Investigation on performance of neural networks using quadratic relative error cost function

Ning Zhang¹, Shui-Long Shen²-³, An-Nan Zhou⁴, Ye-Shuang Xu¹

¹ State Key laboratory of Ocean Engineering, School of Naval Architecture, Ocean, and Civil Engineering, Shanghai Jiao Tong University, Shanghai 200240, China
² Department of Civil and Environmental Engineering, College of Engineering, Shantou University, Shantou, Guangdong 515063, China.
³ Key Laboratory of Intelligence Manufacturing Technology, Ministry of Education, Shantou University, Shantou, Guangdong 515063, China.
⁴ Civil and Infrastructure Engineering Discipline, School of Engineering, Royal Melbourne Institute of Technology, Victoria 3001, Australia.

Corresponding author: Shui-Long Shen (e-mail: shensl@stu.edu.cn).

This work was supported in part by the National Natural Science Foundation of China (NSFC) (Grant No. 41672259).

ABSTRACT The performance of neural networks with quadratic cost function (MSE cost function) are analyzed in terms of the adjustment rate of weights and its performance in multi-magnitude data processing using a qualitative mathematical method based on mean squared error. However, neural networks using quadratic cost functions exhibit low weight updating rates and variations in performances in multi-magnitude data processing. This study investigates the performance of neural networks using a quadratic relative error cost function (REMSE cost function). Two-node-to-one-node models are built to investigate the performance of the REMSE and MSE cost functions in adjustment rate of weights and multi-magnitude data processing. A three-layer neural network is employed to compare the training and prediction performances of the REMSE cost function and MSE cost function. Three LSTM networks are used to evaluate the differences between REMSE, MSE, and Logcosh in actual applications by learning stress and strain of soil. The results indicate that the REMSE cost function can notably accelerate the adjustment rate of weights and improve the performance of the neural network in small magnitude data regression. Applications of the REMSE cost function are also discussed.

INDEX TERMS Optimization, Cost function, Neural networks, Mean square error methods, Stress, Strain, Soil

I. INTRODUCTION

Artificial Neural Network (ANN) methods are capable of mapping non-linear relationships between the input and output datasets by involving several sets of hidden layers and weights between them [1][2]. In essence, a neural network is part of a data processing network, which is composed of layered nodes, weights and activation functions. Data is first imported into the network by input nodes, then run through nodes in hidden layers, and finally output by output nodes. Nodes in hidden layers provide non-linear transfer to data in nodes by activation functions. Weights define the contribution of nodes to the next linked nodes. These weights are actually similar to the weights in other network science, like complex networks [3]-[5]. One of the important applications of an ANN is to fit multidimensional datasets in data processing, instead of using numerous mathematical derivations to explore general rules between the input and output datasets [6]-[8]. The general rules are expressed in weights between nodes in the ANN, which can be generally optimized by the error back propagation method. The cost function defines the difference between the output data and the target data. The optimization process actually involves minimizing the cost function by adjusting weights. At present, there are primarily three types of cost functions to evaluate the error in output data during back propagation: (i) quadratic cost function [9,10] which is popular in function regression, (ii) cross-entropy cost function [11], and (iii) softmax loss function [12]. The latter two are widely applied in image recognition and classification.
The quadratic cost function was initially proposed by reference [8] and has been employed extensively during the past three decades [10]. It is extensively used in data regression analysis because of the least squares principle. One shortcoming of ANNs with a quadratic cost function is the low rate of descent in back propagation [13], which requires extensive training time in the ANN model, in particular for big data. This problem is even more acute for deep neural networks, such as deep long-term memory recurrent neural networks and deep convolutional networks. Furthermore, using a quadratic cost function may lead to variations in performance in data processing, meaning that the precision is lower for smaller data in the same dataset (details will be discussed in Section 4). The issue also exists in other applications of neural networks, such as speech recognition and classification [14]-[16].

This study aims to investigate the performance of neural networks with quadratic relative error cost function in data regression. First, the reasons for the low rate of weight descent and variations in performance in multi-magnitude data processing are discussed in terms of cost function. Then, the characteristics of the quadratic relative cost function are analyzed in comparison with the quadratic cost function. Finally, the performances of the two cost functions are discussed using four numerical examples.

II. BRIEF REVIEW OF QUADRATIC COST FUNCTION

The cost function determines the descent amount of the weight in each step of the gradient descent algorithm. The quadratic cost function (C) is defined as the mean squared error (MSE) between the output data and the target data by (1). For simplicity, the target data is assumed to be one-dimensional here.

\[
C = \frac{1}{m} \sum (y_i - \hat{y}_i)^2 / 2 = \frac{1}{m} \sum C_i \tag{1}
\]

where \( m \) is the number of samples, \( y_i \) is the output of the \( i \)th sample in the ANN, \( \hat{y}_i \) is the target data of the \( i \)th sample, \( C_i \) is the mean squared error of the output data and the target data of the \( i \)th sample.

A. WEIGHT UPDATE

The ANN repeatedly adjusts the weights of the network by back propagating errors based on the chain rule to minimize the difference between the output \( y_\text{out} \) and the target data \( y_t \) [9]. It is well-known that an ANN with the quadratic cost function using error back propagation is slow [13]. Generally, the weight adjustment in the ANN employs regularization and learning rate adjustments based on (2).

\[
W = W - \frac{\partial C}{\partial W} \tag{2}
\]

where the term \( \partial C/\partial W \) in the procedure of the back propagating error is expressed by (3).

\[
\frac{\partial L_i}{\partial y_i} = (y_i - y_t)f'(z_i)
\]

\[
\frac{\partial l_i}{\partial W} = \frac{\partial (l + 1)}{\partial (a_i')}f'(z_i')
\]

\[
\frac{\partial C_i}{\partial W} = \delta(l + 1)(a_i')^T
\]

\[
\frac{\partial C}{\partial W} = \sum \frac{\partial C_i}{\partial W}
\]

where \( z'_i \) is the input data of the \( i \)th layer of the \( l \)th sample, \( a_i' \) is the output data of the \( i \)th layer of the \( l \)th sample, \( W \) is the weight of the \( l \)th layer, \( f \) is the active function, \( L \) refers to the output layer, and \( \delta l \) is the error of the \( l \)th layer of the \( i \)th sample. The superscript \( T \) stands for the matrix transposition. As shown in (3), the partial derivative of the cost function for the weight \( W \) depends on the difference between the output \( y_\text{out} \) and target data \( y_t \). As \( y_\text{out} \) approaches \( y_t \), the difference between them, \( (y_\text{out} - y_t) \), approaches 0, which in turn causes the rate of weight adjustment to approach zero.

B. VARIATIONS IN PERFORMANCE IN MULTI-MAGNITUDE DATA

The weight adjustment in the ANN is derived from errors between the output data and target data of all samples. Samples with different magnitudes affect the adjustment rate differently. The magnitude of data in this text refers not only to the order of data, but also to the amount value. The weight adjustments of the output layer for \( i \)th sample are presented in (4).

\[
\frac{\partial C}{\partial W} = \frac{\partial C}{\partial y_i} = (y_i - y_t)f'(z_i)(a_i')^T \tag{4}
\]

Considering \( W_{k-1}^{l-1} \) is the \( k \)th element of the weight \( W^{L-1} \), when the active function is a hyperbolic tangent function, equation (4) becomes:

\[
\frac{\partial C}{\partial W_{k-1}^{l-1}} = (y_i - y_t)(1 - a_k^{l-1})^T \tag{5}
\]

Considering \( z = \sum a_k W_k \), and \( z_k = 1/2\ln(1 + y_{\text{out}}) - 1/2\ln(1 - y_{\text{out}}) \). The weight adjustments of \( W_{k-1}^{l-1} \) can be expressed as:

\[
\frac{\partial C}{\partial W_{k-1}^{l-1}} = 2(1 - y_t)(1 - a_k^{l-1})^T \frac{1}{W_k^{l-1}} \left( \ln \left( \frac{1 + y_\text{out}}{1 - y_\text{out}} \right) - A_k \right) \tag{6}
\]

where \( H \) is the number of nodes in the (L-1)th layer.

It can be found that the adjustment of weight \( W_{k-1}^{l-1} \) is a complex function of \( y_{\text{out}} \). However, we could analyze the main component of the complex function to determine the variation tendency of adjustments of weight \( W_{k-1}^{l-1} \) corresponding to different samples. The main component of the function in (6) is \((y_{\text{out}} - y_t)(1 - a_k^{l-1})\ln((1 + y_{\text{out}})/(1 - y_{\text{out}}))\). Assuming three samples with \( y_1 = 0.9, y_2 = 0.5, \) and \( y_3 = 0.1 \), the main component function is plotted in Fig. 1 with \( y_{\text{out}} = y_{\text{out}} \).
Fig. 1 presents the tendency of samples with larger magnitudes under larger adjustments of weight. The adjustments of weight for small magnitude samples become extremely weak when approaching the target data. In addition to the different weight adjustment rates of different samples, the variations in performance in multi-magnitude data processing in neural networks also includes the varying precision in approximation between the output data and target data [16]. The approximation relative error of large target data tends to be higher than that of small target data.

![Main component function](image)

**Main function**

\[
(f_\text{main})(y) = \sum_{i} w^i \frac{1}{1+y_1} \cdot \left(1 + y_1ight) \cdot \ln \left( \frac{1+y_1}{1-y_1} \right)
\]

**Output data**

\[
y_{oi}^k = y_{i}^{(L)} (a_i \cdot W_k^{L-1})
\]

\[
\frac{\partial C}{\partial W_k^{L-1}} = \frac{1}{y_i^{2}} \left( \frac{y_i - y_{oi}}{y_i} \right) \frac{1}{W_k^{L-1}} \cdot \ln \left( \frac{1+y_i}{1-y_i} \right) - A_i
\]

where the superscript REMSE refers to the corresponding item being based on REMSE cost function, while the superscript MSE refers to the MSE cost function. A neural network using the REMSE cost function generally gains acceleration that is \( A (A \text{ is larger than } 1) \) times that gained by a network using the MSE cost function in the beginning. At each iteration, the weight adjustment to the \( th \) sample of REMSE would gain \( 1/y_{1}^{2} \) times that gained by the weight adjustment to the same sample of MSE. This is why a neural network with the REMSE cost function is much faster than that with the MSE cost function.

**B. PERFORMANCE IN MULTI-MAGNITUDE DATA**

Regarding the variations in performance for multi-magnitude data, the REMSE cost function could improve the performance of the neural network with small magnitude data, as the relative error is applied to measure the approximation precision. Referring to (6), the adjustment of weight the \( W_k^{L-1} \) of the \( th \) sample is given by:

\[
A_i = \sum_{j=1, j \neq i}^{N} a_i W_j^{L-1}
\]

**FIGURE 2. Weight adjustment trend of a neural network using the REMSE cost function for samples of different magnitudes.**

Comparing (8) with (3), we found that the error of each layer in the neural network using the REMSE cost function will be \( 1/y_{1}^{2} \) times that using the MSE cost function. As \( y_{i} \) lies in range of 0 to 1, \( 1/y_{1}^{2} \) is larger than 1 and increases as the target data \( y_{i} \) decreases. However, for samples with \( y_{i} = 0 \), one small number with sufficient precision, such as \( 10^{-3} \) or less, should replace \( y_{i} \) to ensure that the denominator does not become zero. The larger back propagating error of the output layer raises the adjustment rate of weights. The comparison of weights adjustments is presented as follows:

\[
\frac{\partial C^{\text{REMSE}}}{\partial W^i} = \frac{1}{y_i} \sum_{l=1}^{N} (\delta (l+1) (a_i^l)^T) W^{L-1} = A \sum_{l=1}^{N} (\delta (l+1) (a_i^l)^T) W^{L-1}
\]

(9)

The slow weight update and variations in performance in multi-magnitude data are attributed to the employment of the mean squared error (MSE). Another cost function (denoted later in this text as REMSE cost function) is analyzed based on the relative mean squared error (REMSE), which reverses the performance of the neural network for data of different magnitudes.

Corresponding to (1), the REMSE cost function is expressed as:

\[
\frac{\partial C^{\text{REMSE}}}{\partial W^i} = (y_{oi} - y_{i}) (1 - y_{oi}^2) (a_i^{L-1})^T
\]

Then, the first error calculation in back propagation is modified as:

\[
\delta l = \frac{(y_{oi} - y_{i})}{y_i^2} f'(z_i)
\]

Equation (10) illustrates that the weight adjustment of the \( th \) sample in a neural network using the REMSE cost function gains a \( 1/y_{1}^{2} \) times enlargement compared with using that with the MSE cost function. The smaller the target data is, the more the enlargement will be. The main component function in Eq. (10) is plotted in Fig. 2, which presents the
weight adjustment trend. The multi-magnitude data would gain a similar approximation precision. Moreover, the weight adjustment for small magnitude data $y_i$ would be enlarged as $1/y_i^2$, which means that a data sample with a smaller magnitude would get enlarged to a greater degree and make a greater contribution to the weight adjustment. The weight adjustment of samples with small target data would make a greater contribution to the total weight adjustment $\partial C/\partial W$ (see (3)) and have priority in weight adjustment, which means that the cost of a neural network using the REMSE cost function will exhibit a segmented decline (verified in section IV C)

IV. VERIFICATION

A. TWO-NODE-TO-ONE-NODE MODEL

![Two-node-to-one-node model schedule](image)

**FIGURE 3.** Two-node-to-one-node model schedule

Two simple models, called the MSE model and the REMSE model, were built to compare the MSE and REMSE cost functions in MATLAB. The two models were two-node-to-one-node networks (see Fig. 3), which is helpful to analyze the weight adjustment in error back propagation. The two-node-to-one-node model can be treated as the basic unit of complex neural networks. The weight updating by error propagation in this model is actually not only the same as that in complex neural networks, but also simple and convenient for virtualization to understand the differences in performance of the REMSE and MSE cost functions.

![Comparison of dataset 1](image)

**TABLE I.** Dataset Used in Two-Node-To-One-Node Model

<table>
<thead>
<tr>
<th>Dataset 1</th>
<th>Dataset 2</th>
<th>Dataset 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$</td>
<td>$y$</td>
<td>$X_1$</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0.2</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0.4</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0.6</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0.8</td>
</tr>
</tbody>
</table>

MSE error $= \frac{1}{m} \times \sum_{i} (y_m - y_i)^2 / 2$

REMSE error $= \frac{1}{m} \times \sum_{i} \left( \frac{y_m - y_i}{y_i} \right)^2 / 2$

Relative error $= \frac{y_m - y_i}{y_i}$

The hyperbolic tangent function is applied as the active function and the initial weight is defined as a zero vector to start calculation at the same point. No bias item is considered. Table I provides details on the four datasets used in the model. Each dataset contains five samples. For simplicity, $X_1$ and $X_2$ are defined as the same. Three types of mapping relationships are applied to investigate the performance of the REMSE and MSE cost functions.

![Comparison of dataset 2](image)

**FIGURE 4.** Comparison of dataset 1 (a) $\partial C/\partial W$ at each iteration; (b) mean squared error and relative mean squared error at each iteration; (c) relative error at each iteration; (d) training results.
FIGURE 5. Comparison of dataset 2 (a) $\partial C / \partial W$ at each iteration; (b) mean squared error and relative mean squared error at each iteration; (c) relative error at each iteration; (d) training results.

FIGURE 6. Comparison of dataset 3 (a) $\partial C / \partial W$ at each iteration; (b) mean squared error and relative mean squared error at each iteration; (c) relative error at each iteration; (d) training results.
For dataset 1, y = X_1 and every element is the same, which means that there is no difference among samples. For dataset 2, X_s = X_2 = y, the target y in different samples is different, but with the same magnitude. For dataset 3, y = 2X_1, and the target y in different samples shows a greater difference with different magnitude. In back propagation, the L2 regularization is applied and the parameter of L2 regularization is defined as 0.01, while the learning rate is defined as 0.1. The regular iterations for datasets 1 and 2 are initially defined as 300, while the regular iterations for dataset 3 is initially defined as 1000. Iterations may be adjusted in different cases.

All datasets are used in the training of simplified models to check the mean squared error (MSE error) or relative mean squared error (REMSE error) of samples, relative error, and adjustment rate of weight, W, of all samples at each iteration. The MSE error, REMSE error and relative error are defined in (11). The results are presented in Fig. 4 to Fig. 6.

As shown in Fig. 4(a), the weight adjustment rate for each sample is the same because the data in each sample is the same. At the beginning, the adjustment rate of the REMSE model is 5, while that of the MSE model is 1.25. Therefore, the REMSE error in the REMSE model reduces much faster than the MSE error in MSE model, as shown in Fig. 4(b). The REMSE model takes 118 iterations keeping the REMSE error stable at $2.13 \times 10^{-7}$ and 104 iterations keeping the MSE error stable at $2.67 \times 10^{-2}$, while the MSE model takes 300 iterations to arrive at an MSE error of $4.31 \times 10^{-6}$. We further found that the MSE error in the MSE model arrived at $4.1989 \times 10^{-10}$ at the 576th iteration and remained almost unchanged at the 10000th iteration. The relative error of each sample in the REMSE model also reduced faster than that in the MSE model (see Fig. 4(c)). As shown in Fig. 4(d), the mapping relationship in dataset 1 can be fit well by the two-node-to-one-node model. The REMSE model behaves better in terms of precision and time. Specifically, the REMSE model is 5 times quicker than the MSE model in the regression of dataset 1 and also produces a higher approximation precision.

The magnitude of data in dataset 2 is of the same order. Fig. 5(a) shows the obvious effect of the REMSE model on eliminating the gap in the adjustment rate of weight of different samples. In the MSE model, the initial adjustment rates of weight for 5 samples (0.1, 0.2, 0.3, 0.4, 0.5) are -0.01, -0.04, -0.16, -0.36 and -0.64. The difference in the sample magnitude is only 5 times, while the difference in adjustment rate could arrive at 64 times. In the REMSE model, the adjustment rates change to -1, -1, -1, -1, -1, which are at the same magnitude and are much higher in absolute value than those in the MSE model. The REMSE model takes 72 iterations keeping the relative mean square error stable at $7.63 \times 10^{-4}$, and 47 iterations keeping the mean squared error stable at $1.33 \times 10^{-3}$. The MSE model takes 592 iterations keeping the mean squared error stable at $7.0182 \times 10^{-4}$ and remains almost unchanged at the 10000th iteration. The mean squared error of the REMSE model is a little higher than that in the MSE model, which could be attributed to the mapping relation that is beyond the capability of the two-node-to-one-node model. In terms of relative error in Fig. 5(c), the REMSE model effectively reduces the relative error of samples with a small amount of target data and causes a small increase in the relative error of samples with a large amount of target data. The variation is from 0.16, 0.14, 0.09, 0.005, and -0.009 to 0.06, 0.05, 0.004, -0.006, and -0.14. However, the range of reduction is larger than that of increase. Fig. 5(d) illustrates that the REMSE model performs better with small data, while the MSE model performs better with large data.

The maximal order difference of data in dataset 3 arrives at $10^2$, which is much larger than that in dataset 2. Nevertheless, the REMSE model can eliminate the difference in adjustment rates that occurs in the MSE model (see Fig. 6(a)). The adjustment rates are transferred from -0.0002, -0.0032, -0.08, -0.32, -0.5 to -0.5, -0.5, -0.5, -0.5, and -0.5. The convergence rate of the REMSE model is also much faster than the MSE model, as presented in Fig. 6(b). The relative errors between the output data and the target data change from 0.26, 0.26, 0.17, -0.04, and -0.18 in the MSE model to 0.06, 0.06, 0.005, -0.14, and -0.21 in the REMSE model. The reduction of relative error in small data is greater than the increase in large data.

**B. PERFORMANCE IN WEIGHTS SPACE**

![Comparison of neural networks using the REMSE and MSE cost functions in weights space.](image)

In order to verify the effect of the REMSE cost function on the adjustment of weights, the MSE and REMSE cost functions are plotted in Fig. 7 based on one three-layer network (one hidden layer) with one node in each layer and two weights $W_1$ and $W_2$ in total. Dataset 2 is employed here. As shown in Fig. 7, there are much fewer plateaus for the neural network using the REMSE cost function than for that using the MSE cost function. The adjustment rate of weights is much higher with the REMSE cost function, which means a faster convergence rate.
C. THREE-LAYER NEURAL NETWORK

Results from two-node-to-one-node models illustrate the difference in weight adjustment and variations in performance for data of different magnitudes using the MSE and REMSE cost functions. A real three-layer neural network with two nodes in the hidden layer was built to compare the regression performance of the MSE cost function (called the MSE model) and the REMSE cost function (called the REMSE model) in neural networks. The input data X and target data y employed in the neural network result from the one-dimensional quadratic function listed in Table II. The iterations of the neural network are defined as 100 and no regularization rule is applied. The classical sigmoid active function is applied.

The output data and cost of the two models are shown in Fig. 8. The neural networks using two cost functions are seen to have similar function approximation performance, whereas neural networks using the REMSE cost function present a larger variance in training and prediction of larger magnitude data. Fig. 8 (b) presents the faster convergence of neural networks using the REMSE cost function.

Fig. 9 depicts the error comparison of neural networks using MSE and REMSE cost function in mean squared error and relative error. For both mean squared error and relative mean squared error, the neural network using the REMSE cost function shows more approximation for small magnitude data whereas it shows larger variance for large magnitude data in training and test process compared to that using the MSE cost function. In terms of relative mean squared error, the improvement in training and prediction precision for small magnitude data regression is extremely high for the REMSE cost function, whereas the variance enlargement for large magnitude data regression is correspondingly slight.

### TABLE II

<table>
<thead>
<tr>
<th>Number</th>
<th>X</th>
<th>y</th>
<th>Number</th>
<th>X</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1</td>
<td>0.01</td>
<td>9</td>
<td>0.9</td>
<td>0.81</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.04</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0.3</td>
<td>0.09</td>
<td>11</td>
<td>0.08</td>
<td>0.0064</td>
</tr>
<tr>
<td>4</td>
<td>0.4</td>
<td>0.16</td>
<td>12</td>
<td>0.15</td>
<td>0.0225</td>
</tr>
<tr>
<td>5</td>
<td>0.5</td>
<td>0.25</td>
<td>13</td>
<td>0.25</td>
<td>0.0625</td>
</tr>
<tr>
<td>6</td>
<td>0.6</td>
<td>0.36</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.7</td>
<td>0.49</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.8</td>
<td>0.64</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE 8.** Performance of neural networks using the MSE and REMSE cost functions. (a) output data and target data; (b) cost at each iteration.

**FIGURE 9.** Error comparison of neural networks using the MSE and REMSE cost functions. (a) mean squared error; (b) relative mean squared error.

D. LSTM DEEP LEARNING NETWORK

In order to investigate the performance of the REMSE cost function in real tasks, a larger database consisting of deviator stress and axial strain of soil from numerical experiments...
was built and trained using LSTM deep learning networks. Stress and strain are important data in geotechnical engineering, and are used for determining the safety and stability of infrastructure [17]-[22] such as excavations [23,24], tunnels [25]-[29], slopes [30], piles [31], and anchors [32]. The soil test module in Plaxis software was applied to simulate the triaxial compression of soil and output stress and strain data. The constitutive model of soil is assigned as the modified cam-clay model. Parameters of the constitutive model and the simulation procedure are listed and noted in Table III.

In total, 127 sets of stress and strain data with different lengths of time were included in the database; 100 sets were used for training, while the rest were used as the testing set. Each set of stress and strain data presents strong sequential characteristics—the magnitude increases with time. Besides, the maximum value for each set of data is also different in magnitude.

### TABLE III

<table>
<thead>
<tr>
<th>Parameters in Numerical Tests of Soil Using Modified Cam Clay Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>$\lambda$</td>
</tr>
<tr>
<td>0.5</td>
</tr>
<tr>
<td>0.55</td>
</tr>
<tr>
<td>0.6</td>
</tr>
<tr>
<td>0.06</td>
</tr>
<tr>
<td>0.09</td>
</tr>
<tr>
<td>0.75</td>
</tr>
<tr>
<td>0.8</td>
</tr>
<tr>
<td>0.5</td>
</tr>
<tr>
<td>0.55</td>
</tr>
<tr>
<td>0.6</td>
</tr>
<tr>
<td>0.9</td>
</tr>
<tr>
<td>0.7</td>
</tr>
<tr>
<td>0.75</td>
</tr>
<tr>
<td>0.8</td>
</tr>
<tr>
<td>0.1</td>
</tr>
<tr>
<td>0.55</td>
</tr>
<tr>
<td>0.6</td>
</tr>
<tr>
<td>0.75</td>
</tr>
<tr>
<td>0.12</td>
</tr>
<tr>
<td>0.7</td>
</tr>
<tr>
<td>0.8</td>
</tr>
<tr>
<td>0.5</td>
</tr>
<tr>
<td>0.05</td>
</tr>
<tr>
<td>0.15</td>
</tr>
<tr>
<td>0.75</td>
</tr>
</tbody>
</table>

Three sets of LSTM network are developed with the same architecture but using different cost functions, namely MSE, REMSE, and Logcosh which is recognized as an appropriate substitute for MSE in some cases [33]. The equation of the Logcosh cost function is given in (12). The input layer of the LSTM networks contains four nodes consisting of three constitutive model parameters ($e$, $\lambda$, and $p$) and one axial strain data, $\varepsilon$. One layer with 12 LSTM units and one layer with 12 fully connected nodes make up two hidden layers. The output layer has only one node of deviator stress data, $q$. LSTM units used in this case are only composed of input gates, forget gates, and output gates without peepholes, which are considered to have a negligible effect on the final performance of LSTM [34].

$$\log \cosh = \frac{1}{m} \sum_i \log(\cosh(y_i - y_i))$$

The input and output data are preprocessed to 0 to 1 by dividing by the maximum value. Therefore, the original features of stress and strain are still contained in these data. The weights in all LSTM networks are initialized by a random initializer from -0.15 to 0.15 by experience. We pick up the batch gradient descent mode optimized using a conjugate descent algorithm to achieve as accurate a result as possible, although the training time is a little longer than other SGD modes. Each LSTM network is trained for 3000 epochs in total.

The coefficient of determination $R^2$ for all training and testing samples from three different LSTM networks are calculated and depicted in Fig. 10. LSTM networks with MSE and Logcosh cost functions present similar $R^2$ values and performance in both the training set and testing set, as shown in Fig. 10 (a). The $R^2$ values of numerous samples are less than 0.9, and even lower than 0.6 in some cases. The LSTM network with the REMSE cost function
outperformed the other two LSTM networks for nearly all samples with an average $R^2$ greater than 0.90 (see Fig. 10 (b)). The MSE error and REMSE errors of the three LSTM networks are tabulated in Table IV. It can be seen that the MSE error of the LSTM network with the REMSE cost function is a little larger than those of the other LSTM networks, while the REMSE error is much lower. This is consistent with former cases.

### TABLE IV
MSE AND REMSE ERRORS OF TRAINING AND TESTING RESULTS FOR DIFFERENT LSTM MODELS

<table>
<thead>
<tr>
<th>LSTM networks</th>
<th>MSE error</th>
<th>REMSE error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training</td>
<td>Testing</td>
</tr>
<tr>
<td>MSE cost function</td>
<td>0.64</td>
<td>0.45</td>
</tr>
<tr>
<td>Logcosh cost function</td>
<td>0.68</td>
<td>0.47</td>
</tr>
<tr>
<td>REMSE cost function</td>
<td>1.47</td>
<td>0.93</td>
</tr>
</tbody>
</table>

The performances of the three LSTM networks on parts of the training and testing samples are compared in Fig. 11. It can be seen that the REMSE cost function dramatically improves the performance of LSTM on samples with small magnitudes. LSTM networks with MSE and Logcosh not only failed to learn all the features, but also produced oscillation during the initial phase. Fig. 12 records the development of the normalized cost function $J$ and REMSE error during the training process for all LSTM networks. From Fig. 12 (a), the convergence rate of LSTM networks with MSE and Logcosh is rapid at the beginning, but slows down rapidly after that. In comparison, although the LSTM network with the REMSE cost function is initially slightly slower than the other two LSTM networks, it exhibits a faster whole convergence rate. As shown in Fig. 12 (b), the REMSE cost function drives LSTM network convergence faster to the lowest REMSE error. However, MSE and Logcosh failed to decrease the REMSE error of the LSTM networks.
V. DISCUSSION

For the REMSE cost function, the range of $y$ should be from 0 to 1. When $y=0$, the reciprocal of $y$ tends to infinity. Two possible methods are provided when using the REMSE cost function if $y=0$. One method is to replace zero with a small value, such as $1 \times 10^{-3}$ or less, and the other method is to locally apply the MSE cost function at the point $y=0$.

The quadratic relative cost function (REMSE cost function) increases the weight adjustment rate to approach the target data at the same learning rate as that of the MSE cost function. However, the large learning rate may cause numerical oscillation. When a reducing learning rate is applied to avoid numerical oscillation, the acceleration of weight adjustment from the REMSE cost function might be weakened. Even so, using the advanced gradient algorithm with varying learning rates might avoid numerical oscillation while enhancing the adjustment rate and approximation precision.

The characteristic of better approximation to small magnitude data in neural networks using the REMSE cost function is not always advantageous. If the target dataset contains larger magnitude data, neural networks using the REMSE cost function would reduce the precision of output data and seems worse than that using MSE cost function. However, when the dataset contains more small magnitude data or small magnitude data attracts more attention, neural networks using REMSE cost function would work better and should get preferential recommendation.

VI. CONCLUSION

This paper investigates the performance of neural networks using the quadratic relative error cost function (REMSE cost function). Neural networks using the REMSE cost function exhibits a regression capability similar to that of networks using the MSE cost function. Neural networks using the REMSE cost function possess advantages in terms of the adjustment rate of weights and present variations in performance in multi-magnitude data regression. The approximation to the target data of the neural network using the REMSE cost function shows a higher convergence rate and higher precision in small magnitude data regression. Therefore, the application of the REMSE cost function in neural network is recommended for datasets with a greater fraction of small magnitude data. Besides, small batch sizes in large datasets can be treated as small datasets, and the results can be optimized using this technique.

ACKNOWLEDGMENT

The financial support by the National Natural Science Foundation of China (NSFC) (973 Program: 2015CB057806) and the Innovative Research Funding of the Science and Technology Commission of Shanghai Municipality (Grant No. 18DZ1201102) are gratefully acknowledged.

REFERENCES


FIGURE 12. Training curves of three LSTM networks (a) normalized cost function $J$ (b) REMSE error
