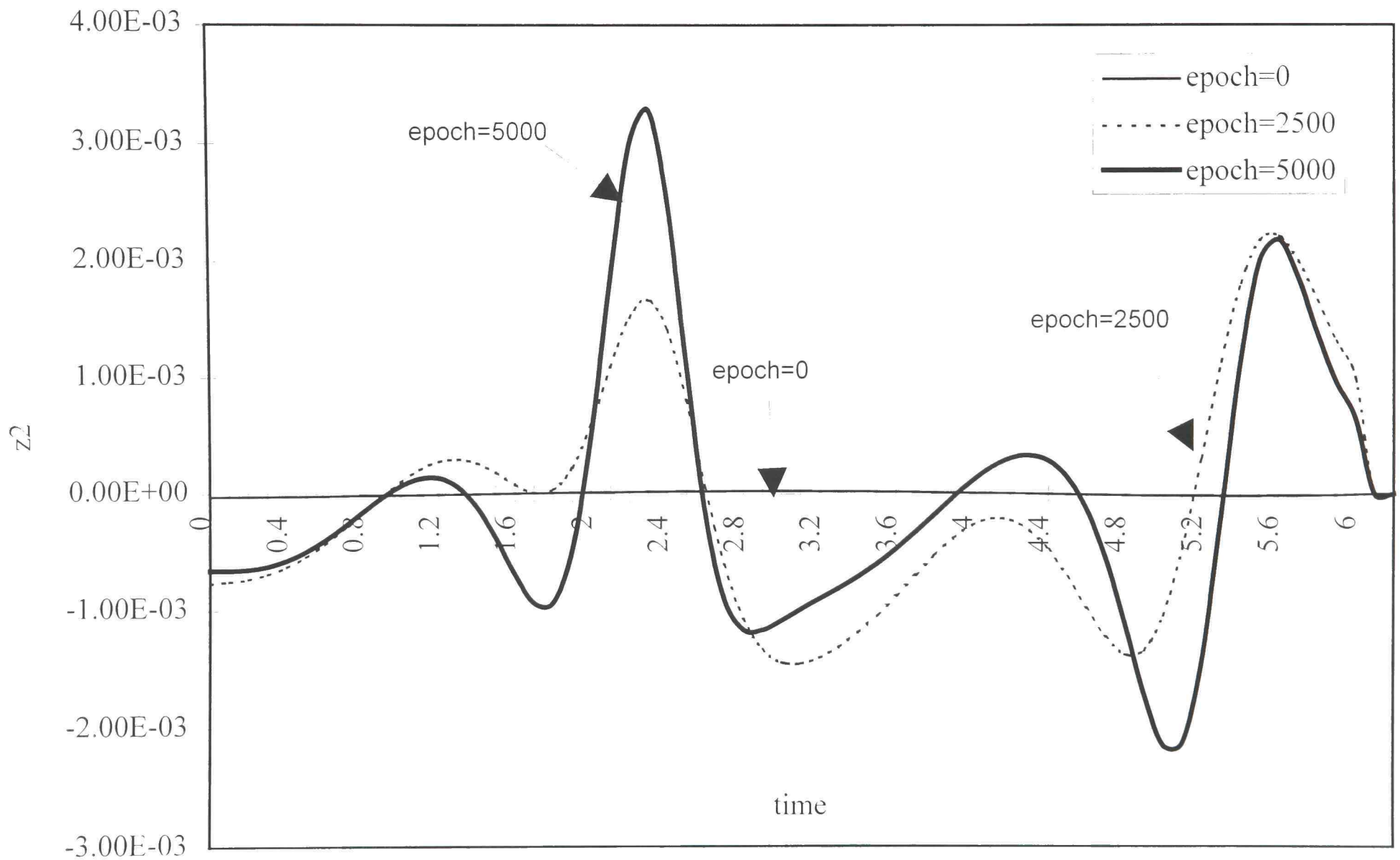

Figure 3.11. z2 value of hidden neuron 2 with step size= 0.1

From the above figures, we can see that the curve predicted by recurrent neural network cannot fit the sine wave well. The main errors occur near the maximum and the minimum of the sine wave. The reason is that near the maximum and the minimum of the sine wave, the curve is most nonlinear. It is hard for the network to follow the nonlinear parts of the sine wave, so the error increases greatly in those parts. When we analyze the figures, we take a small interval in the sine wave curve, for instance, the interval [3, 3.6]. where the sine wave is close to linear. This can be seen on Figure 3.2. From the theory, the first-order difference of z should approach to a constant in this small time interval. However, from Figure 3.6, which shows the curves of the first-order of z value of the output neuron, we can see that in this interval the first-order of z is not close to a constant. Actually, in that interval the first-order difference of z changes a lot. The same result were obtained for the hidden neurons. Also from the theory, in this interval, the second-order difference of z should approach 0. From Figure 3.8, which shows the curves of the second-order difference of z of the output neuron, we can see that it is not what we expected. The z value in the interval [3, 3.6] does not approach 0, rather the maximum value of z_2 in this interval is about 0.005.

Also from the above figures, we can see the tendency of the z values, the first-order of z and the second-order of z from the curves of different iteration numbers. Generally speaking, when iteration number gets larger, the amplitudes of z value, the first-order difference and the second-order difference of z of the output neuron get smaller.

We can see from Figure 3.3 that at the iteration number of 0, the z values of the output neuron are very large and as the iteration number increases, the z values get smaller. At the end of the training, which is iteration of 5000, z values approach to 0. The reason is that at the beginning of training, the network is totally untrained and magnitude of the error is very big. Consequently, every change of total error caused by the change of y is amplified by the square formation of the error. Thus, the z value is very big at the beginning of training. The z values approach to zero at the end of the training. The explanation can be given by the z equations of the Pearlmutter algorithm. The differential equation of z is:

$$\frac{dz_i}{dt} = z_i - \sum_j w_{ij}\sigma'(\tilde{x}_j)\cdot z_j - e_i .$$

For a good training, e_i approaches zero. So the equation becomes:

$$\frac{dz_i}{dt} = z_i - \sum_j w_{ij}\sigma'(\tilde{x}_j)\cdot z_j .$$

Since the z value is initialized to zero, that is $z_i(t_f)=0$, where f stands for final state since the propagation is backward. So we can get that $z_i(t_f-\Delta t)=0$, further we can get that z_i is zero everywhere. Same explanation holds for the following cases of different step sizes.

However, the hidden neurons have different behaviors. At the beginning of training, z values of the hidden neurons are close to 0. As the iteration number increases, the magnitude of z first increases and then decreases. We can see this trend from Figure 3.3 to Figure 3.5. The absolute values of z for hidden neurons are the largest at iteration number of 2500 among three iteration numbers. It can be explained as follows. The

variation of y on hidden neurons has no direct contribution to the error and the contribution comes indirectly from the effect of variation of y on the other neurons. Since the weights are initialized randomly, most of this indirect effect is cancelled out in the network. With the training proceeding, the weights in the network become in certain order and the indirect effect becomes bigger. However, to the end of the training, the total error become small and effects decrease due to the square formation of the error. Therefore, there is maximum of z at some point along the training procedure and are comparably small at the both ends of the training. Same explanation holds for the following cases of different step sizes.

Notice that scales we used for the output neuron and the hidden neurons are different. The scale for the output neuron is bigger than those for the hidden neurons. This can be explained by the fact that variation of y of output neuron affects the error directly and thus has larger total effect than that of a hidden neuron. Same explanation holds for the following cases with different step sizes.

For the step size of 0.1, we also notice in Figure 3.9, which shows the curves of the second order difference of z of the output neuron, when time approaches to 6.28, the z_2 changed a lot and it is not stable even at the epoch number of 5000. While this is not the case in Figure 3.10 and Figure 3.11 which show the curves of the z_2 values of the hidden neurons. The reason remains unknown. The behavior of the output neuron at the end point of the training data needs further study. Anyway we can get one thing from it that the behavior of output neuron and the behavior of hidden neuron are quite different.

The reason that the results are different from theoretical analysis is that the step size used in the training is too large. In the next case, we will use a smaller step size.

3.3.2 Case 2: step size $\Delta t = 0.05$

Next the step size is decreased to 0.05.

For a the step size of 0.05, the number of points we need to generate for the training is equal to $2\pi/0.05$. The predictions of the neural network after training are shown in Figure 3.6. The sum of square errors is 0.84, which is equivalent to an error of 0.00728 on average per point. This average error per point is smaller than that of the average error of the step size of 0.1, which is 0.0161.

The z values, the first order difference of z and the second order difference of z are shown in the following figures from Figure 3.13 to Figure 3.21. Again we can see from the figures that different neurons behave differently during the training procedure.

Also in the sine function training, data corresponding to an iteration number are drawn in the same figure in order to keep track of the tendency of the z , $z^{(1)}$ and $z^{(2)}$ values during the training procedure.

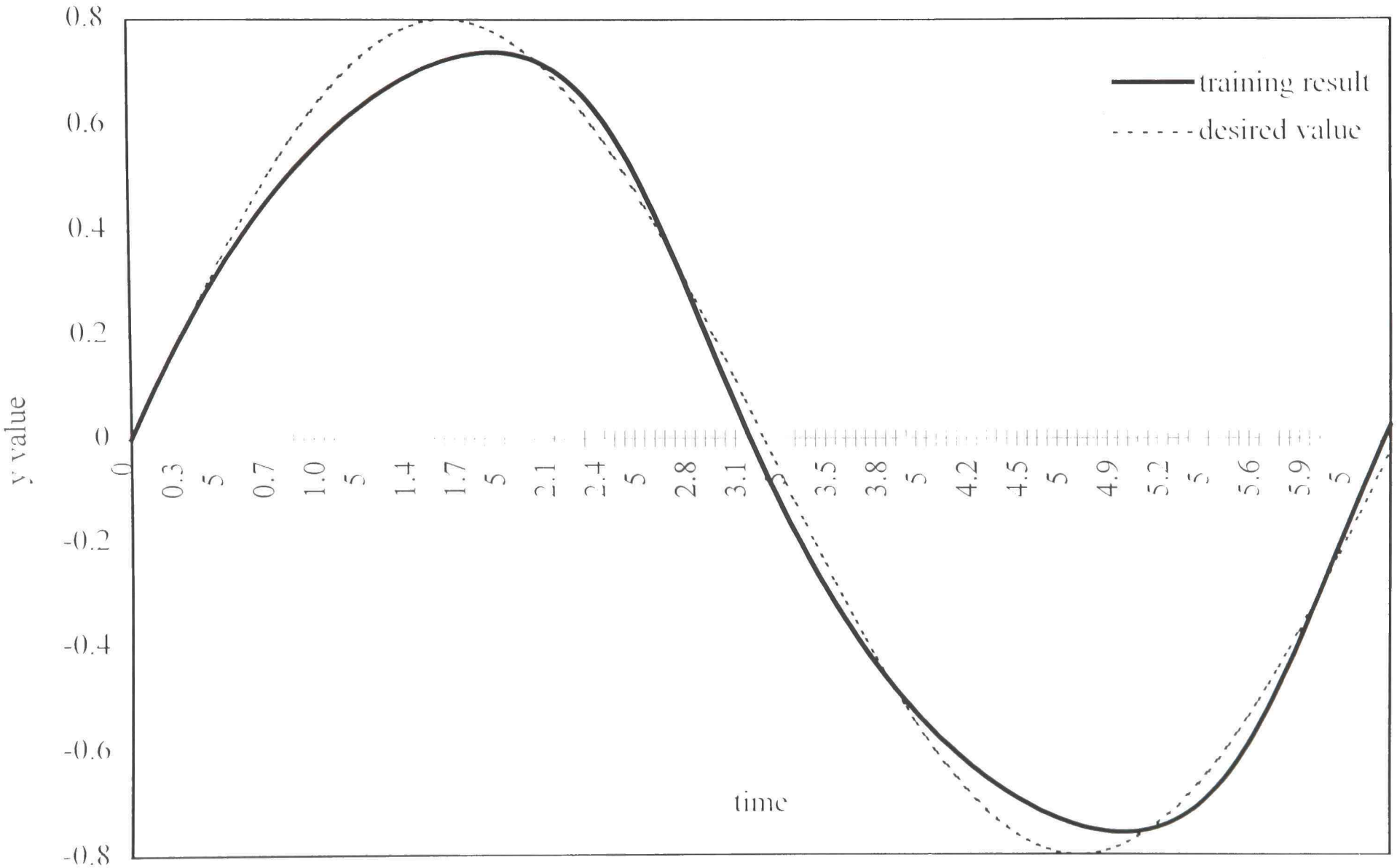

Figure 3.12. Training result of sine wave with step size 0.05 and iteration number 5000

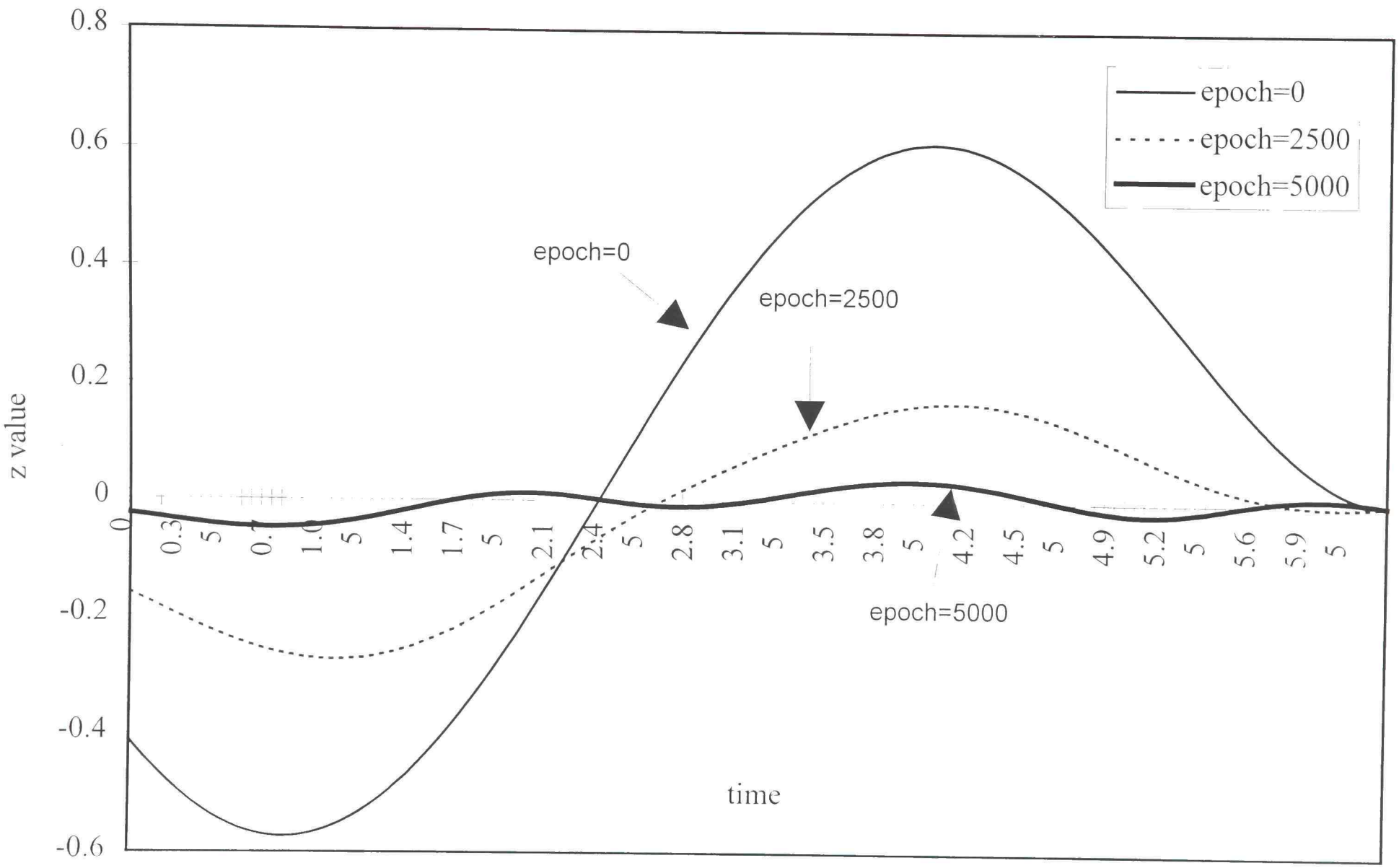

Figure 3.13. z value of the output neuron with step size=0.05

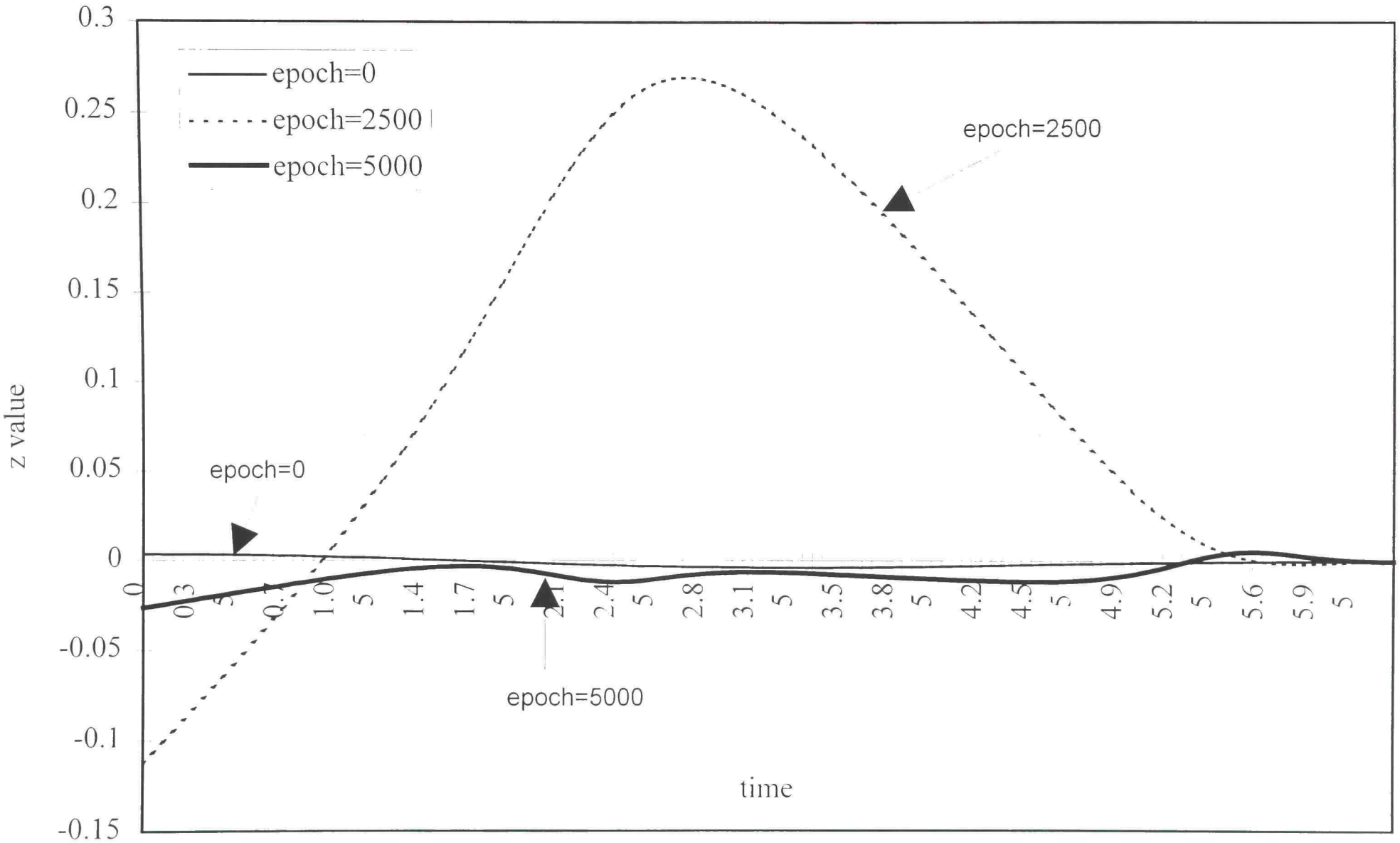

Figure 3.14. z value of hidden neuron 1 with step size=0.05

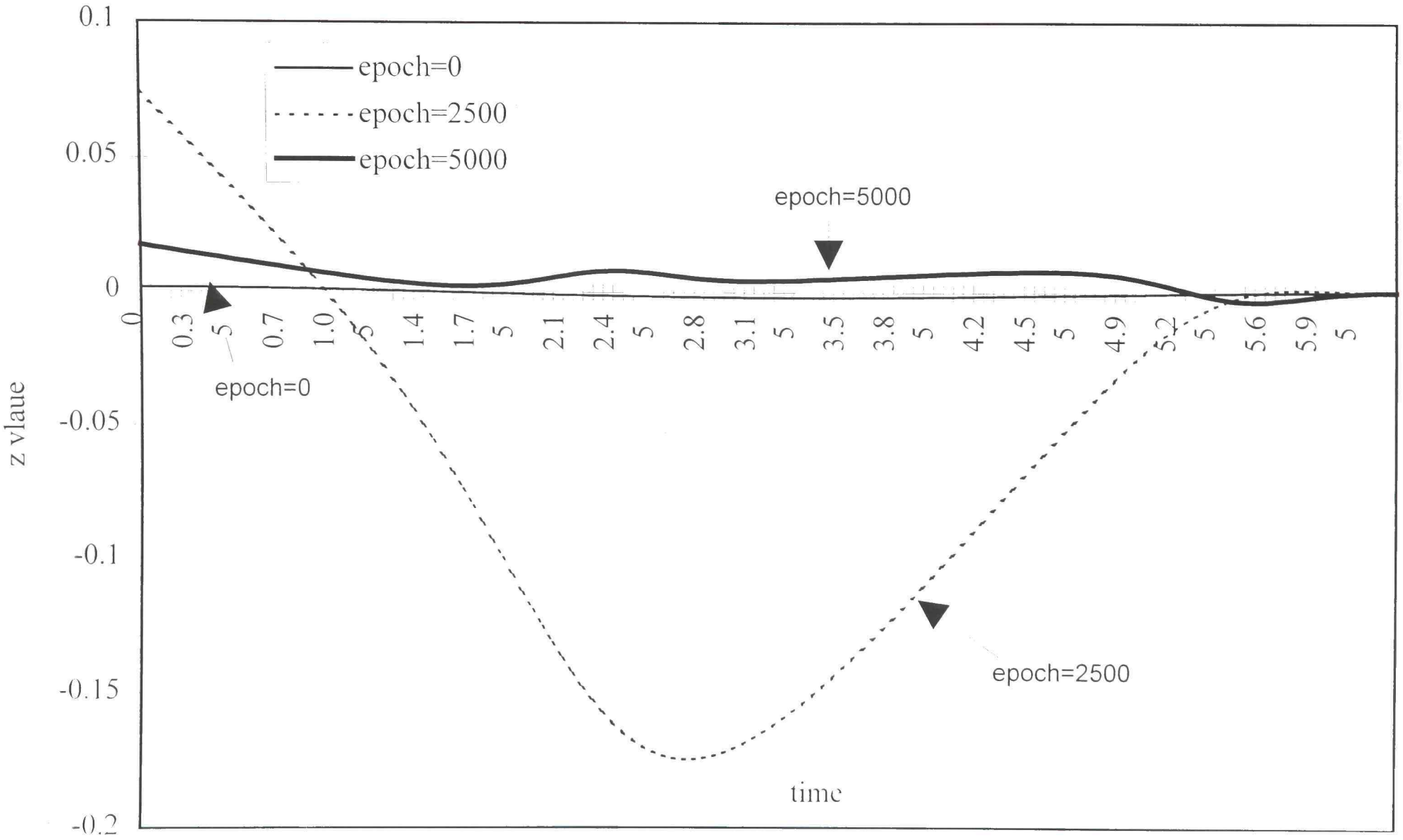

Figure 3.15. z value of hidden neuron 2 with step size=0.05

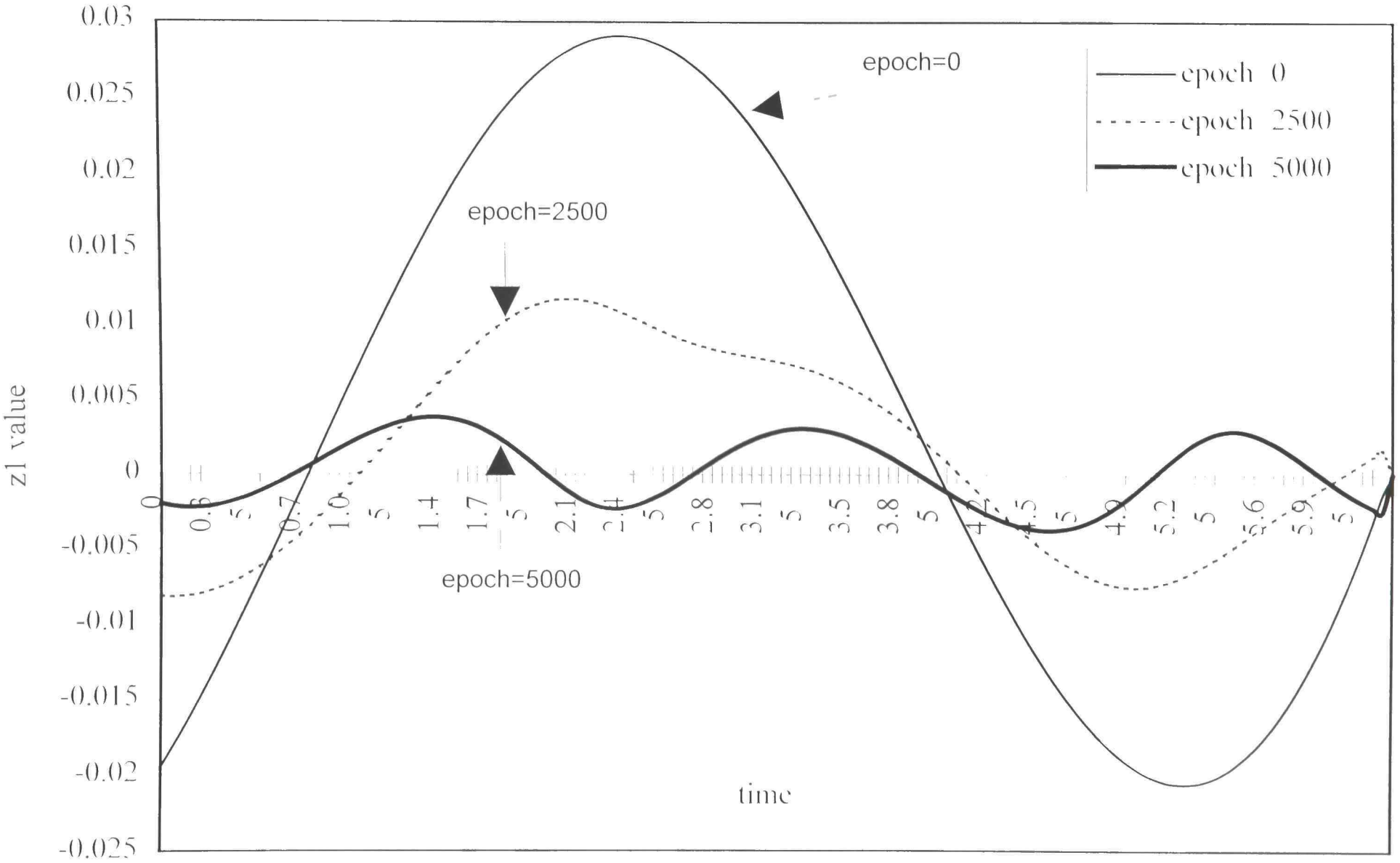

Figure 3.16. z1 value of the output neuron with step size 0.05

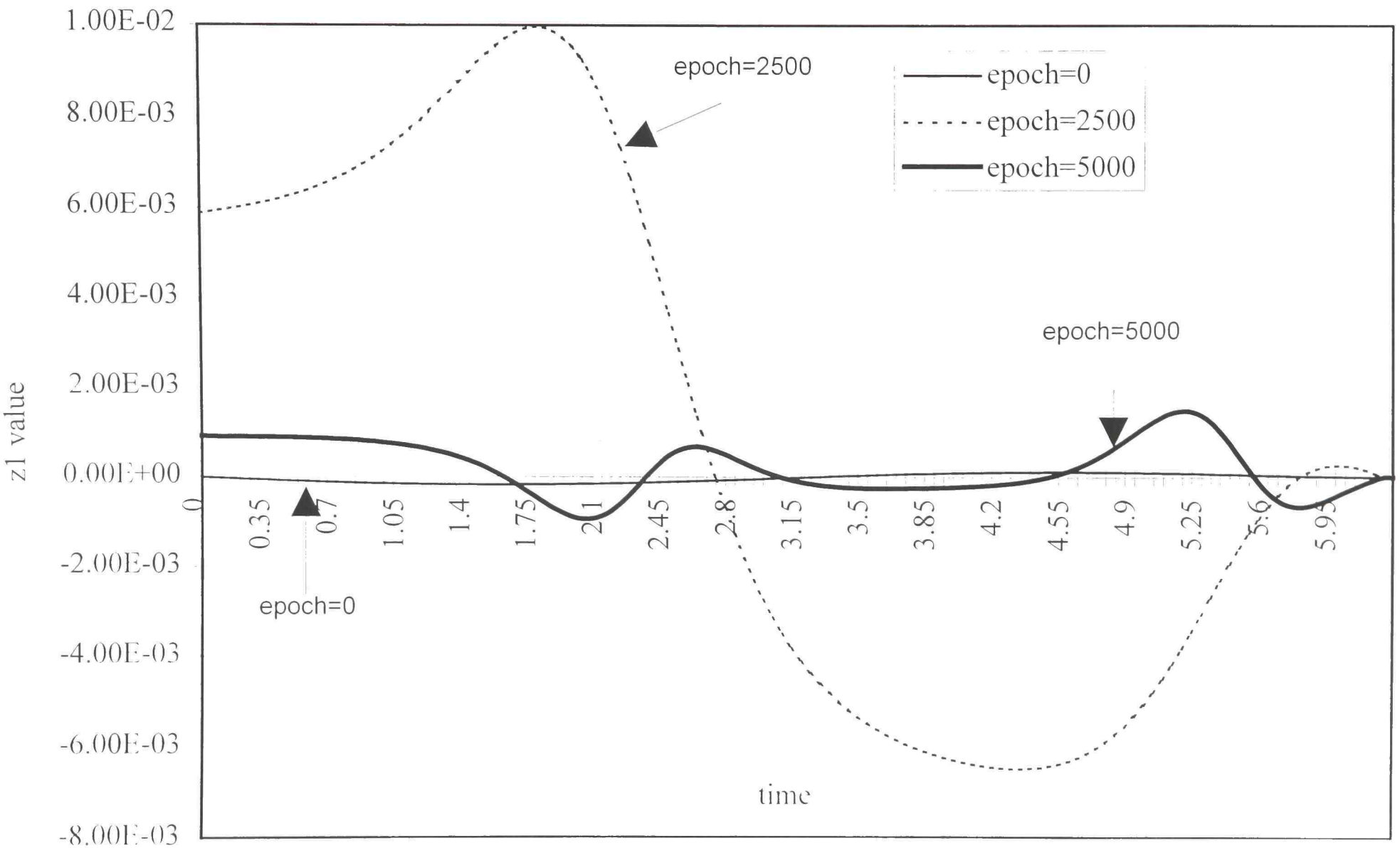

Figure 3.17. z1 value of the hidden neuron 1 with step size=0.05

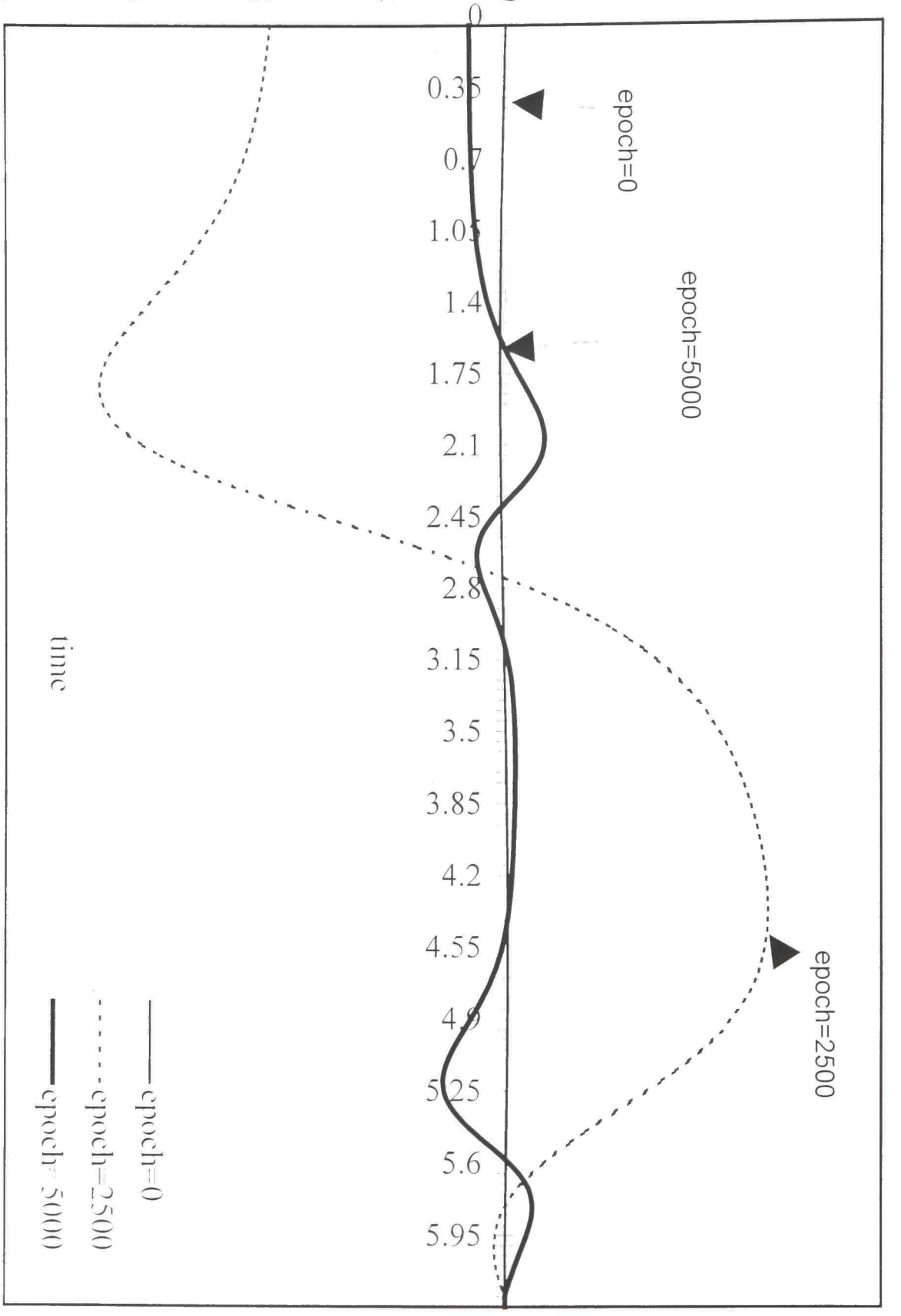

Figure 3.18. z1 value of the hidden neuron 2 with step size 0.05

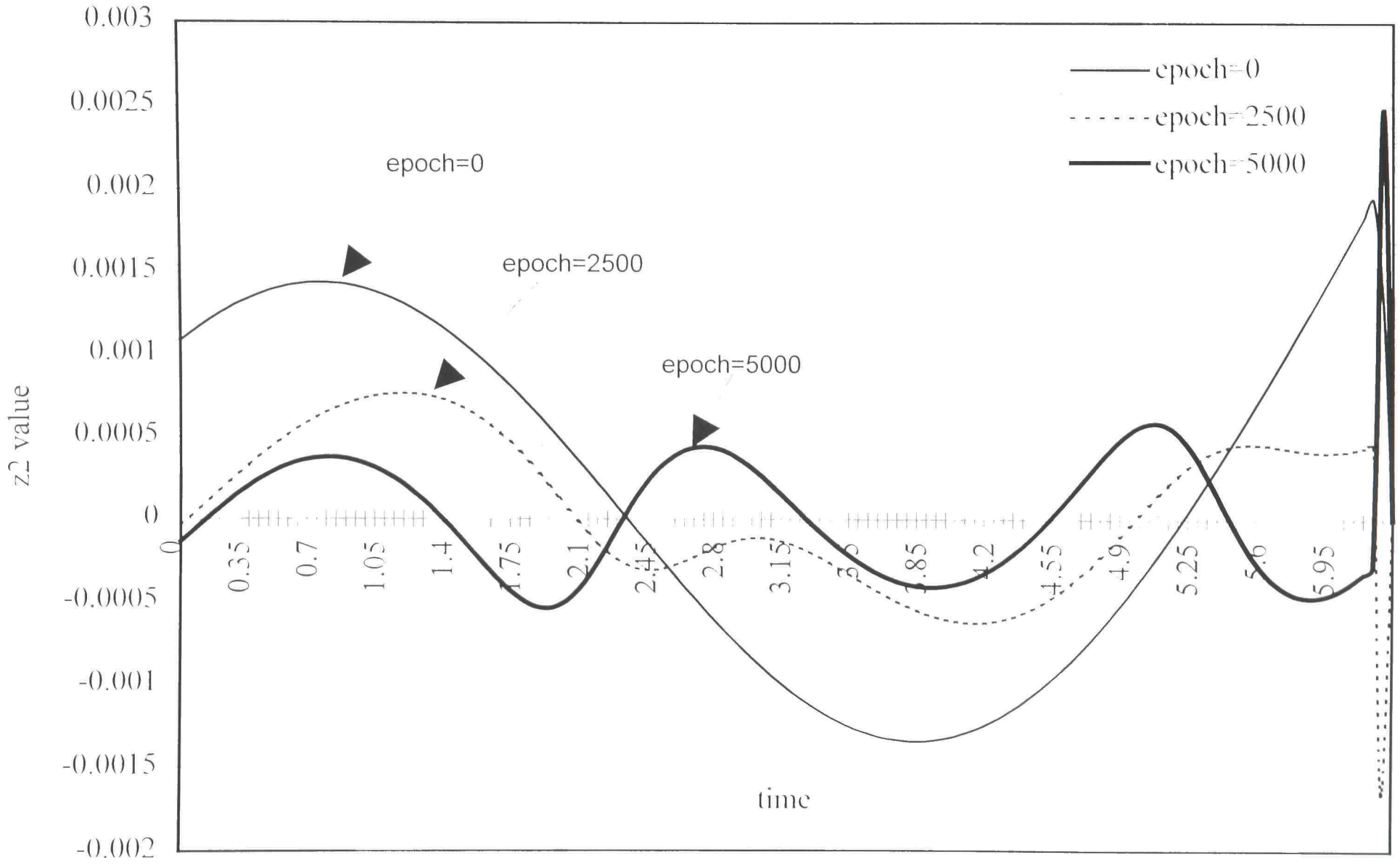

Figure 3.19. z2 value of the output neuron with step size 0.05

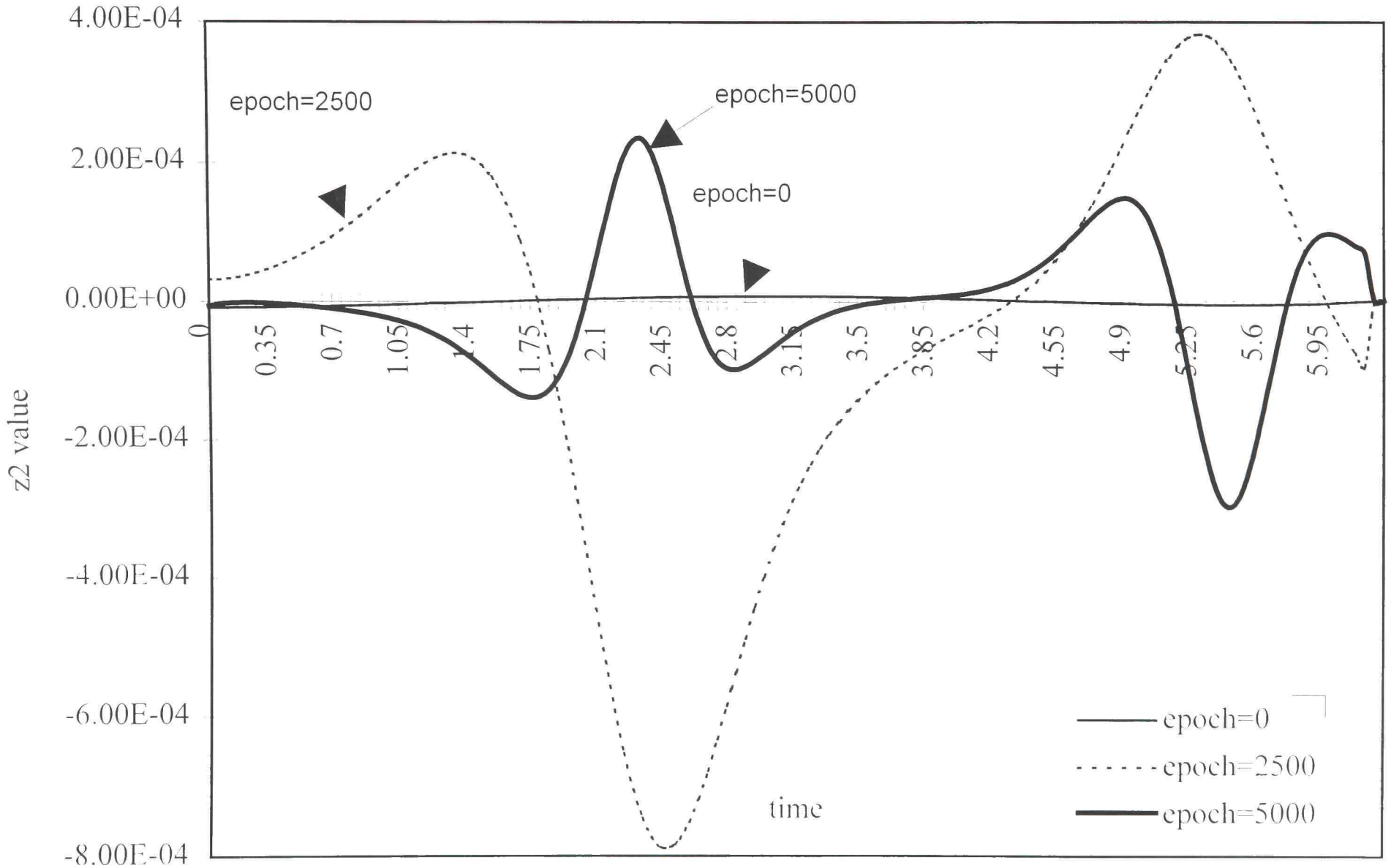

Figure 3.20. z2 value of hidden neuron 1 with step size=0.05

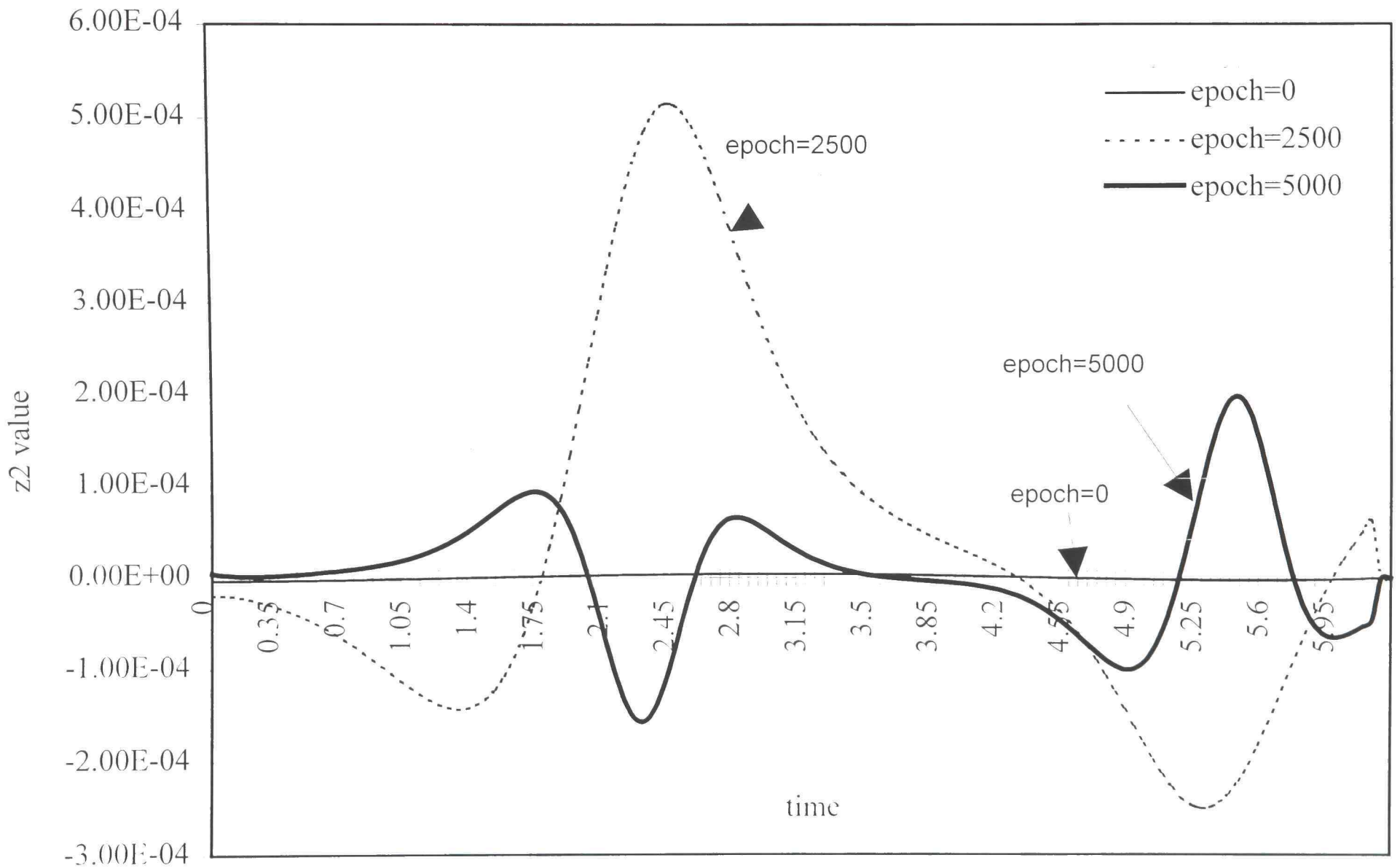

Figure 3.21. z2 value of hidden neuron 2 with step size=0.05

From Figure 3.12. we can see that the curve predicted by recurrent neural network fits the sine wave better. The main error still occurs near the maximum and the minimum of the sine wave. We can see the tendency of z, $z^{(1)}$ and $z^{(2)}$ from the following figures that as the training proceeds. the z values decrease. Also we should notice the different scales we used in the figures. In the case of the step size of 0.05. the scale is smaller than that of the step size of 0.1. This also shows the improvement of the training results. However. when we analyze the results, we take a small interval in the sine wave curve. for instance, the interval [3,3.6], where the sine wave is close to linear. This can be seen in Figure 3.12. Comparing to what we got in the case of step size equal to 0.1. the first-order difference of z gets closer to a constant in this small interval of the sine wave curve and the second-order difference of z is closer to zero. Since it is still not what we expected. we further decreased the step size to see the behaviors of the z equations.

Also from the figures above, we can see the tendency of the z values. the first-order difference of z and the second order of z from the curves of different iteration numbers. The curves of different iteration numbers show the tendency that when iteration number gets larger, the z value, the first-order difference and the second order difference of z are all getting smaller. This is same as what happened in the case of step size equal to 0.1. Since the first-order difference of z still does not approach to a constant in small time intervals. we need to use a even smaller step size.

We can also see the changing trends of the z value, the first-order difference and the second-order difference of z of the hidden neurons are different from those of the output neuron. It is similar to the case of step size equal to 0.1.

We can also see the changing trends of the z value, the first-order difference and the second-order difference of z of the hidden neurons are different from those of the output neuron. They are also similar to those of the case of step size equal to 0.1. That is, the z values of output neuron keep decreasing along the training procedure while for hidden neurons, the absolute values of z first increase then decrease. Notice that scales we used for the output neuron and the hidden neuron are different. The scale for the output neuron is bigger than that for the hidden neurons.

We can get more information from above figures. First, we notice that in Figure 3.13, which shows the curves of the z values of the output neuron, the z values of the output neuron are closer to 0 at the end of the training than those in Figure 3.3. The same results can be obtained for the hidden neurons and they are even more apparent. This can be obtained by comparing Figure 3.4 with Figure 3.14, which both describe the z values of the hidden neuron 1.

Second, in Figure 3.16, which shows the curves of the first order difference of z of the output neuron, we can see that at the end of the training, the z_1 value of the output neuron in the interval of [3, 3.6] is close to a constant, while this is not true in Figure 3.6. The same results hold for the hidden neurons and the trend of the hidden neurons is even more apparent. This can be obtained by comparing Figure 3.7 with Figure 3.17, which both describe the first-order difference of z value of the hidden neuron 1.

Third, in Figure 3.19, which shows the curves of the second-order difference of z, we can see that at the end of the training, the z_2 value of the output neuron in the interval of [3, 3.6] is close to zero, while this is not true in Figure 3.9. Notice different scales we

used in the two figures. The scale in Figure 3.19 is one order smaller than the scale in Figure 3.9. We can also see that the same results hold for the hidden neurons. The trend of the hidden neurons is even more apparent. This can be obtained by comparing Figure 3.10 with Figure 3.20, which both show the second order of difference of z values of the hidden neuron 1.

From the analysis of the first-order difference and the second-order difference of the Lagrangian multiplier, we can decide if smaller step size is needed.

3.3.3 Case 3: step size $\Delta t = 0.01$

Finally, we turn to even smaller the step size of 0.01, which is the smallest step size used in this study. Further decrease of step size is deemed unnecessary since the trend is obvious and smaller step size can not provide more information. More important, as the step size becomes smaller, the number of points used in the training will increase greatly and this will increase the propagation error significantly. So the step size smaller than 0.01 is not used in this study.

For the step size of 0.01, the number of points we need to generate for the training is equal to $2\pi/0.01$. The prediction of the neural network after training is shown in Figure 3.22. The sum of square errors using the step size of 0.03 is 1.85, which is equivalent to an error of 0.00217 on average per point. The error is much smaller than those of previous cases.

The z values, the first-order difference of z and the second-order difference of z are shown in the following figures from Figure 3.23 to Figure 3.31. Again we can see from the figures that different neurons behave differently during the training procedure.

Also in the sine function training, data corresponding to an iteration number are drawn in the same figure in order to keep track of the tendency of the z , $z^{(1)}$ and $z^{(2)}$ values during the training procedure.

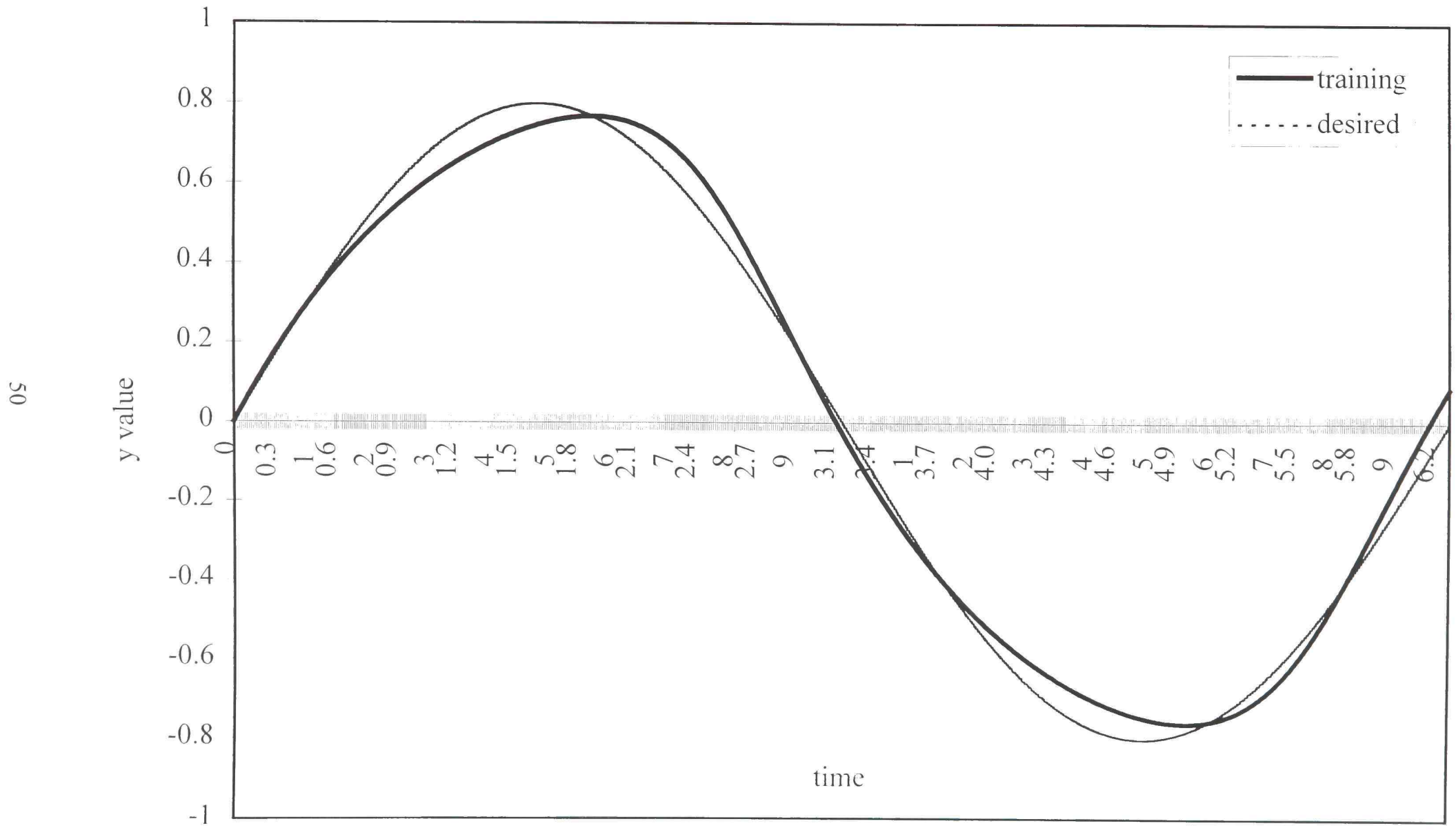

Figure 3.22. Training result of sine wave with step size=0.01

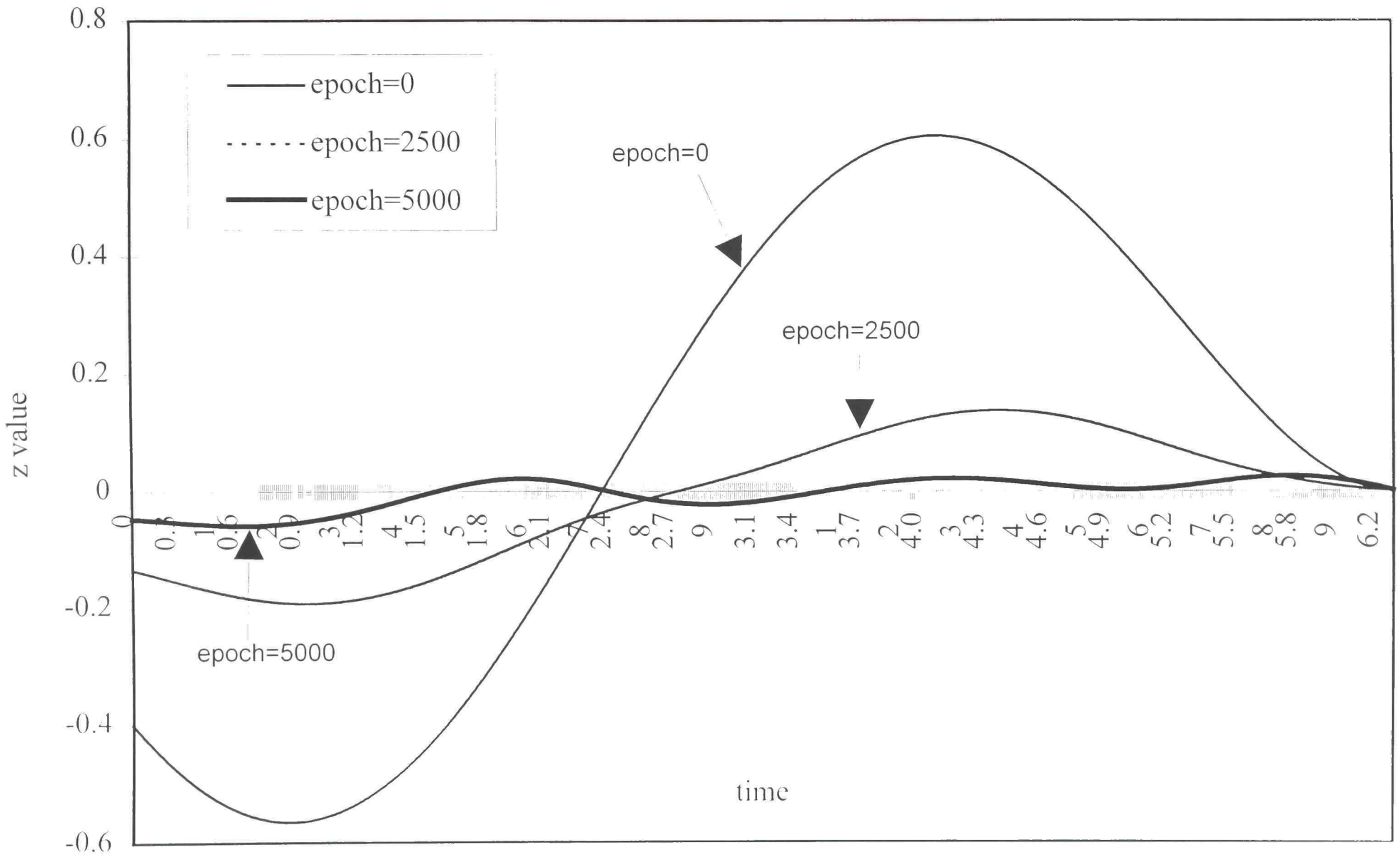

Figure 3.23. z value of the output neuron with step size=0.01

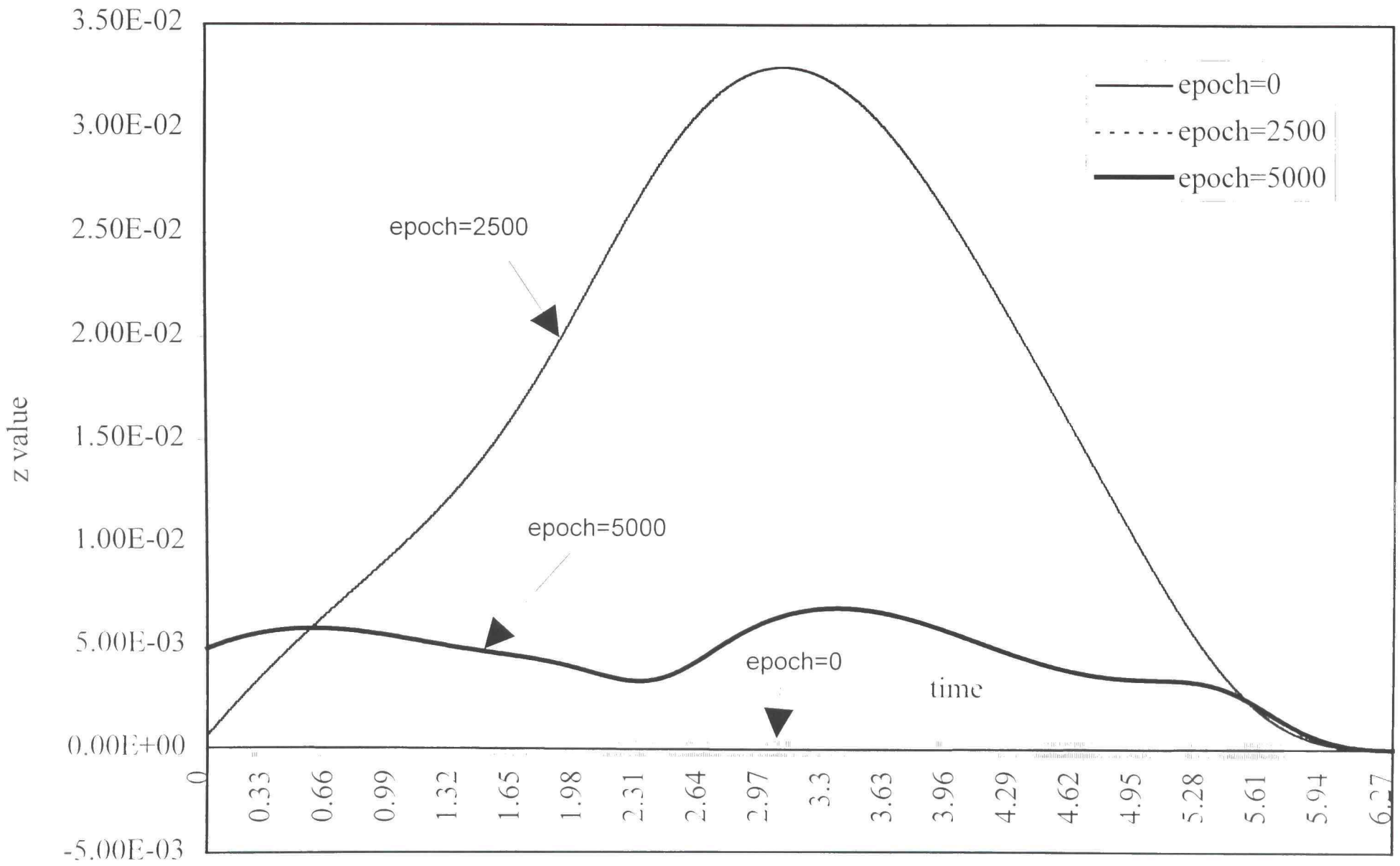

Figure 3.24. z value of hidden neuron 1 with step size=0.01

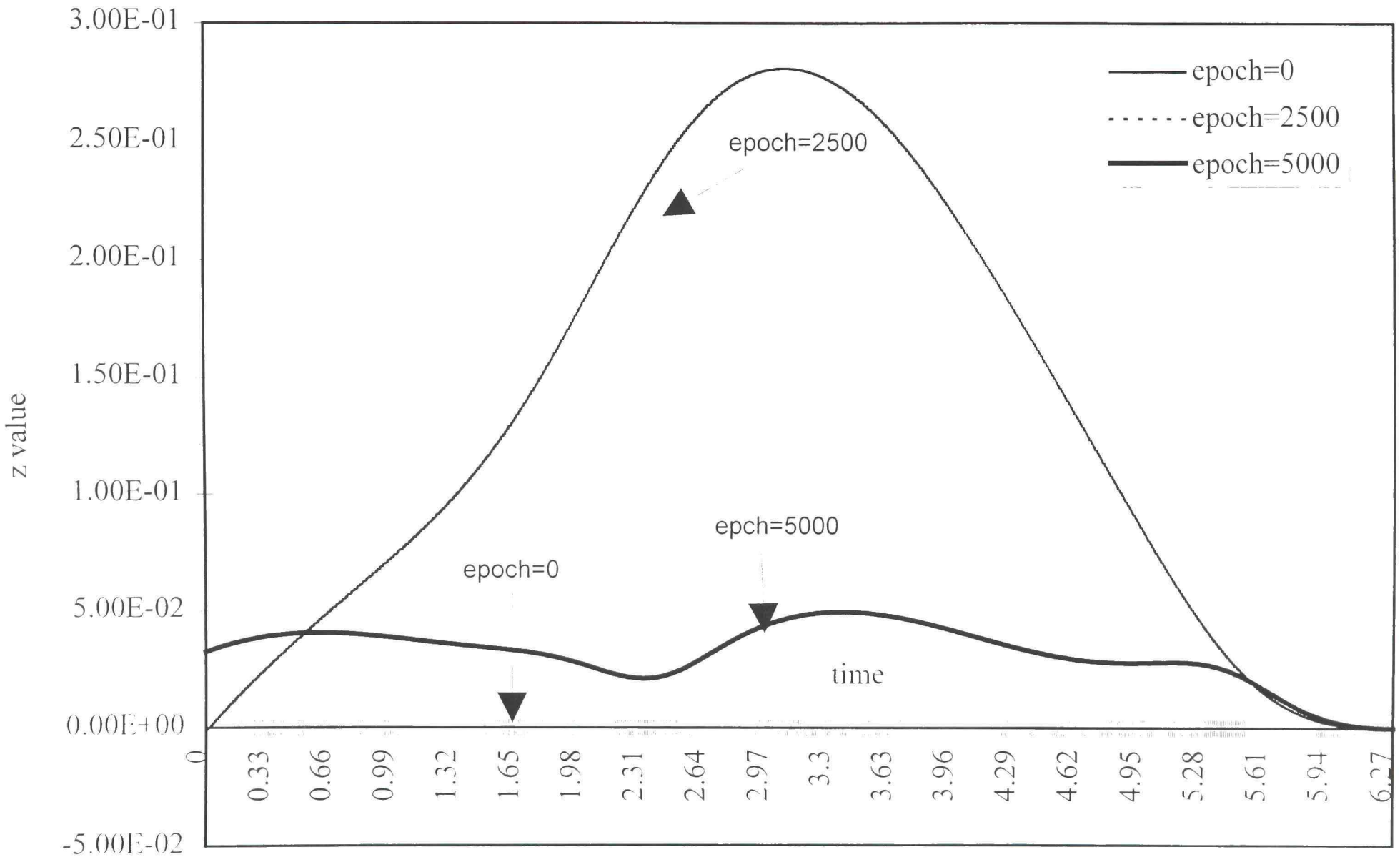

Figure 3.25 z value of hidden neuron 2 with step size=0.01

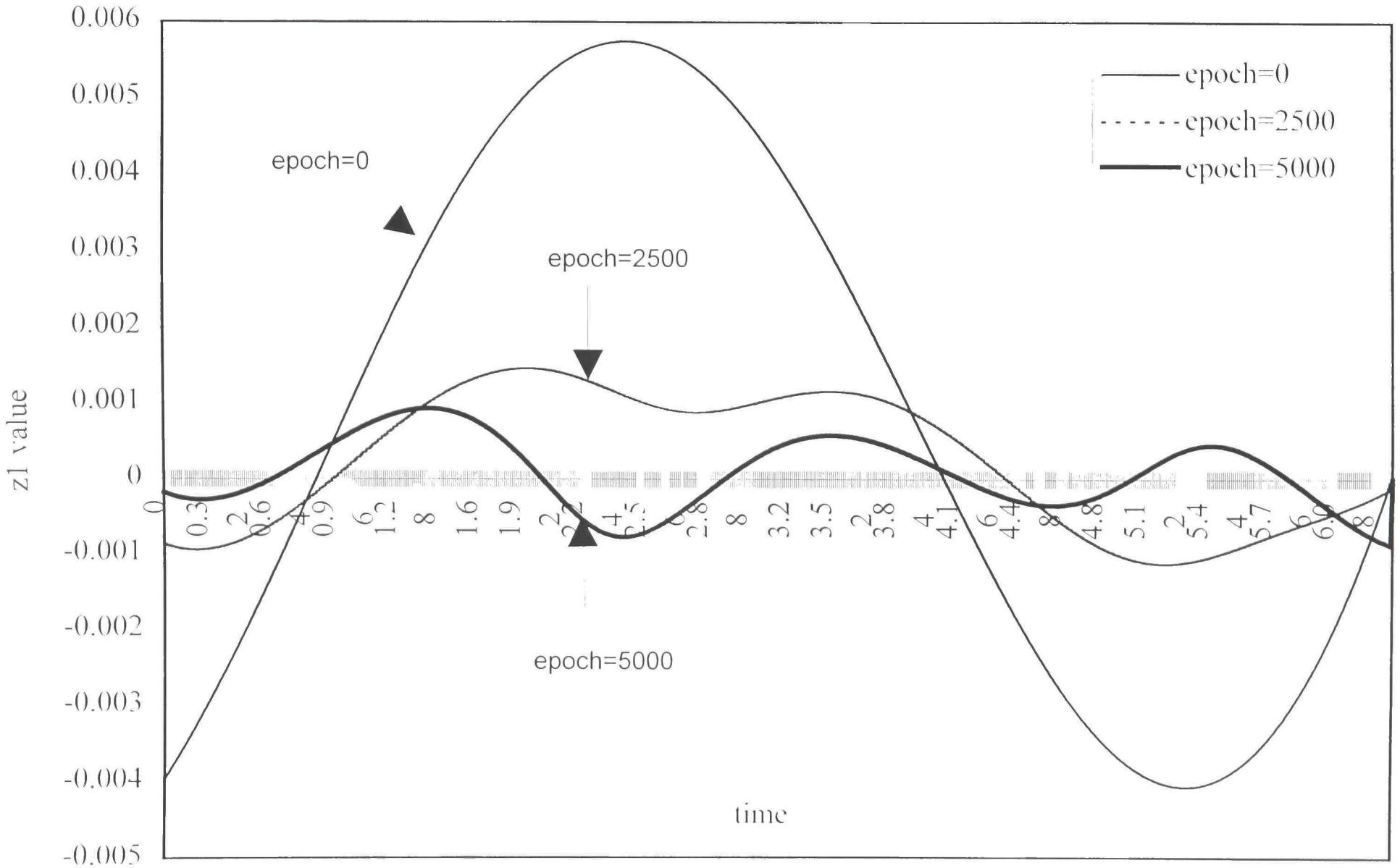

Figure 3.26. z1 value of the output neuron with step size 0.01

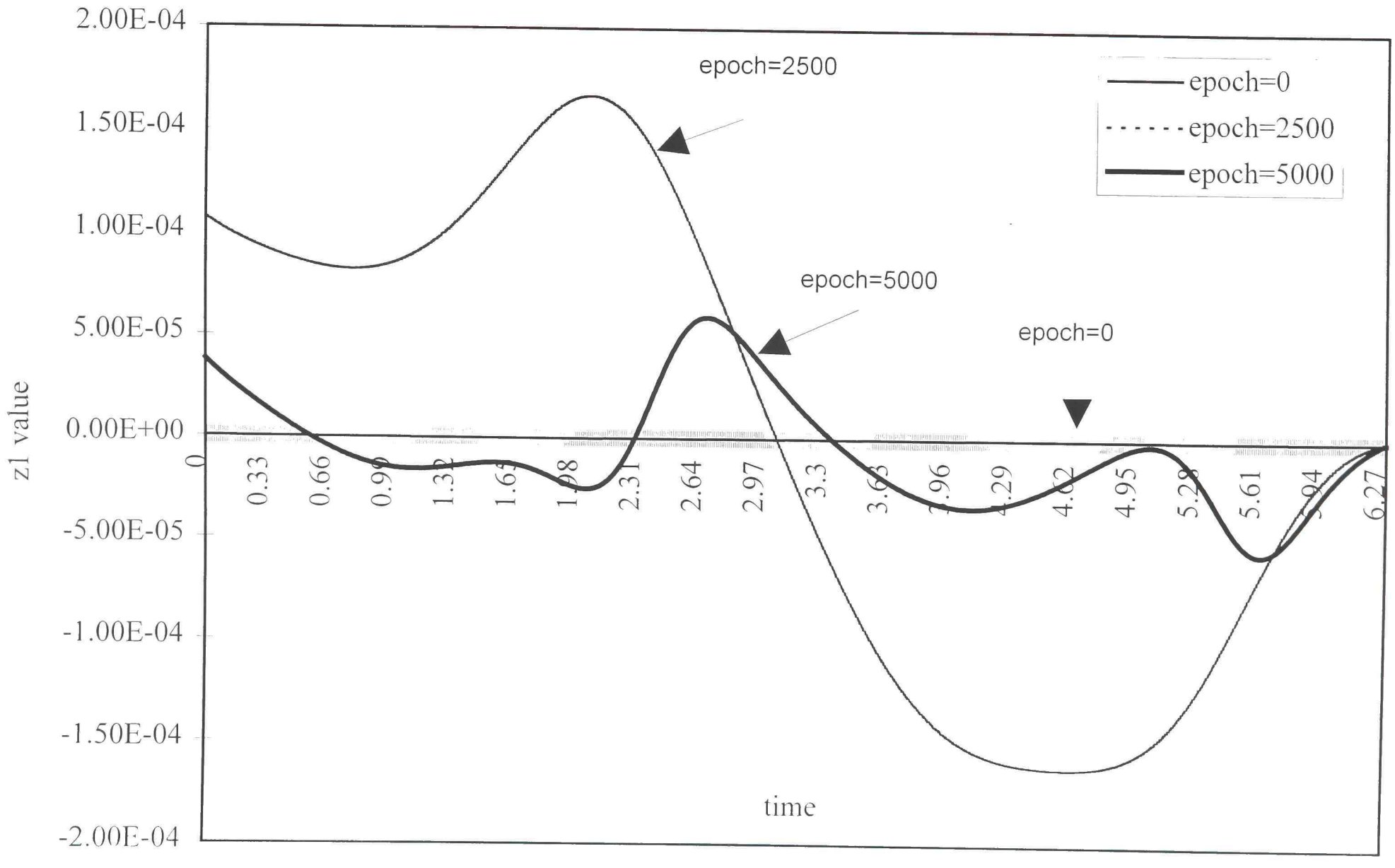

Figure 3.27. z1 value of hidden neuron 1 with step size=0.01

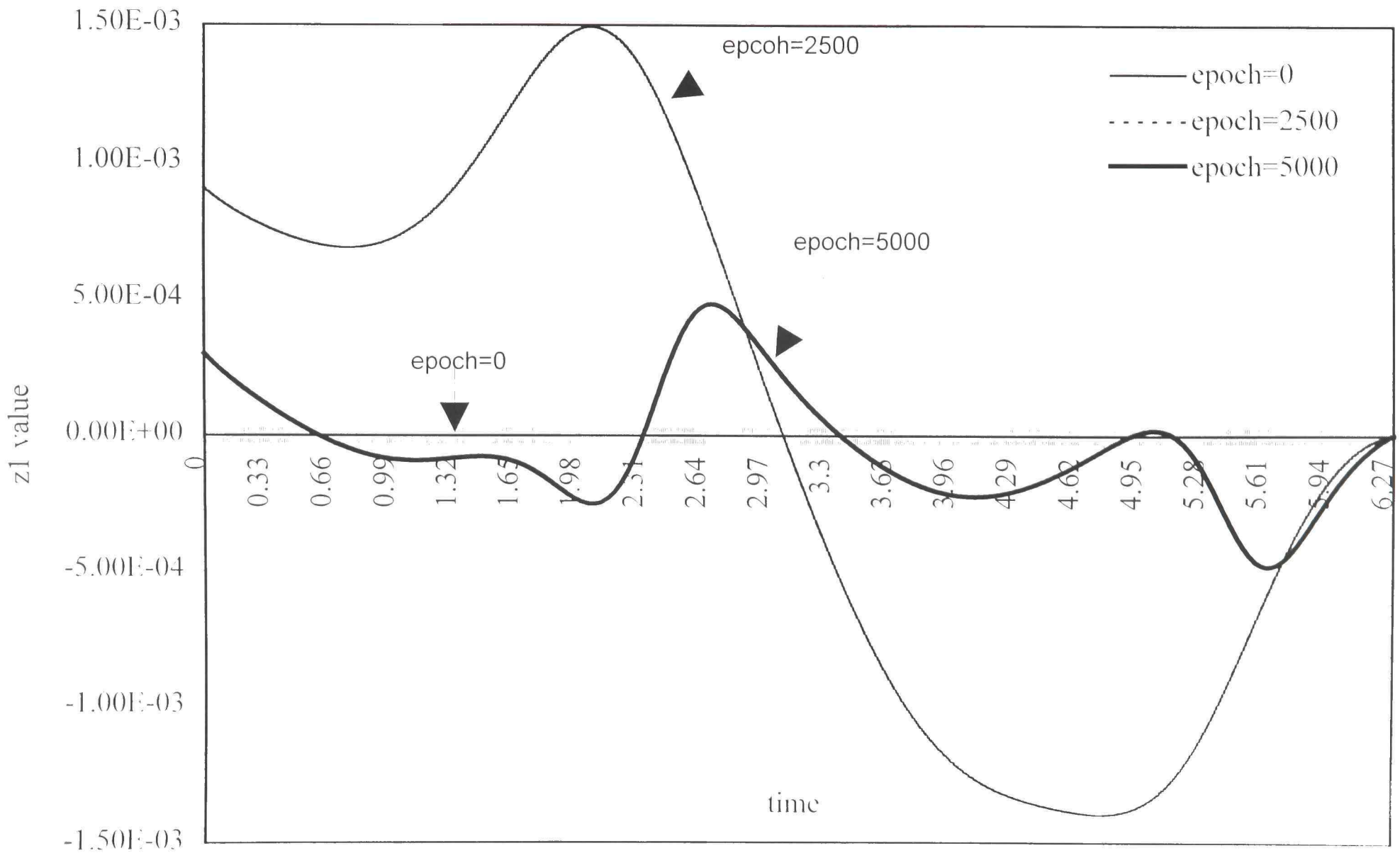

Figure 3.28. z1 value of hidden neuron 2 with step size= 0.01

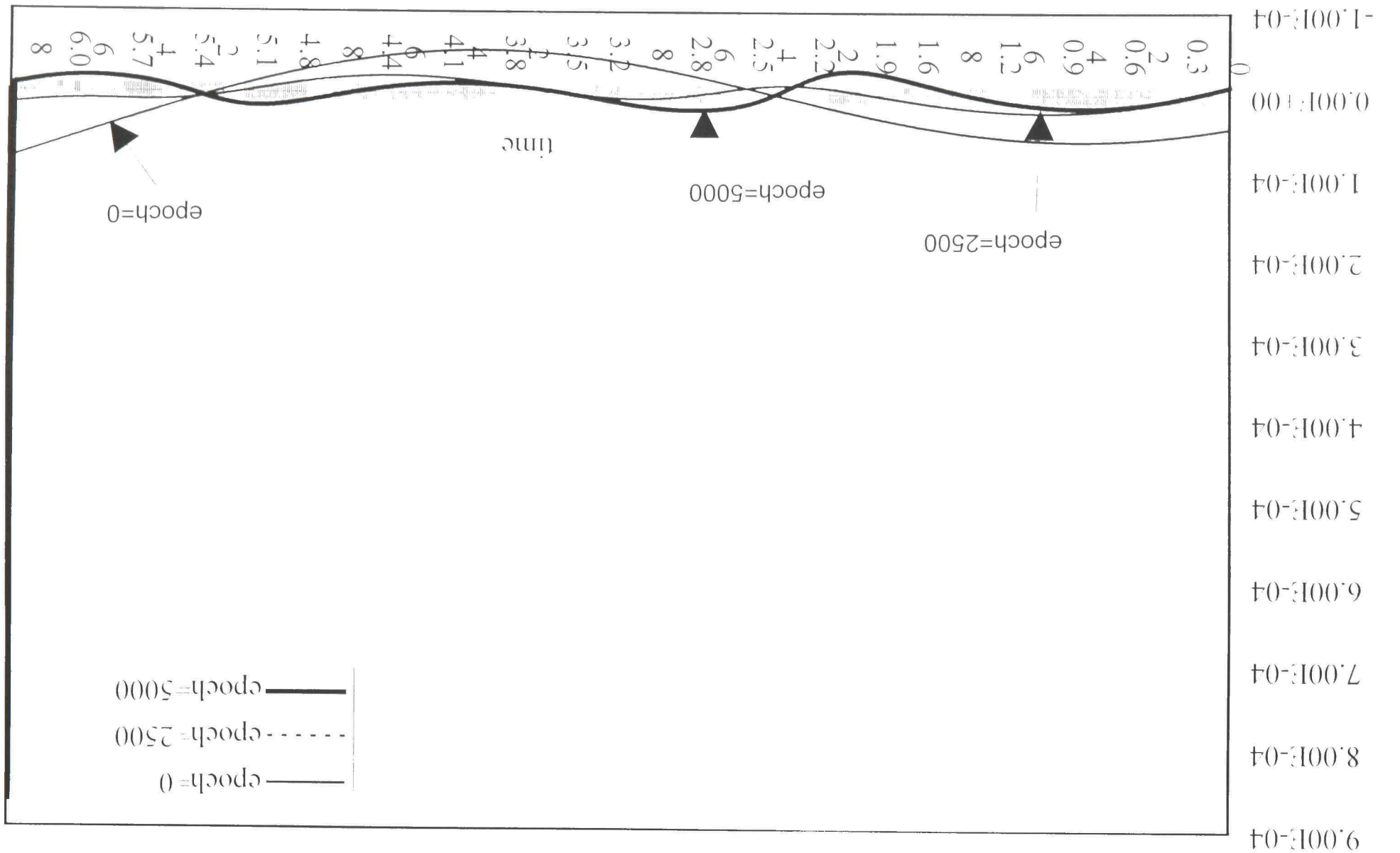

Figure 3.29. z2 value of the output neuron with step size 0.01

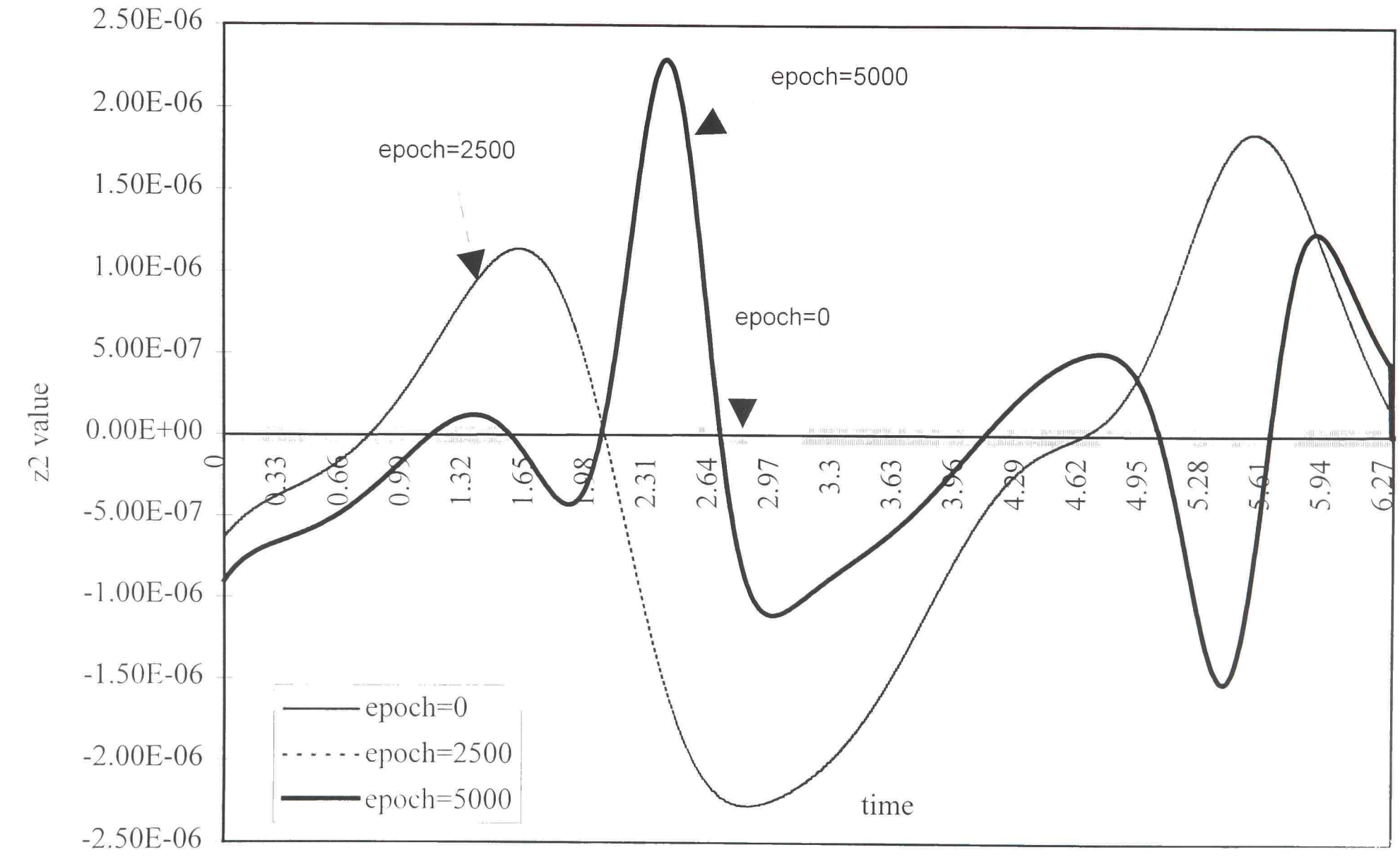

Figure 3.30. z2 value of the hidden neuron 1 with step size=0.01

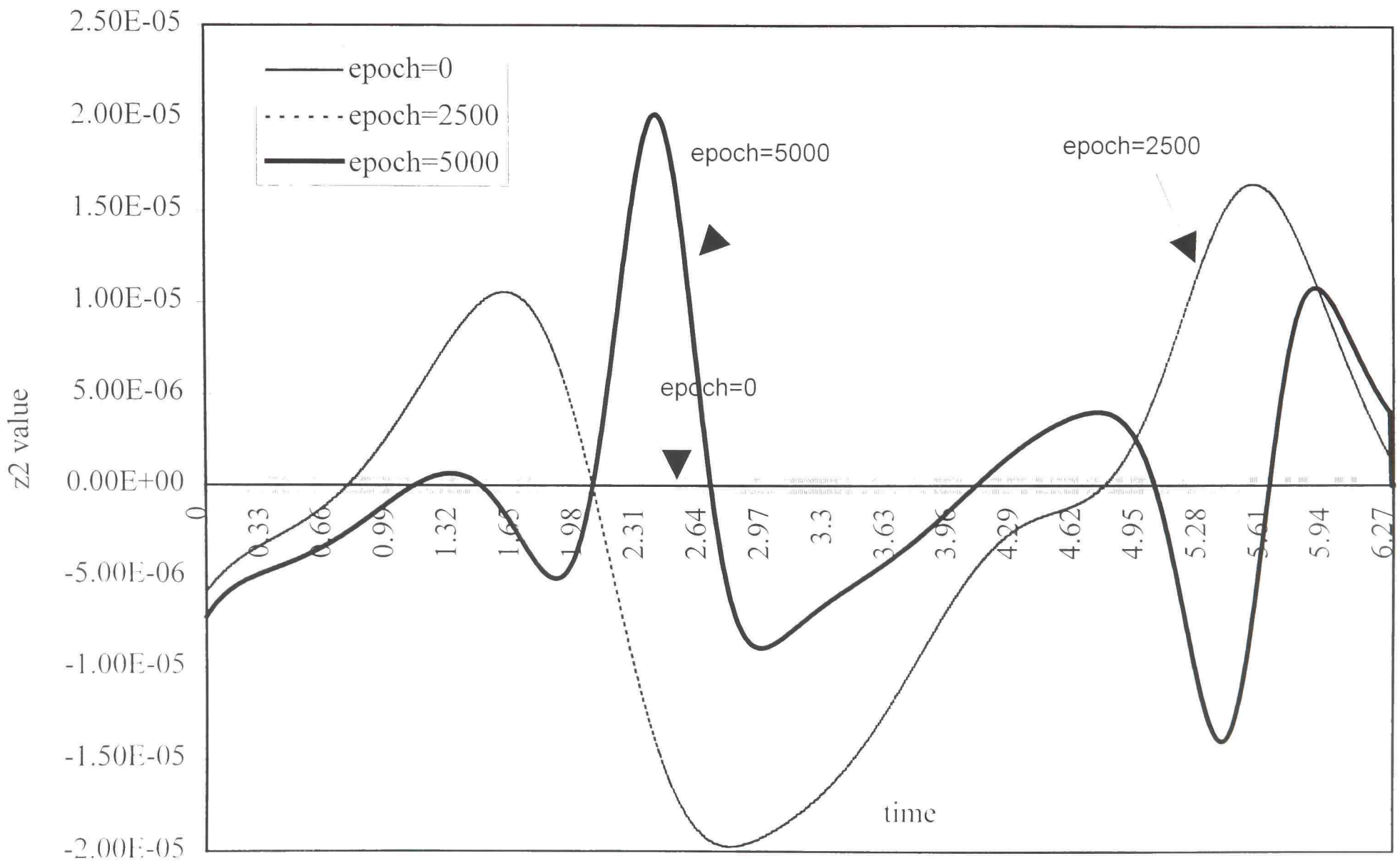

Figure 3.31. z2 value of hidden neuron 2 with step size=0.01

From Figure 3.22, we can see that the curve predicted by recurrent neural network can fit the sine wave curve very well except the areas around the maximum and the minimum. The reason that the error is great near the maximum and the minimum areas is that in those intervals, the sine curve is most nonlinear. Also we need to notice the scales used for three different step sizes are different. In the step size of 0.01, the scales are smaller than those used in the cases of the step sizes of 0.1 and 0.05. This also shows the improvement of the training result. This is what we expected. In the interval of [3, 3.6], comparing to what we got in the case of step size equal to 0.05, the first-order difference of z becomes very close to a constant and the second-order difference of z is very close to zero. We will stop the discussion of the sine wave training for the moment and we will draw conclusions in the next section.

We can get more information from above figures. First, in the training result figures, it is apparent that the results of step size = 0.01 is the best among the three cases. We can also get this from the average error per point. The average error per point in the case of step size = 0.01 is the smallest among the three cases.

Second, in Figure 3.23, which shows the curves of the z values of output neuron, we can see that at the end of the training, the z values of the output neuron is closer to 0 than those in Figure 3.13. We got the same results for the hidden neurons and the trend of the hidden neurons is even more apparent. This can be seen from the comparison between the Figure 3.14 and Figure 3.24, which both describe the z values of the hidden neuron 1.

Third, in Figure 3.26, which shows the curves of the first order difference of z, we can see that at the end of the training, the z_1 value of the output neuron in the interval of

[3. 3.6] is more closer to a constant than those in Figure 3.16. Same results can be got for the hidden neurons and the trend of the hidden neurons is even more apparent. This can be see from the comparison between the Figure 3.17 and Figure 3.27. which both describe the first-order difference of z value of the hidden neuron 1.

Fourth, in Figure 3.29. which shows the curves of the second-order difference of z of the output neuron, we can see that at the end of the training, the z_2 value of the output neuron in the interval of [3. 3.6] is more closer to zero than those in Figure 3.19. We can also see the difference from the different scales used in the two figures. The scale in Figure 3.29 is one order smaller than the scale in Figure 3.19. Same results can be got for the hidden neurons and the trend of the hidden neurons is even more apparent. This can be seen from the comparison between the Figure 3.20 and Figure 3.30. which both describe the second-order of difference of z value of the hidden neuron 1.

The figures also show that after decreasing the step size, the behavior of the z and the first-order of z and the second-order of z are close to theoretical analysis. This means that if we look close at Lagrangian multiplier and its first-order difference and second-order difference. we can get the information about the goodness of the training. If in a linear interval, the first-order difference of the Lagrangian multiplier is not close to a constant. the step size is too big and we need to decrease the step size.

3.3.4 Comparison of the results of different step size in sine wave

Form the above discussion, we can draw a conclusion that the smaller the step size Δt, the better the prediction from the neural network. The reason is that Euler`s method is used in Pearlmutter algorithm to integrate z values along the training. In the mean time, we also need to notice that if the step size is too small, the number of data points is going to increase and the training will become very slow. The accuracy of Euler's method is very sensitive to the step size as we mentioned before. Table 3.2 gives Errors and total number of data points used in the cases of different step size and in the training procedure.

Table 3.2 Comparison of training among three case studies of sine wave learning

Case no.	Step Size	Sum of square errors	Data points	Average error per point
1	0.1	0.69	62	0.0133
2	0.05	0.84	125	0.00728
3	0.01	1.85	628	0.00217

Although the total error increases with the decrease of step size Δt due to greater number of data points used, the average error per point decreases consistently. This means the prediction of the network gets closer to the real sine wave.

Another important conclusion drawn from the study is that in the small interval of the sine wave curve where sine wave is close to linear, the first-order difference of z

becomes closer to a constant with the decrease of the step size Δt. In other words, when prediction of the neural network becomes more accurate. the first-order difference of z does not change in a small interval, which agree with theoretical analysis.

Also at the end of the training, the second-order difference is close to zero and the Lagrangian multiplier should approach 0. When the step size is small enough, we can get this results. We can determine when we need to decrease the step size by looking at the Lagrangian multiplier, the first-order difference of z and the second-order difference of z during the training.

CHAPTER IV

INVESTIGATING LAGRANGIAN MULTIPLIER USING ECG DATA

In this chapter, we will investigate the Lagrangian multiplier in Pearlmutter algorithm using recurrent neural network to learn ECG data. Analysis is given throughout this chapter when necessary. Comparison between the data sets are also made when necessary.

4.1 Neural Network Architecture in ECG Data Learning

The architecture of the neural network used in training the ECG data set is: The number of hidden neurons is 6; the number of output neuron is 1; the number of input neurons is 0. The model is:

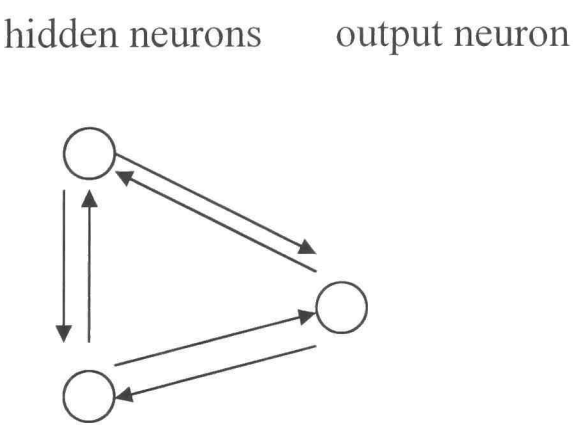

Figure 4.1. the fully connected recurrent neuron network used in ECG data learning

4.2 ECG Data Description

In training the recurrent neural network using ECG data, different step sizes are used to study the effect of step size on training results. Two data sets coming from the same data resource but with different frequency are used in the study. One frequency is 360, and the other is 90. The neural network in this research is a fully connected recurrent neural network. Pearlmutter Algorithm is again used to train the neural networks.

4.3 Case Studies of ECG Data Learning

In this section, z value and the first order of z as well as the second order of z are investigated in different cases with different step size Δt. For each case, the sum of square errors is calculated and used to measure the accuracy of the prediction from the neural network.

4.3.1 Case 1: step size $\Delta t = 1/90$

In the case of step size =1/90, The number of points used in the training of the neural network is 128. The sum of square errors using the step size of 1/90 is 2.880 which is equivalent to an error of 0.0133 on an average per point.

We can see the prediction of the neural network in Figure 4.2. It is obvious that the neural network is not able to learn the ECG data. The prediction of the neural network is almost a straight line and can not be distinguished from the x-axis. The reason is that ECG data has very complicated dynamic behavior. Although it can been seen from Figure 4.2 that ECG data has kind of periodical behavior, the cycles have different

magnitudes and slight different shape. It is easy to predict that the z value will not approach 0, neither will those hidden neurons. The reason is that the magnitude of error is very big and even small variation of y has big effect on the error. Therefore, the z value. which represent the change of total error caused by variation of neuron output y, is big.

In the ECG data training, results at different iteration numbers are recorded in order to keep track of the tendency of the z value, the first-order difference of z and the second-order difference of z during the training procedure. Iteration numbers being used can be easily changed by changing corresponding items in the input file. In this study, as what we did in the sine wave training, we use iteration number of 0, iteration number of 2500 and iteration number of 5000.

The z value, the first-order difference of z and the second-order difference of z are shown in the following figures from Figure 4.3 to Figure 4.5. The neural network used for EDG training has one output neuron and six hidden neurons. In this study, we just consider the behavior of the output neuron instead of considering the behaviors of all the neurons.

It is interesting to notice that the shapes of three curves of z value in Figure 4.3 are almost the same, though the magnitude decrease when the iteration number goes up. Especially, the curves of iteration number of 2500 and 5000 are exactly the same. It indicates that the training of the neural network can not improve the prediction significantly. The conclusion can be drawn that the training does not make the neural network learn the dynamics of ECG data. The same conclusion can drawn by looking at

Figure 4.4 and Figure 4.5. The curves of z1 and z2 are almost the same along the training procedure, which indicates the neural network is not learning.

Since the neural network can not learn the ECG data of step size of 1/90, in the next section we will try a smaller step size to see the behavior to the Lagrangian multiplier, the first-order difference of z and the second-order difference of z.

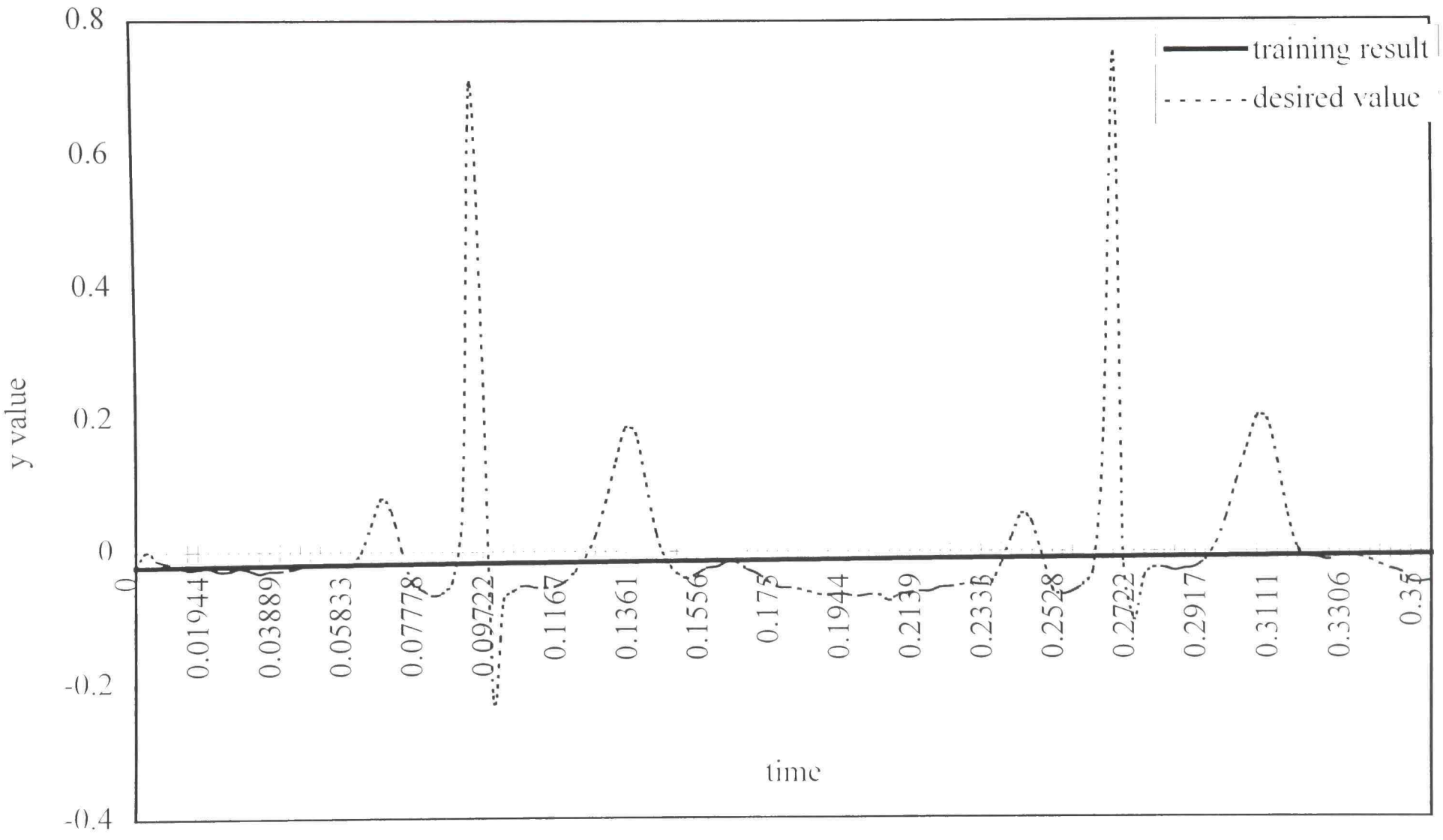

Figure 4.2. Training result of the ECG data with step size=1/90

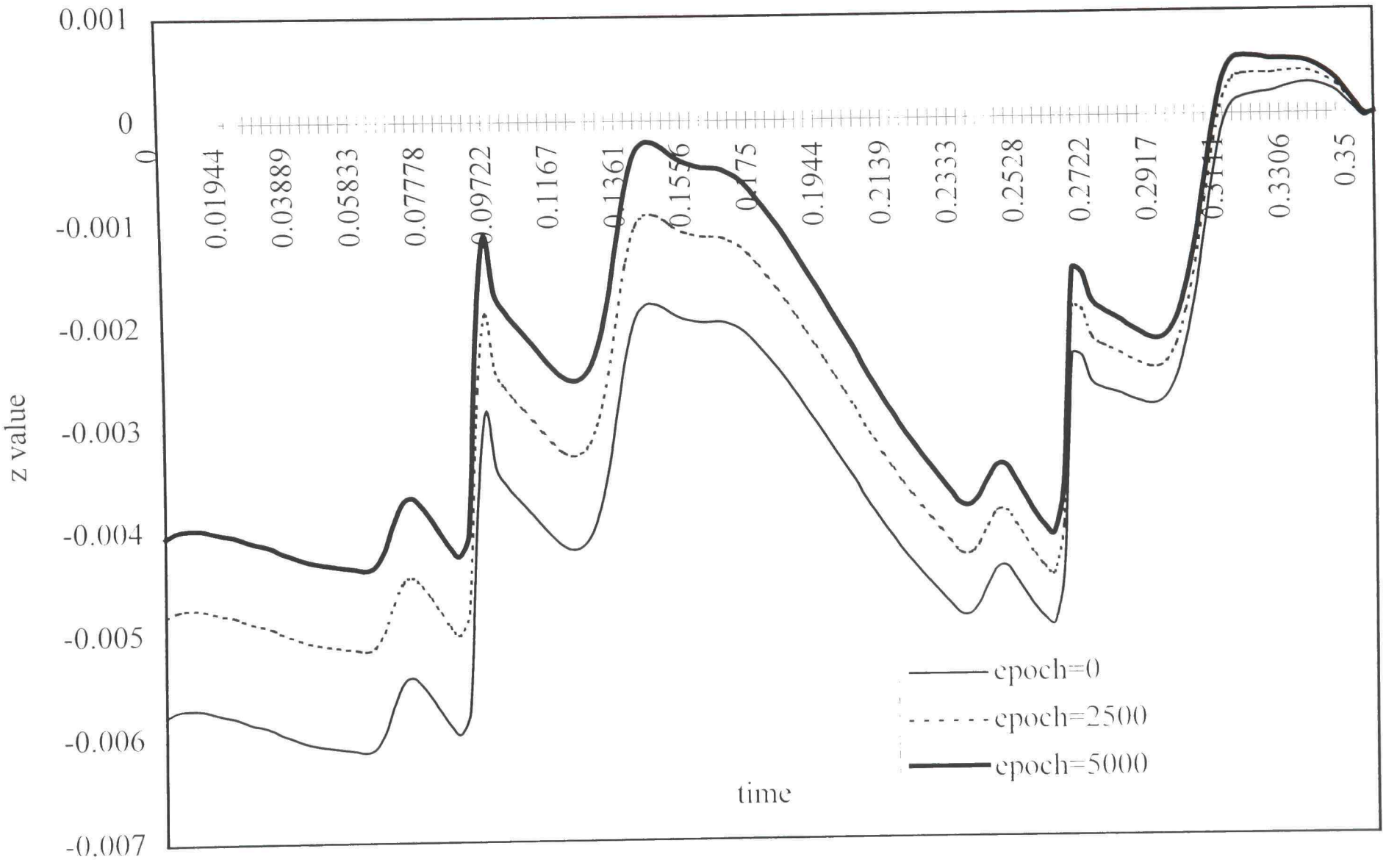

Figure 4.3. z value of the output neuron of ECG data with step size= 1/90

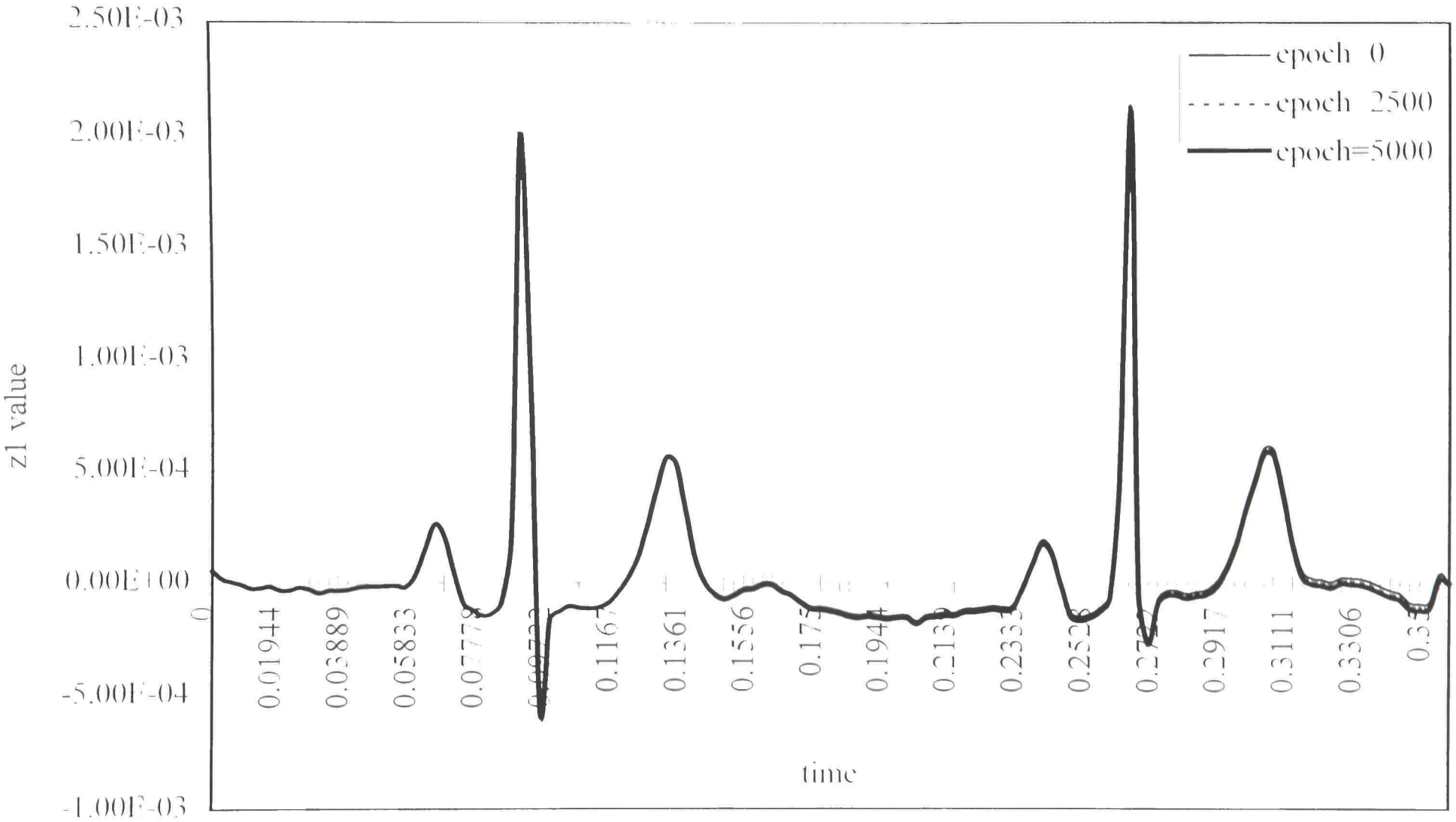

Figure 4.4. z1 value of the output neuron of ECG data with step size 1/90

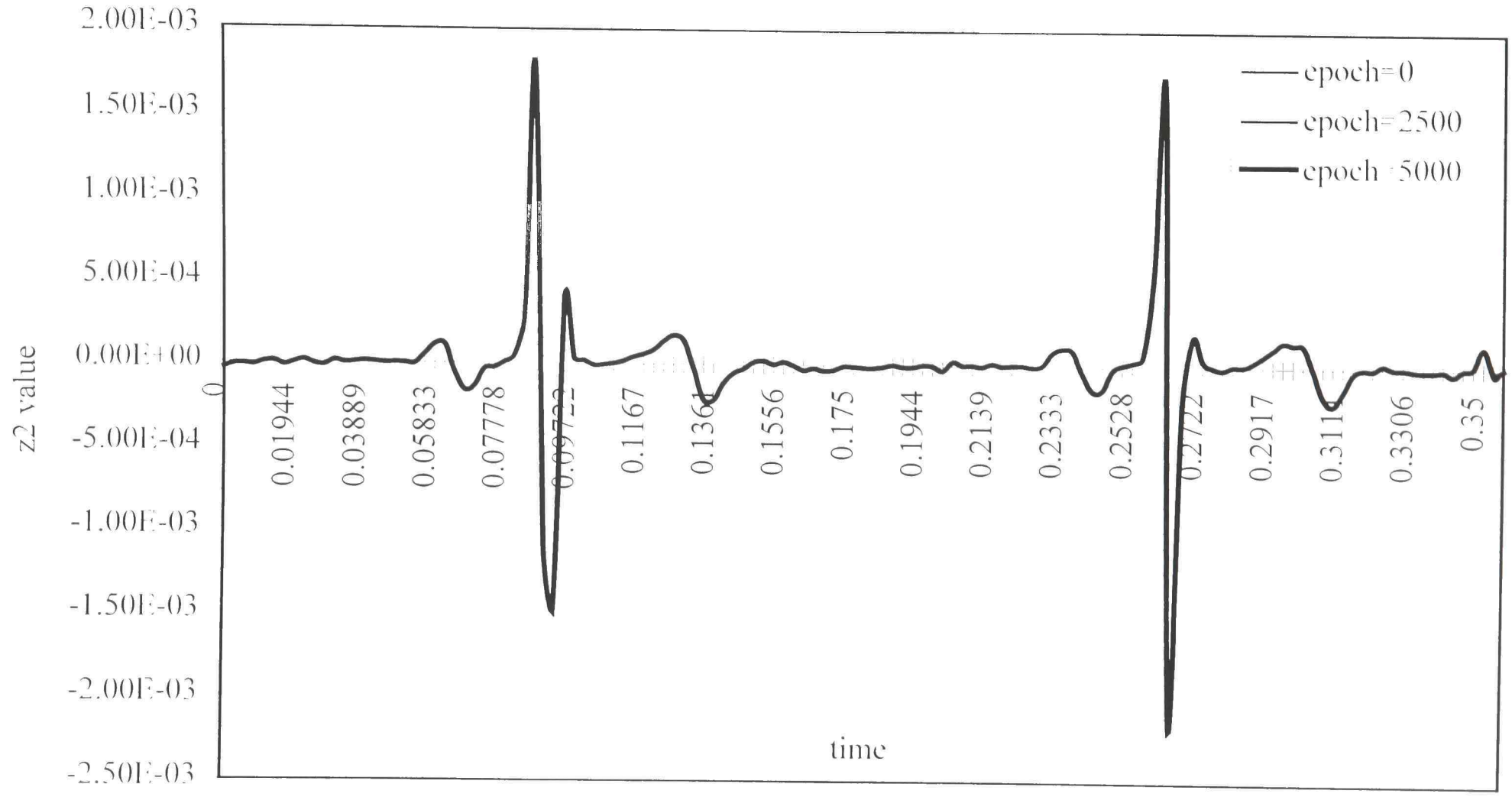

Figure 4.5. z2 value of the output neuron of ECG data with step size 1/90

4.3.2 Case 2: step size $\Delta t = 1/360$

Then we decrease the step size to 1/360 hoping that smaller step size will help the neural network training. The number of points in the training procedure increase to 512 correspondingly. The sum of square errors using the step size of 1/360 is 19.483 which is equivalent to and error of 0.0086 on average per point. The average error per point is even larger than that of the step size = 1/90. This proves that the network has a hard time to train the ECG data. The prediction of the neural network is shown in Figure 4.6. Again the prediction is close to zero. That indicates that the neural network can not learn the ECG data, at lease using the architecture of the neural network in this study.

The z value, the first-order difference of z and the second-order difference of z are shown in the following figures from Figure 4.7 to Figure 4.9. In this study, we just consider the behavior of the output neuron instead of considering the behaviors of all the neurons.

Also in the ECG data training, outputs at different iteration numbers are recorded in order to keep track of the tendency of the z value, the first-order difference of z and the second-order difference of z during the training procedure. We again see that in Figure 4.7, the z values do not approach to zero and all three curves have same shape although the magnitude decrease when the iteration number goes up. Again, the conclusion can be drawn that the training does not make the neural network learn the dynamic of ECG data. In Figure 4.4 and Figure 4.5, the curves of z1 and z2 are almost the same for different iteration numbers, which also indicates the neural network is not learning.

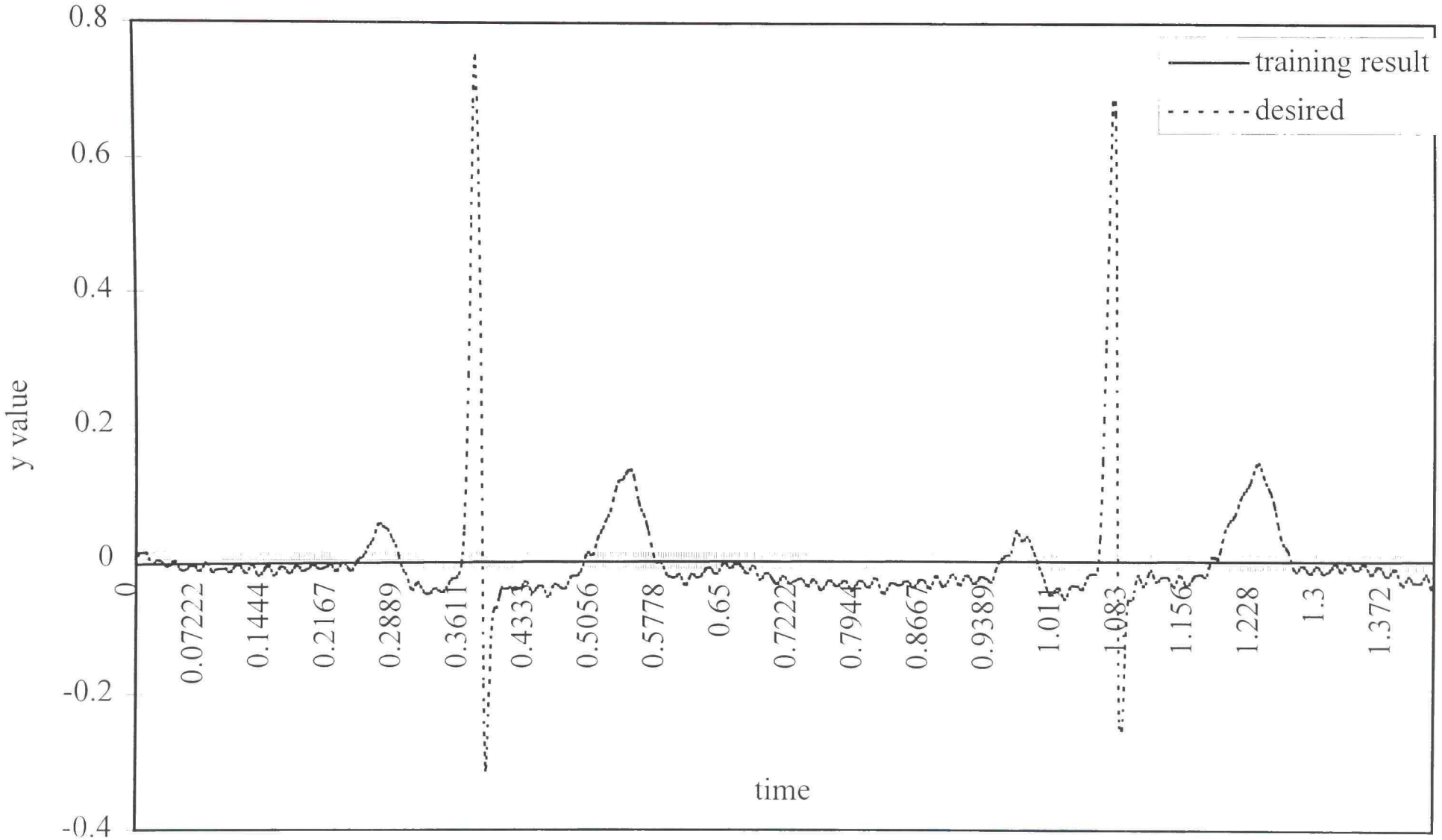

Figure 4.6. Training result of the ECG data with step size=1/360

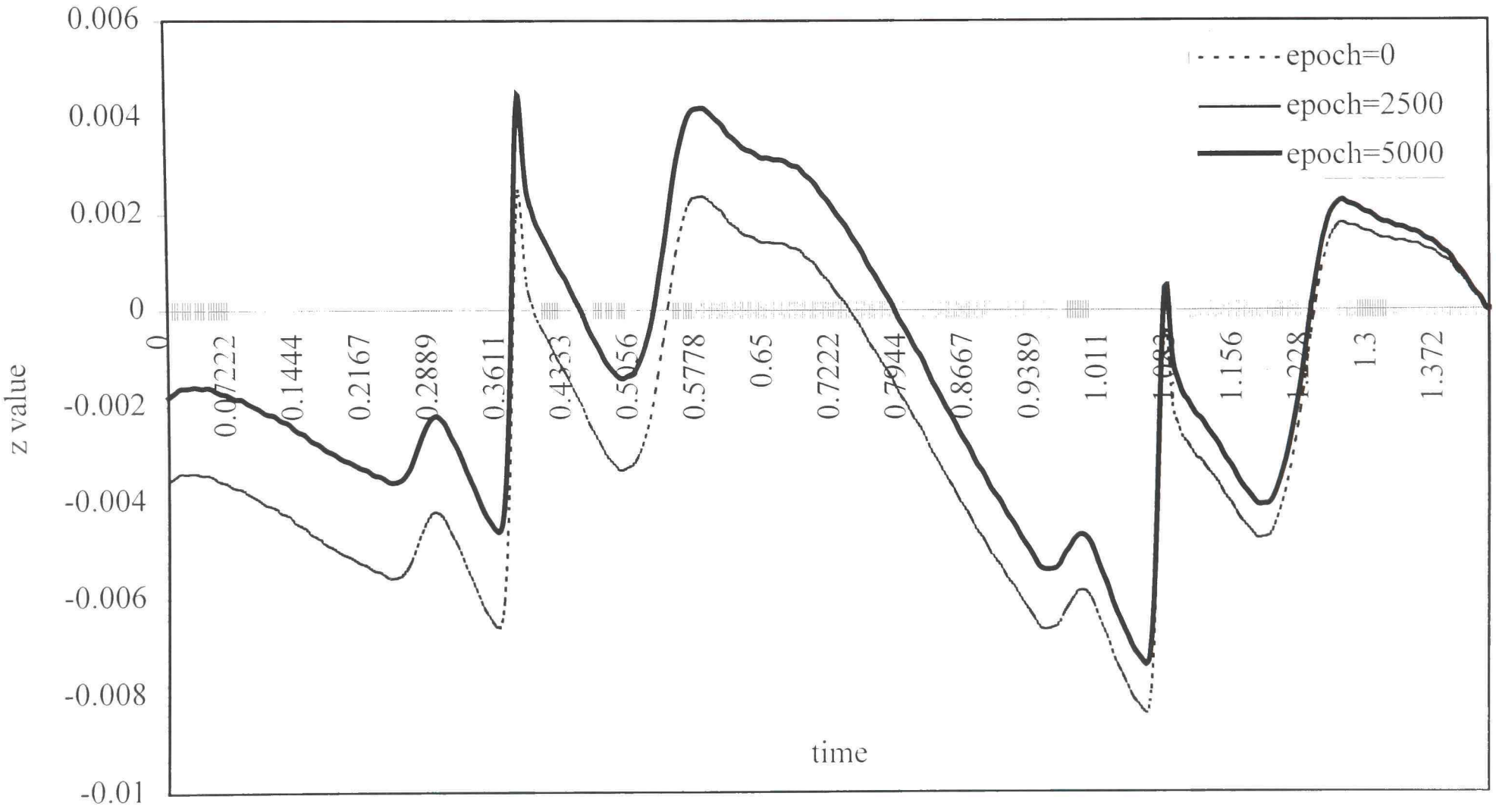

Figure 4.7. z value of the output neuron of ECG data with step size=1/360

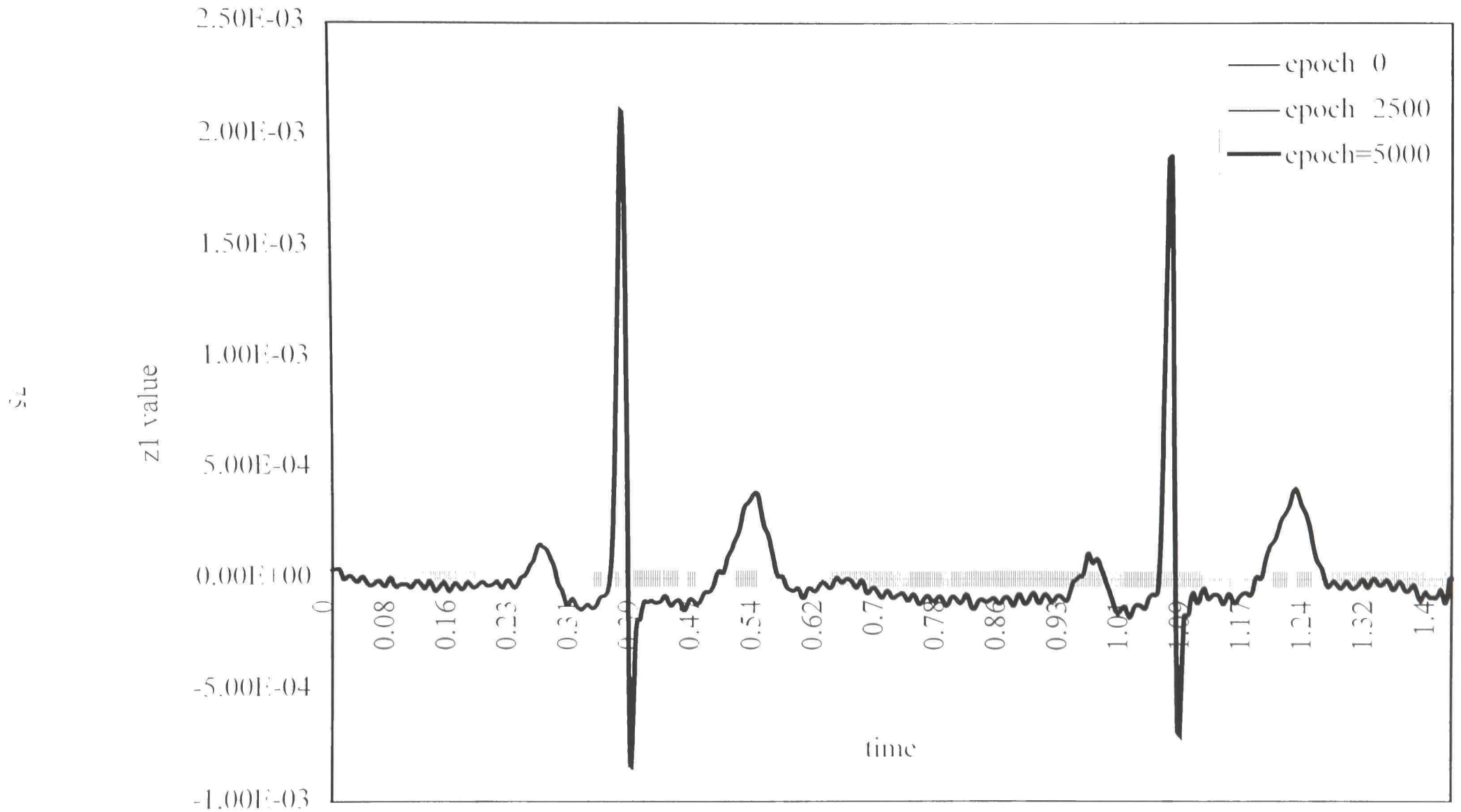

Figure 4.8. z1 value of the output neuraon of ECG data with step size 1/360

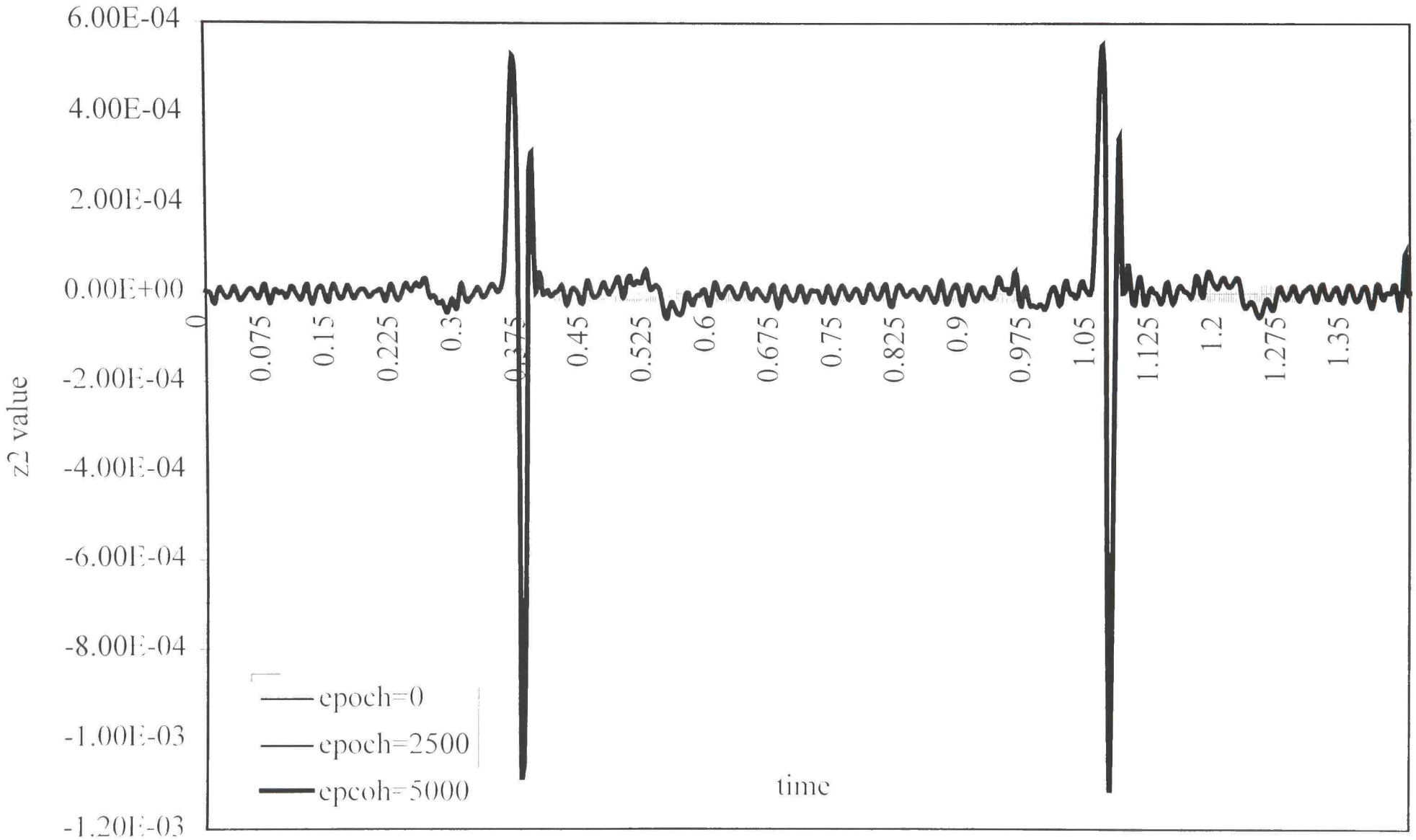

Figure 4.9. z2 value of the output neuron of ECG data with step size 1/360

4.3.3 Comparison of ECG data learning using two different step sizes

The sum of square errors of training the ECG data with different step sizes and the number of points and iteration number in the training are listed in Table 4.1. Although the case with smaller step size has smaller error per point, indeed both cases are unsuccessful.

Table 4.1 Comparison of training among two case studies of ECG data learning

Case No.	Step Size	Sum of square errors	Data points	Average error per point
1	1/90	2.880	128	0.0133
2	1/360	19.483	512	0.0086

CHAPTER V

INVESTIGATING LAGRANGIAN MULTIPLIER USING MACKEY-GLASS DATA

In this chapter, we will investigate the Lagrangian multiplier in Pearlmutter algorithm using recurrent neural network to learn Mackey-Glass data. Analysis will be given throughout the chapter when necessary. Comparisons between the different cases are also made.

5.1 Architecture Description in Mackey-Glass Data Learning

The architecture of the neural network used in training the Mackey-Glass data set is: The number of hidden neurons is 19, the number of output neurons is 1 and the number of input neuron is 0.

The model is:

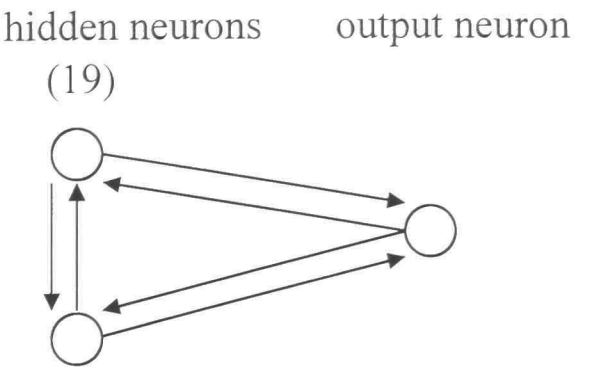

Figure 5.1. The fully connected recurrent neuron network used in Mackey-Glass data learning

5.2 Mackey-Glass Data Description

For Mackey-Glass data training. the step size of 0.09 is used to study the Lagrangian multiplier. The data points of real Mackey-Glass wave. which are the desired outputs of the neural networks and used in neural network training, are given. The neural network in this research has one output neuron, nineteen hidden neurons, and zero input neuron. This neural network is much bigger than those used previously. Especially, to obtain better learning. we used six output neurons in this study. Since Mackey-Glass data is smoother than that of the ECG data, we expect the neural network can learn Mackey-Glass data better than the ECG data.

5.3 Case Study Using Step Size Δt of 0.9

In this section, the z value, the first order of z as well as the second order of z will be investigated using step size equal to 0.9. For each case, the sum of square errors are recorded and used to make comparison among different cases.

In the case of using step size equal to 0.9. The number of points used is 211. The training result is shown in Figure 5.2. It can be seen from the figure that Mackey-Glass data shows certain periodical characteristic but not perfect. There are some small dips and humps from cycle to cycle. The neural network can learn the Mackey-Glass generally but miss the dips and humps. The sum of square errors using the step size of 0.9 is 5.768. which is equivalent to and error of 0.0114 on average per point. Generally speaking, the neural network can learn Mackey-Glass better than ECG data because of smoother curve. Table 5.1 shows the results of the Mackey-Glass training.

Table 5.1 Results of Mackey-Glass data learning

Case no.	step size	Sum of square errors	Data points	Average error per point
1	0.9	5.768	211	0.0114

The z value, the first order difference of z and the second order difference of z are shown in the following figures from Figure 5.2 to Figure 5.4. We will not draw the results of the hidden neuron this time since there are so many hidden neurons in the neural network. We will only draw the result of the output neuron.

In the Mackey-Glass data training, results of different iteration number are recorded in order to keep track of the tendency of the z value, the first order difference of z and the second order difference of z along the training procedure. Here we still use iteration of 0, iteration of 2500 and iteration of 5000 to draw curves.

We can see the tendency of z and z1 and z2 of the output neuron during the training in the following figures. At the beginning of training, the values of z are very small. After that to the end of the training, with the increase of the iteration number, z values become bigger and bigger. It can be explained as follows. The effect of the variation of output y on the total error includes direct effect and indirect effect referring to the definition of z. The neural network used in the Mackey-Glass data training is much larger than those used previously. Thus, the indirect effect of the variation of output y on total error dominates in the total effect. Since the weights are initialized randomly, much part of the indirect effect of the variation of output y of the output neuron is canceled out

in the network. With the training proceeding, the weights becomes in certain order and the indirect effect becomes much bigger. Therefore, z value keep increasing during the training procedure.

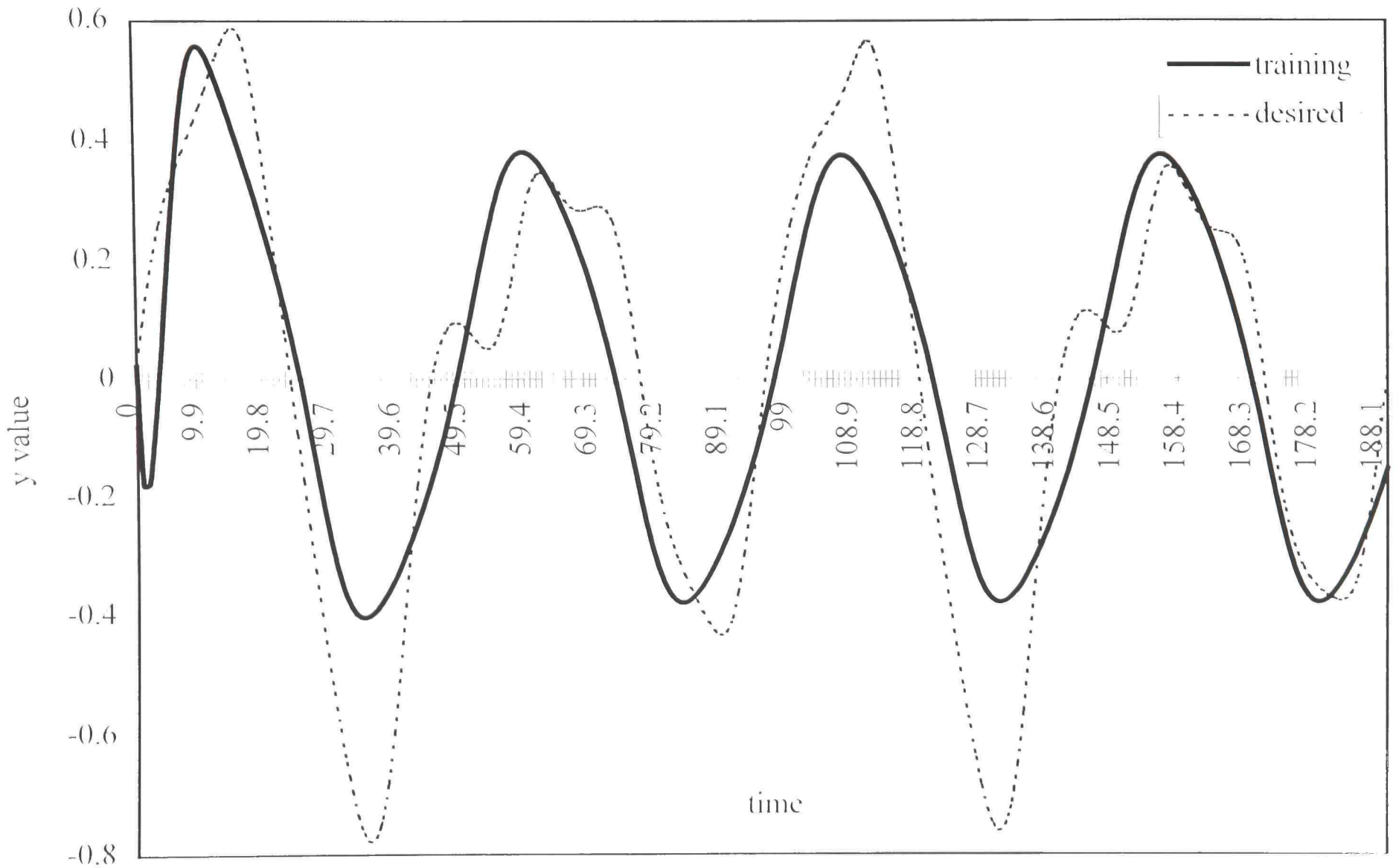

Figure 5.2 training result of the Mackey-Glass data with step size 0.9

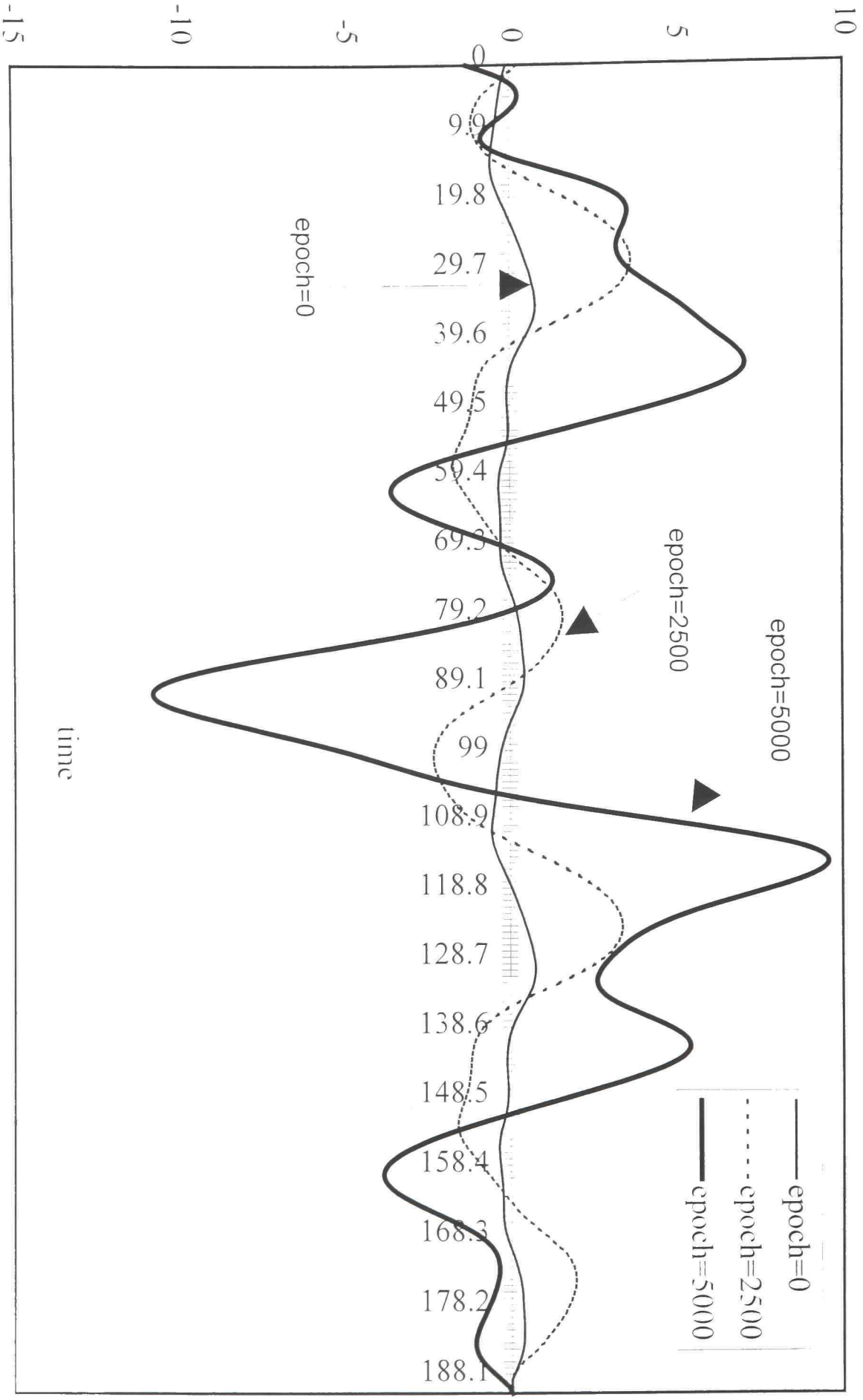

Figure 5.3 z value of the output neuron of Mackey-Glass data with step size 0.9

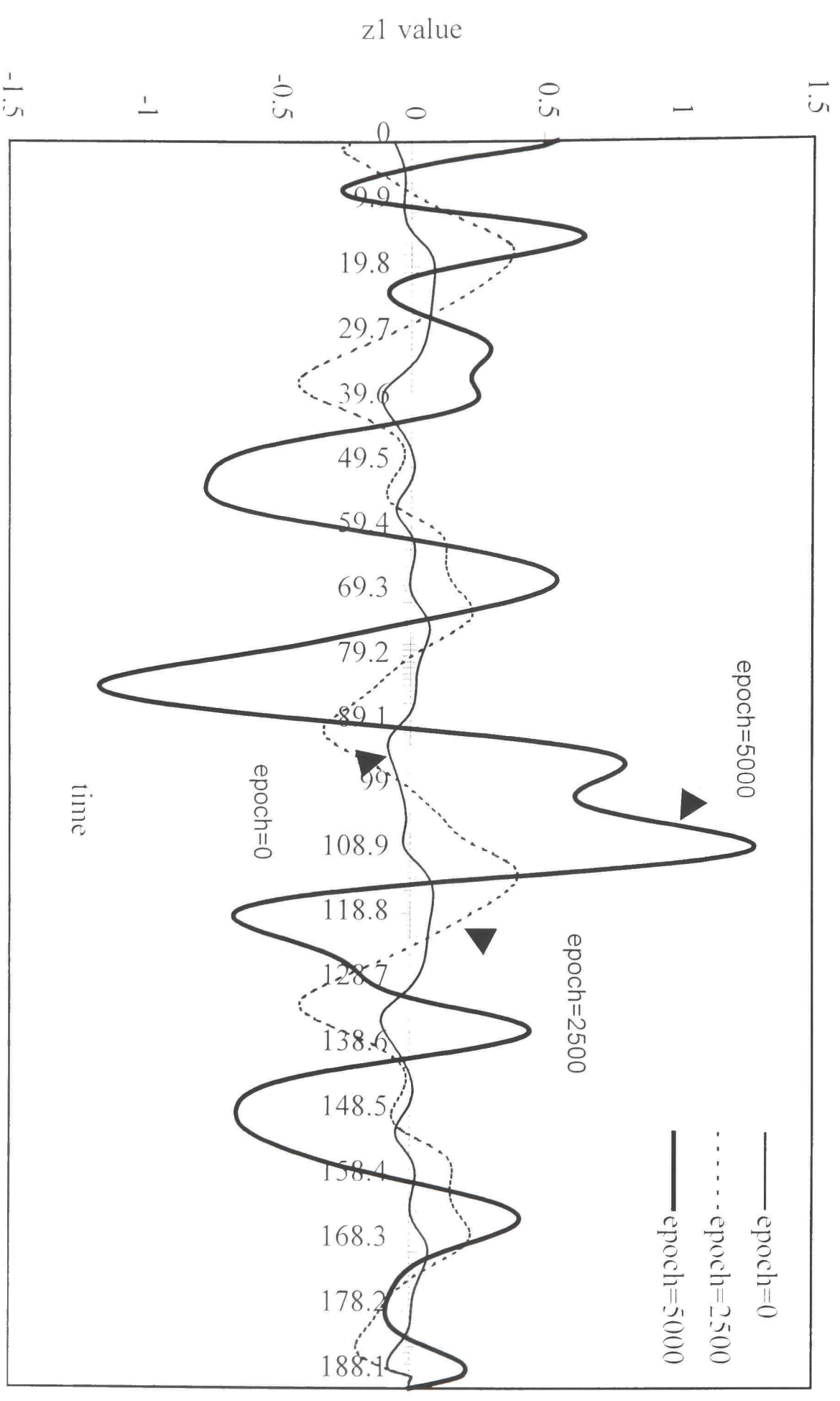

Figure 5.4 z1 value of the output neuron of Mackey-Glass data with step size=0.9

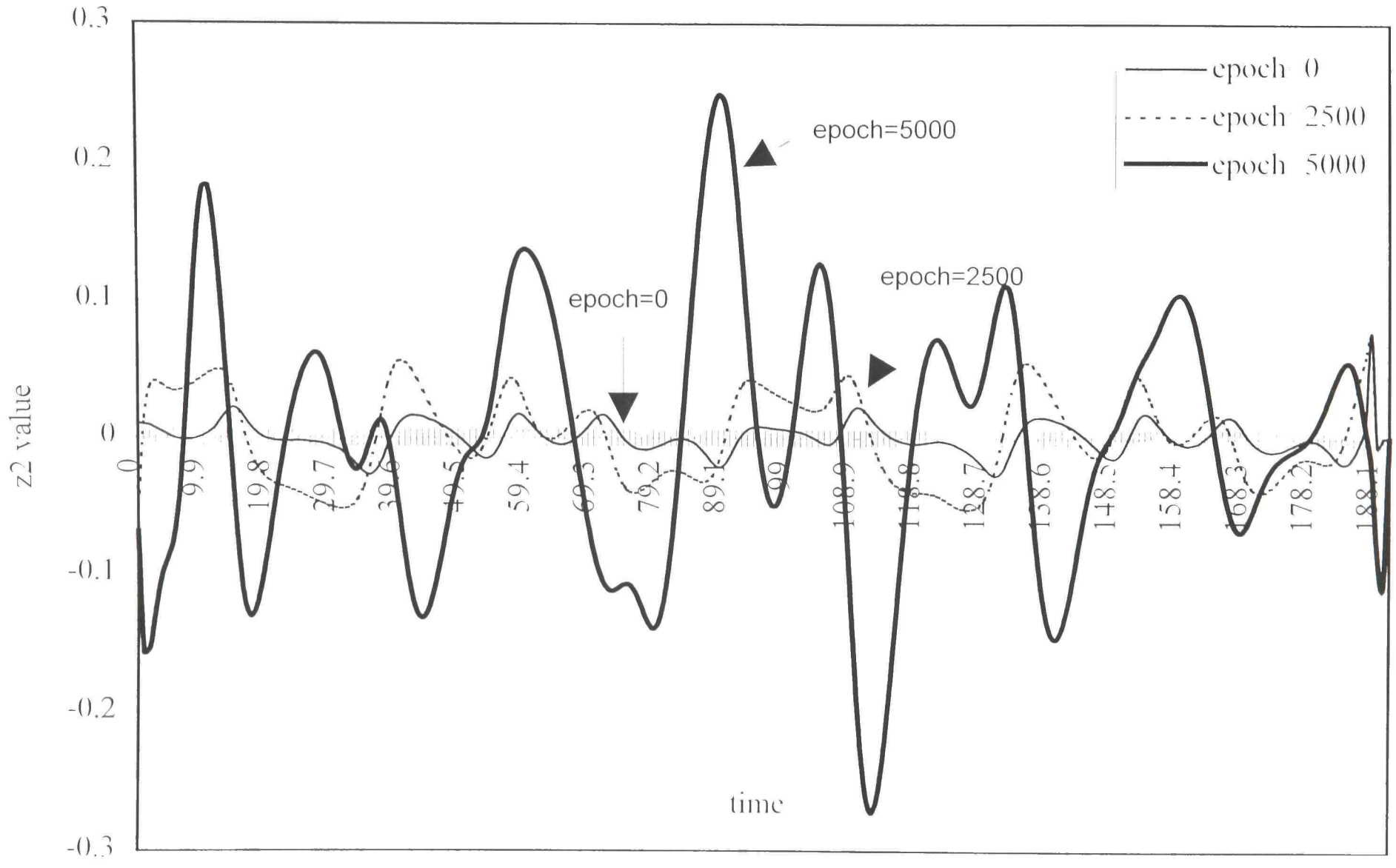

Figure 5.5 z2 value of the output neuron of Mackey-Glass data with step size 0.9

CHAPTER VI

CONCLUSION AND SUGGESTION

The main goal of this thesis was to investigate the behavior of Lagrangian multiplier in the Pearlmutter algorithm in training recurrent neural networks. In the study, three different kinds of data are used: Sine wave data, ECG data and Mackey-Glass data. The results of the latter two sets of data, which recurrent neural network used in the study is not able to learn or not able to learn well, are used to compare to those of the sine wave data that the recurrent neural network is able to learn.

In the study, we used different step size Δt in training sine wave data and ECG data. Since Euler's method is used in the integration, the smaller the step size, the more accurate the Pearlmutter algorithm is. If in a small time interval where the curve to be learned is approximately linear, the first order difference of z still should be a constant value in the same small time interval. If not, we should use a smaller step size. The conclusions drawn from this study are as follows:

1. During the learning procedure of sine wave data, when the training is good, the Lagrangian multiplier z approach to 0 at the end of training.
2. For the data that the neural network can learn well, such as the sine wave in this study, when the training is good, the value of the first-order difference of z of output neuron should approach to a constant in a small time interval where the curve to be learned by the neural network is close to linear. Consequently, the second-order difference of z of the output neuron should approach to 0 in this interval.

3. In a small linear interval in the sine wave or ECG data or Mackey-Glass data, we expect the first-order of z, which is denoted as $z^{(1)}$, approach to a constant. If this is not the case, that means the step size we chose is too big, we should decrease the step size.
4. For some cases, the z value can be used as a stop criteria as well as the sum of square errors. For example, if we have two training cases whose sum of square errors are almost the same, how can we decide which training is better? I propose to analyze the z value, the first-order difference of z and the second order difference of z. The training with smaller z value is the better training.
5. For the data that the neural network can learn well, at the beginning of training, z values of the hidden neurons are close to 0. As the iteration number increases, the magnitude of z first increases and then decreases. The absolute values of z for hidden neurons are largest at certain point in the training procedure.
6. For the data that the neural network cannot learn, such as ECG data in this study, using a recurrent neural network with only a few hidden neurons, the z values of the output neuron decrease along the training. The first-order of z and the second-order difference of z of the output neuron are very large and do not change during the learning.
7. For the data that the recurrent neural network either can learn generally but have a difficult time learning the details, such as Mackey-Glass data in this study, using a recurrent neural network with many hidden neurons, the z values of the output neuron

increase along the training. The magnitude of the first-order of z and the second-order difference of z of the output neuron also increases as training continues.

8. During each training, we may get both good training and bad training. The trends of the z value, the first order difference of z and the second order difference of z in good training are totally different from those in bad training. If we look at the results of bad training, we may get total different ideas about the trends. Due to these, when we investigate the Lagrangian multiplier, we should try to train the network many times to get best training results instead of look at results of some training which get stuck in local minimums.
9. It is well known that it is very easy to get stuck in a local minimum using the gradient method in neuron network training. Since the Pearlmutter algorithm is a gradient method, we need to run the program several times and using different initial weights trying to find the global minimum.
10. It is found that the Pearlmutter algorithm is very sensitive to the learning rate and the time interval. When a large learning rate and a large time interval are used, the learning is very difficult to converge. However, if a small time interval is used, the computation load increases sharply. When the curve being learned becomes more complicated, the energy curve becomes more complicated and has more local minimums. Thus, the learning becomes more difficult. Other learning algorithms may be more efficient in learning such complex functions. For example, according to research in our group , if we add some external inputs to the fully connected recurrent

neural network in learning ECG data, the training result is better. However, this needs to be further studied.

11. From the study, we can draw a conclusion that in the training procedure, in a linear interval, if the first-order difference of z is not close to a constant, the step size is too big and we need to decrease the step size until the first-order difference is close to a constant. However, this only holds for a curve that the neural network can learn well, such as sine wave. This does not hold for a curve that the neural networks have a hard time learning such as the ECG data. For the data that the neural networks have a hard time learning, even if we decrease the step size, we still cannot get good learning.
12. For all three cases, at the begin of training, the outputs of the all neurons and the weights are initialized to some random numbers. The error is the difference between the output of the output neuron and the desired output at each time point. Then if the error limit is not reached, the summation of square of errors are used to update the weights. After updating weights, another epoch begins. This cycle of predicting and updating continues until the number of epochs reaches the maximum number of epochs, which is set to be 5000 for all three cases. The reason that the final iteration number is set to 5000 in this study is that it is found that when the training reaches iteration of 5000, error changes very slowly. More iterations do not improve the training significantly. Thus, we use iteration 5000 as the stopping criteria in the training.

Several suggestions are made for further research:

1. In this study, we have investigated the Lagrangian multiplier in Pearlmutter algorithm using fully connected recurrent neural networks to learn three different time series data, However, the variation of Lagrangian multiplier in learning other time series data still need to be investigated in order to draw a general conclusion.
2. We use Euler's method to obtain the values of Lagrangian multiplier during the training. Using higher order integration method may get more accurate values.
3. More mathematical analysis of Lagrangian multiplier is need to further understand the relation between Lagrangian multiplier and neural network training.

REFERENCES

[1] Christiansen, B. T., "*ECG Time Series Prediction with Neural Networks.*" M.S. thesis. Texas Tech University, 1995.

[2] Connor, Jerome, Les Atlas and Doug Martin, "Recurrent Neural Networks and Time Series Prediction," *Advances in Neural Information Processing System 4*, John E. Moody, Steven J. Hanson and Richard P. Lippmann, editors, 1992.

[3] Corwin, E. M., "*Chaos and Learning in Recurrent Neural Networks,*" Ph. D. dissertation of Texas Tech University, 1995.

[4] Crutchfield, J. P. and McNamara, B. S. "Equations of motion from a data series." *Complex systems*, 1: 417-452, 1987.

[5] Dorddner G., Leitgeb, E. and Koller, H., "Toward Improving Exercise ECG for Detecting Is Chemic Heart Disease witrh Recurrent and Feedforward Neural Nets." *Neural Networks for signal processing IV*, pp.499, New York, 1994.

[6] Gersheufeld, N. and Weigend, A., 1993. "The Future of Time Series: Learning and Understanding." *In time series prediction: Forcasting the Future and Understanding the Past,* Reading, MA: Addison-Wesley.

[7] Gomez-Gil, Pilar. " Analysis of Recurrent Neural Network Trained with Backpropagation Through Time for Prediction of Electrocardiograms." *Research Progress Report.* Department of Computer Science, Texas Tech University, Lubbock, Texas, 1997.

[8] Goudreau, M. W., Giles, C. L. "Using Recurrent Neural Networks to Learn the Structure of Interconnection Networks," *Neural Networks*, Vol. 8, No. 5, pp793-804, 1995.

[9] Haykin, S. *"Communication System.."* 2nd ed. New York: Wiley, pp. 45-90, 1994.

[10] Hassoun, M. H., *Fundamentals of Artificial Neural Networks*. Cambridge, MA: The MIT Press, 1995.

[11] Kechriotis, G., Zervas, E., and Manolakos, E. S., "Using Recurrent Neural Networks for adaptive communication channel equalization," *IEEE Transaction on Neural Networks*, Vol.5, pp267-278, 1994.

[12] Lapedes, Alan and Robert Farber. "Nonlinear Signal Processing Using Neural Networks: Prediction and System Identification." *IEEE Signal Processing Magazine*, pp. 29-43, July, 1991.

[13] Logar, Antonette, "*Recurrent Neural Networks and Time Series Prediction*." Ph.D. Dissertation, Texas Tech University, 1992.

[14] Parlo, A.G., Cjpmg, K. T., amd Atiga, A.F. "Application of the Recurrent Multilayer Perceptron in modeling complex process dunamics." *IEEE Transaction on Neural Networks,* Vol. 5, pp255-266, 1994.

[15] Pealmutter, B. A., "Dynamic Recurrent Neural Networks," *Technical report,* CMU-CS-90-196, December 1990.

[16] Pineda, F., "Generalization of Back Propagation to Recurrent Neural Networks." *Physical Review Letters*, 19(59): 2229-2232, 1987.

[17] Robinson, A. J. "An Application of Recurrent Nets to Phone Probability estimation," *IEEE Transaction on Neural Networks*, Vol. 5, pp298-305, 1994.

[18] Sauer, T., 1993. "Time Series Prediction by Using Delay Coordinate Embedding," *Time series predicton: Forecasting the future and understanding the Past,* Reading, MA: Addison-Wesley.

[19] Vemuri, V. R., "Artificial Neural Networks: Concepts and Control Applications." *IEEE Computer Society Press,* 1992.

[20] Weigend, A. S., Rumelhart D., Huberman E., Bernardo A., Back Propagation, Weight-Elimination and Time Series Prediction. *Proceedings of the 1990 Connectionist Models Summer School,* pp105, Standford, CA, 1990.

[21] Weigend, A., Huberman B., and Rumelhart D., "Prediction the Future: A Connectionist Approach," *International Journal of Neural Systems*, Vol. 1, p193, 1990.

[22] Werbos, P. J., "Generalization of backpropagation with application to a recurrent gas market model." *Neural Networks*, Vol. (1), pp339-356, 1988.

[23] Werbos, P. J., "Backpropagation through time: What it does and how to do it." *Proceedings of the IEEE*, Vol. 78, pp. 1550-1560, October 1990.

[24] Williams, R. J. and Zipser, D., " A Learning Algotithm for Continually Running Fully Recurrent Neural Networks," *Neural Computation*, Vol, 1(2), 1989.

[25] Zipser, D., "Short-term Active Memory: A Recurrent Network Model of the Neuron Mechanism," *Technical Report 9003*, Department of Congnitive Science, UCSD, June 1990.

PERMISSION TO COPY

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Texas Tech University or Texas Tech University Health Sciences Center, I agree that the Library and my major department shall make it freely available for research purposes. Permission to copy this thesis for scholarly purposes may be granted by the Director of the Library or my major professor. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my further written permission and that any user may be liable for copyright infringement.

Agree (Permission is granted.)

Julian Sun — April 1

Student's Signature — Date

Disagree (Permission is not granted.)

Student's Signature — Date