Computer Technology and Application 7 (2016) 73-82
doi: 10.17265/1934-7332/2016.02.002

Neural Network Inversion for Multilayer Quaternion Neural Networks

Takehiko Ogawa

Department of Electronics and Computer Systems, Faculty of Engineering, Takushoku University, Tokyo 193-0985, Japan

Abstract: Recently, solutions to inverse problems have been required in various engineering fields. The neural network inversion method has been studied as one of the neural network-based solutions. On the other hand, the extension of the neural network to a higher-dimensional domain, e.g., complex-value or quaternion, has been proposed, and a number of higher-dimensional neural network models have been proposed. Using the quaternion, we have the advantage of expressing 3D (three-dimensional) object attitudes easily. In the quaternion domain, we can define inverse problems where the cause and the result are expressed by the quaternion. In this paper, we extend the neural network inversion method to the quaternion domain. Further, we provide the results of the computer experiments to demonstrate the process and effectiveness of our method.

Key words: Inverse problems, neural network inversion, quaternion, inverse mapping, inverse kinematics.

1. Introduction

Inverse problems determine the inner mechanisms or causes of an observed phenomenon. The solutions to inverse problems have been required in various engineering fields [1, 2]. The network inversion method is a neural network-based solution [3] that has been studied in order to apply the solution to the inverse problems of image restoration, inverse kinematics, and so on [4-6].

Recently, artificial neural network models and their training methods for handling high-dimensional numbers that include complex numbers or quaternions have been proposed [7-11]. High dimensional neural networks are able to learn and estimate the relationship between high dimension inputs and outputs. Given that the quaternions easily express the geometrical relationships in the 3D (three-dimensional) space, they are used in the fields of computer graphics, robot control, and so on.

Inverse problems whose causes and results extend to the high dimensional domain can arise in various engineering fields. For example, these problems are included in the image restoration of the frequency domain, the inverse estimation of the attitude change of objects in the 3D space, and more. The adaptive solution of learning with neural networks to solve inverse problems must be extended to the high dimensional domain. Presently, an extension to the complex domain of network inversion has been proposed [12, 13].

In this paper, we propose an extension of the neural network inversion method to the quaternion for solving inverse problems in the quaternion domain. We performed simulations of the inverse mapping problem in the 3D space, and applied a procedure of the proposed quaternion network inversion method to the inverse kinematics problem of robot arms. Moreover, we performed a comparison with a usual real-value network inversion to demonstrate the effectiveness of the proposed quaternion network inversion method.

The rest of this paper is organized as follows: Section 2 discusses inverse problems and neural networks; Section 3 introduces QNNs (quaternion neural networks); Section 4 presents computer experiments; Section 5 provides our conclusions.

Corresponding author: Takehiko Ogawa, Dr. (Eng.), research field: artificial neural networks and their application.

2. Inverse Problems and Neural Networks

An inverse problem is an estimation of the cause or internal mechanism that produces a given phenomenon from a set of observed phenomena. A direct problem leads to the results from a cause; in contrast, an inverse problem finds a cause from the results, or estimates an input from the output. Inverse problems have the disadvantage that the existence, uniqueness, and stability of a solution are not guaranteed in general; for this reason, inverse problems are referred to as ill-posed inverse problems [1]. Though ill-posed inverse problems are often referred to simply as inverse problems, in this paper, we refer to the inverse problem as such that which estimates a cause from the result, regardless of its ill-posedness.

2.1 Network Inversion

A multilayer neural network usually learns an input-output relationship from the direction of the input to the output. In addition, it estimates the output from the given input using the learned relationship in the forward direction. Therefore, such neural network is suitable for a solution of direct problems. In general, the error back-propagation method based on the gradient method is used for learning.

Network inversion is a method to estimate the input from the output using a learned network [3]. Essentially, by repeating modifications based on the gradient method, the input is estimated from the output. In an actual case of solving the inverse problem by a network inversion, a two-step process is utilized, where forward modeling occurs during the learning phase (the first step), and inverse estimation occurs during the inverse use of the forward model (the second step). In the learning phase, forward modeling is performed by the error back-propagation method using tutorial data, similar to the usual learning of the multi-layer neural network. In the inverse estimation phase, the inverse problem is solved using the forward model to estimate the input from the output of the network.

The principle of inverse estimation by network inversion is an iterative minimization of the output error based on the gradient method, similar to the principle of learning by the error back-propagation method. In the error back-propagation method, the output error is assumed to be caused by errors in the weights; to correct the weights, the error is minimized. In the inverse estimation method using network inversion, it is assumed that the weights obtained by learning are correct, and the input is to be corrected by minimizing the output errors.

In the learning phase of the usual error back-propagation method, we feed the tutorial input x, calculate the output y, provide the tutorial output y', and calculate the output error E. Then, the weights w are corrected to minimize the output error E as

$$w^{new} = w^{old} - \varepsilon_t \frac{\partial E}{\partial w} \tag{1}$$

where ε_t is the learning coefficient. The input/output relationship is composed in the multilayer networks that completed the learning. In other words, the output error is assumed to be caused by an error of the weight in the learning.

In the inverse estimation phase, we can correct the input x instead of the weights w to minimize the output error E, based on the duality of the weights w and the input x. By fixing the weights obtained in the learning, the input x is iteratively updated from the provided output y', based on the gradient method, as

$$x^{new} = x^{old} - \varepsilon_e \frac{\partial E}{\partial x} \tag{2}$$

where ε_e is the estimation coefficient. When the output error is substantially reduced by iterative modifications of the input, we obtain the input that corresponds to the given output. The learned relationship is inversely used in the network inversion. In other words, the output error is assumed to be caused by an error of the input in the inverse estimation. This is the principle of inverse estimation of the network inversion.

3. QNNs (Quaternion Neural Networks)

Quaternion, which is one of the high dimensional numbers, was devised by Refs. [14, 15]. The quaternion is a high dimensional number with a real part and three imaginary parts, and is a system that expresses the characteristics of the 3D space efficiently. Therefore, the quaternion is known widely, because it is used for the attitude control satellite, the imaging of the solid model in computer graphics, and more [16, 17].

Complex-valued network inversion [12] uses a multilayer neural network that includes complex weights and complex neurons. In this method, the complex-valued neural network estimates a complex input from a complex output using a trained network. It is an extension of the input correction principle of a usual network inversion to the complex domain.

The quaternion is expressed as $\boldsymbol{x} = x_1 + ix_2 + jx_3 + kx_4$, where i, j, and k are three imaginary units. The numbers x_1, x_2, x_3, and x_4, which are real numbers, represent each component of the quaternion. In addition, the imaginary units satisfy the relationships: $i^2 = j^2 = k^2 = ijk = 1$, $ij = -ji = k$, $jk = -kj = i$, and $ki = -ik = j$. Quaternions satisfy the associative law of multiplication and the distributive law of addition. However, quaternions do not satisfy the commutative law of multiplication. In particular, the quaternion whose real part is zero is called a pure imaginary quaternion, and it is expressed as $\boldsymbol{x} = ix_2 + jx_3 + kx_4$.

Various neural network models that were extended to the quaternion, such as single neuron, multilayered, and recurrent type have been proposed and studied [9-11]. Using quaternion neurons and weights, multilayer QNNs can learn the relationship between inputs and outputs that is extended to the quaternion. In this paper, we consider a solution to the inverse problem using the multilayer QNN.

3.1 Quaternion Network Inversion

In this paper, we use a multi-layer neural network that extends to the quaternion input, output, and weights. We propose a quaternion network inversion to solve an inverse problem using a network that has completed forward learning. This method estimates the quaternion input that corresponds to the given quaternion output using a trained multilayer QNN. The input modification principle of this method is an extension of the quaternion area of the network inversion. In addition, we use a neuron to apply the sigmoid function independently to each part of the quaternion input. That is, we consider the three-layer network that uses the sigmoid function expressed as

$$\boldsymbol{f}(\boldsymbol{s}) = f(s_1) + if(s_2) + jf(s_3) + kf(s_4),$$

$$f(u) = \frac{1-e^{-u}}{1+e^{-u}} \tag{3}$$

for the hidden layer and the output layer, where $\boldsymbol{s} = s_1 + is_2 + js_3 + ks_4$ represents the weighted sum of the inputs. The schematic view of the network is shown in Fig. 1.

In the learning phase, the network iteratively updates the quaternion weights from a given quaternion learning input and output data, based on the error back-propagation method that has been extended to the quaternion. The output error function is defined as

$$E = \frac{1}{2}\sum_n \sum_r (\boldsymbol{d}_{nr} - \boldsymbol{y}_{nr}) \cdot \overline{(\boldsymbol{d}_{nr} - \boldsymbol{y}_{nr})} \tag{4}$$

where $\boldsymbol{d}_{nr}$ and $\boldsymbol{y}_{nr}$ are the quaternion learning output and the quaternion network output that are provided to the r-th element of the n-th training data, respectively. In addition, the error signal $\boldsymbol{\delta}_{nr}$ from the r-th output element and the error signal $\boldsymbol{\delta}_{nq}$ from the q-th hidden element can be calculated as

$$\begin{aligned} \boldsymbol{\delta}_{nr} &= (\boldsymbol{d}_{nr} - \boldsymbol{y}_{nr}) \cdot (\mathbf{1} - \boldsymbol{y}_{nr}) \cdot (\mathbf{1} + \boldsymbol{y}_{nr}), \\ \boldsymbol{\delta}_{nq} &= (\mathbf{1} - \boldsymbol{v}_{nq}) \cdot (\mathbf{1} + \boldsymbol{v}_{nq}) \cdot \sum_r \boldsymbol{\delta}_{nr} \cdot \overline{\boldsymbol{w}_{qr}}, \end{aligned} \tag{5}$$

where $\boldsymbol{v}_{nq}$ and $\boldsymbol{w}_{qr}$ are the output from q-th hidden element of the n-th training data and the weight from the q-th hidden element to the r-th output element, respectively. We represent the multiplication of each element of the quaternion by the symbol "$\cdot$".

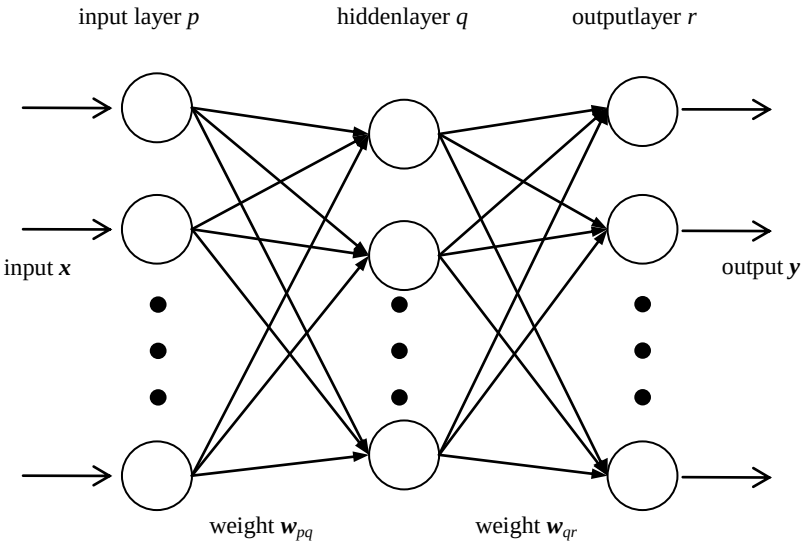

Fig. 1 Example of a three-layer QNN.

Furthermore, we express the quaternion whose every element is one by the symbol "**1**". Based on the error signals, the weight $\boldsymbol{w}_{qr}$ between the hidden and the output elements, and the weight $\boldsymbol{w}_{pq}$ between the input and the hidden elements are updated by

$$\boldsymbol{w}_{qr}^{new} = \boldsymbol{w}_{qr}^{old} - \varepsilon_t \sum_n \boldsymbol{\delta}_{nr} \cdot \overline{\boldsymbol{v}_{nq}},$$

$$\boldsymbol{w}_{pq}^{new} = \boldsymbol{w}_{pq}^{old} - \varepsilon_t \sum_n \boldsymbol{\delta}_{nq} \cdot \overline{x_{np}}, \tag{6}$$

where ε_t is the learning coefficient. By repeating this updating procedure, the quaternion weights distribution to compose the input-output relationship of the given learning data is obtained.

In the inverse estimation phase, we provide a tentative quaternion input while fixing the relationship obtained by learning. Then, we calculate the output error function from the output obtained from the quaternion, which is defined as

$$E = \frac{1}{2} \sum_r (\boldsymbol{d}_r - \boldsymbol{y}_r) \cdot \overline{(\boldsymbol{d}_r - \boldsymbol{y}_r)}, \tag{7}$$

where $\boldsymbol{d}_r$ and $\boldsymbol{y}_r$ are the quaternion learning output and the quaternion network output that are provided to the r-th element of test data, respectively. The error signal $\boldsymbol{\delta}_r$ from r-th output element and the error signal $\boldsymbol{\delta}_q$ from the q-th hidden element can be calculated as

$$\boldsymbol{\delta}_r = (\boldsymbol{d}_r - \boldsymbol{y}_r) \cdot (\mathbf{1} - \boldsymbol{y}_r) \cdot (\mathbf{1} + \boldsymbol{y}_r),$$

$$\boldsymbol{\delta}_q = (\mathbf{1} - \boldsymbol{v}_q) \cdot (\mathbf{1} + \boldsymbol{v}_q) \cdot \sum_r \boldsymbol{\delta}_r \cdot \overline{\boldsymbol{w}_{qr}}, \tag{8}$$

where $\boldsymbol{v}_q$ and $\boldsymbol{w}_{qr}$ are the output from the q-th hidden element and the weight from the q-th hidden element to the r-th output element, respectively. The amount of input correction $\boldsymbol{\delta}_p$ for the p-th input element is expressed as

$$\boldsymbol{\delta}_p = \sum_q \boldsymbol{\delta}_q \cdot \overline{\boldsymbol{w}_{pq}}, \tag{9}$$

where $\boldsymbol{w}_{pq}$ is the weight from the p-th input element to the q-th hidden element. Based on the amount of input correction, the input $\boldsymbol{x}_p$ is updated by

$$\boldsymbol{x}_p^{new} = \boldsymbol{x}_p^{old} - \varepsilon_e \boldsymbol{\delta}_p, \tag{10}$$

where ε_e is the coefficient of input correction. By repeating this updating procedure, we can estimate the quaternion input from the given quaternion output, using the quaternions weight distribution obtained in the learning phase.

4. Computer Experiments

In this study, we handle the inverse estimation problem of the 3D mapping and the inverse kinematics problem of the 2-DOF (two-degree-of-freedom) robotic arm in the 3D space to demonstrate the operation of the proposed method.

In order to study the effect of the proposed method, we provide learning data with different dimension or sparseness to examine the characteristics of the learning and inverse estimation. In addition, we compare the quaternion network with a real network in each problem in order to show the effectiveness of the proposed method.

4.1 Inverse Mapping Problem

We address the inverse estimation problem of mapping the 3D space as an example of a simple inverse problem in the quaternion domain. First, the QNN learns the mapping provided to the points in the 3D space. Then, the network estimates the inverse mapping of the points given in the 3D space using the forward relationship obtained in the learning phase. Here, we perform computer simulations in three types of transformation (rotation, translation, and scaling), three types of settlement of the learning points (on a line, on a plane, and in the 3D grid), and two types of network (the usual real-value neural network, NN, and the QNN). As a consequence, we examine 18 types of experiments.

We consider the three types of transformation (rotation, translation, and scaling) as follows:

- Rotation: $\pi/12$ and $\pi/6$ rotations around the x_2 and x_3 axes;
- Translation: 0.1, 0.2, and 0.3 translations in the direction of the x_2, x_3, and x_4 axes;
- Scaling: 0.6, 1.0, and 1.4 times scaling in the direction of the x_2, x_3, and x_4 axes.

For each transform, the network learns the relationship between the data before and after the transformation, and estimates the input that corresponds to the given output.

Then, we consider the three types of preparation of the learning data (on a line, on a plane, and in the 3D grid) as follows:

- On a line: 15 points that satisfy $x_1 = 0.0$, $x_2 = x_3 = x_4 = k$, $k = \{-0.7, -0.6, \ldots, 0.7\}$;
- On a plane: 25 points that satisfy $x_1 = x_2 = 0.0$, $x_3 = k_1$, $x_4 = k_2$, $k_1 = \{-0.6, -0.3, \ldots, 0.6\}$, $k_2 = \{-0.6, -0.3, \ldots, 0.6\}$;
- In the 3D grid: 125 points that satisfy $x_1 = 0.0$, $x_2 = k_1$, $x_3 = k_2$, $x_4 = k_3$, $k_1 = \{-0.6, -0.3, \ldots, 0.6\}$, $k_2 = \{-0.6, -0.3, \ldots, 0.6\}$, $k_3 = \{-0.6, -0.3, \ldots, 0.6\}$.

We use the previous data as the network input, and use the points that are transformed by each transformation as the corresponding network output. As an example, the three types of learning data used in the simulation of the rotation are shown in Fig. 2.

In addition, we consider the two types of networks, the NN and the QNN, to compare the effectiveness of the proposed method. The architecture of the former is four inputs and four outputs, and of the latter it is one input and one output. The parameters of the networks are shown in Table 1.

As the estimation data, we use the points that are distributed on an ellipsoid, which is described as

$$\frac{x_2^2}{r_2^2}+\frac{x_3^2}{r_3^2}+\frac{x_4^2}{r_4^2}=1, \tag{11}$$

where $r_2 = 0.4$, $r_3 = 0.4$, $r_4 = 0.2$, as shown in Fig. 3. Here, we describe x_2, x_3, and x_4 in the polar coordinate, and obtain the 144 points that satisfy the arguments $\theta_1 = \{0, \pi/6, \ldots, 11\pi/6\}$ and $\theta_2 = \{0, \pi/6, \ldots, 11\pi/6\}$. We use the data as the outputs. In addition, we use the data that are inverse-transformed by the previous transformations as the network input. The network estimates the corresponding input through the provided output.

For the learning phase in each case, we confirm the convergence of errors and the completion of the learning phase. The mean squared errors between the inverse-estimated inputs and the correct values are listed in Table 2 and shown in Fig. 4. First, we consider the results in each transformation. According to the results, the error is reduced with increased dimensions of the learning data in any transformation. Comparing the QNN and the NN, there is no much difference in the learning data in the 3D grid.

However, for the learning data on a line and on a plane, the errors by the QNN are smaller than the

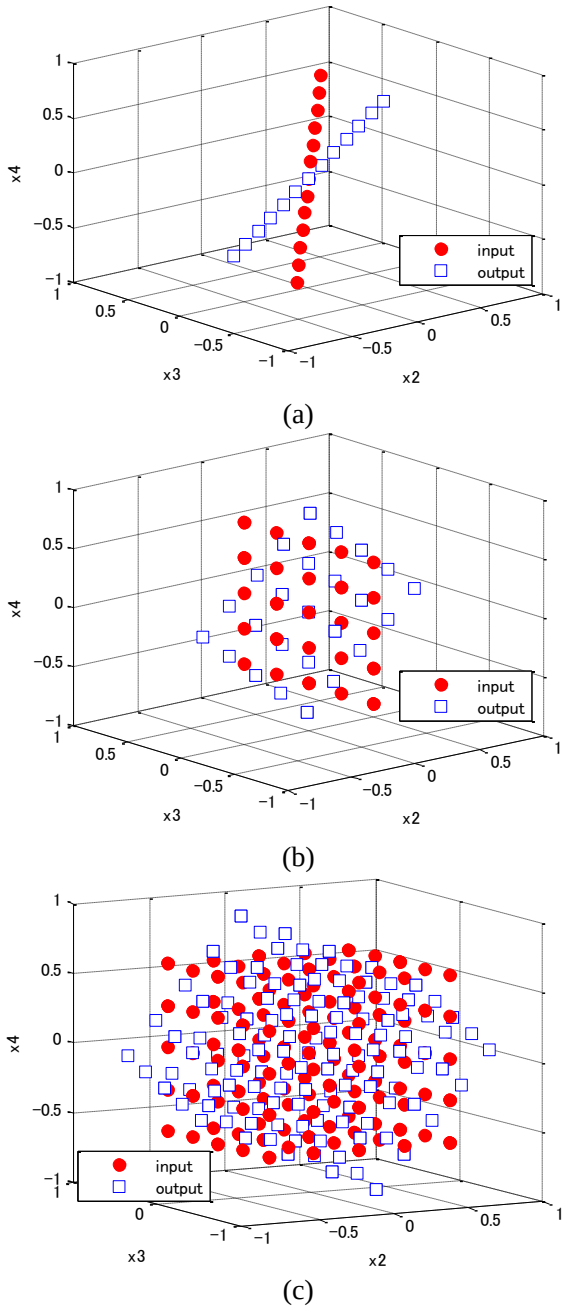

Fig. 2 Learning data used in the simulation (a) on a line; (b) on a plane; and (c) in the 3D grid.

errors by the NN. We consider that the QNN is able to learn the transforms in the 3D space sufficiently, even when given lower dimensional training data. Although the NN is able to learn the transforms with learning data of sufficient dimension, it cannot learn the transforms with learning data of lower dimension.

We consider that the QNN realized structurally the quaternion-based transforms by the quaternion-valued

Table 1 Network parameters for the inverse mapping simulation.

Parameters	Values	
	NN	QNN
Number of input neurons	4	1
Number of hidden neurons	60	15
Number of output neurons	4	1
Training rate ε_t	0.0005	0.0005
Input correcting rate ε_e	0.001	0.001
Max. number of training epoch	20000	20000
Training error to be attained	0.0001	0.0001
Number of estimating epoch	5000	5000

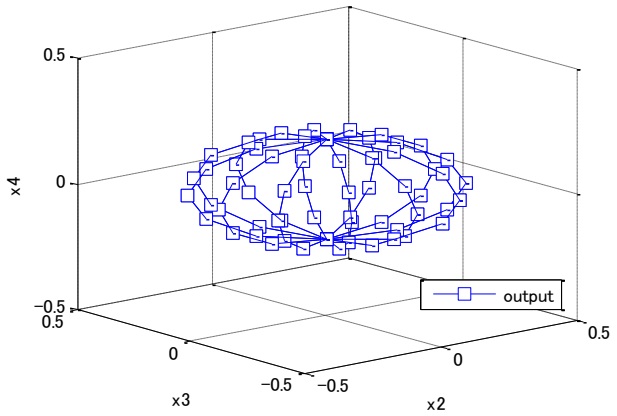

Fig. 3 Ellipsoid output data provided to estimate inputs.

weights, and is able to acquire the transforms from the sparse training data. From the results, we confirm inverse estimation by the proposed method, and show the effectiveness of the proposed method with the transforms in the 3D space.

4.2 Inverse Kinematics Problem

It is considered that the problem of calculating joint angles with respect to the end-effector coordinates of the robot arm is an inverse problem. Such inverse problem expressed by the quaternion can be composed in the 3D space. In this paper, we consider the estimation of the joint angles from the end-effector coordinates of the 2-DOF robotic arm that operates in the 3D space, which is shown in Fig. 5. In this problem, the QNN learns the relationship between the two input joint angles and the corresponding end-effector coordinates output. Then, it estimates the two input joint angles with respect to the arbitrarily

Table 2 Estimated network input error in each inverse mapping simulation.

Learning points	(1) rotation		(2) translation		(3) scaling	
	NN	QNN	NN	QNN	NN	QNN
(a) on a line	0.3561	0.1145	0.6777	0.0776	0.7011	0.1897
(b) on a plane	0.3186	0.1008	0.6677	0.0611	0.1787	0.1319
(c) in the 3D grid	0.0510	0.0405	0.0365	0.0375	0.0951	0.0322

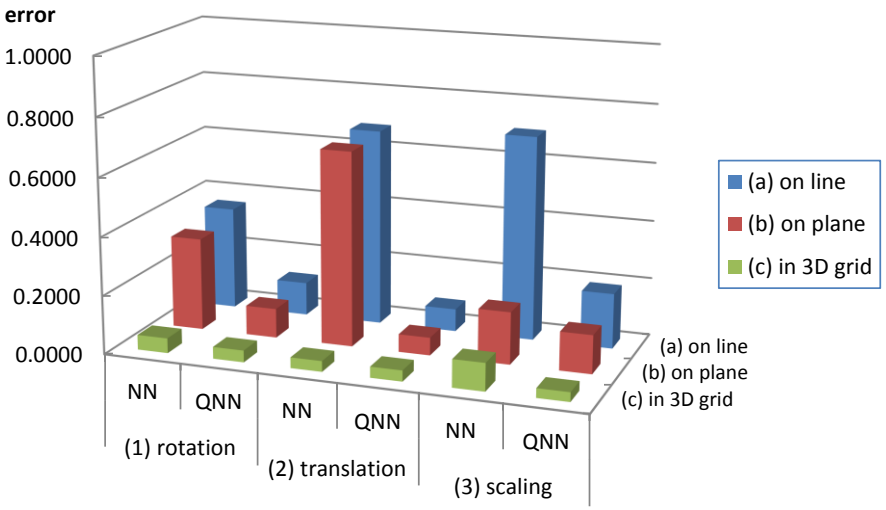

Fig. 4 Plot of estimated network input error in each inverse mapping simulation.

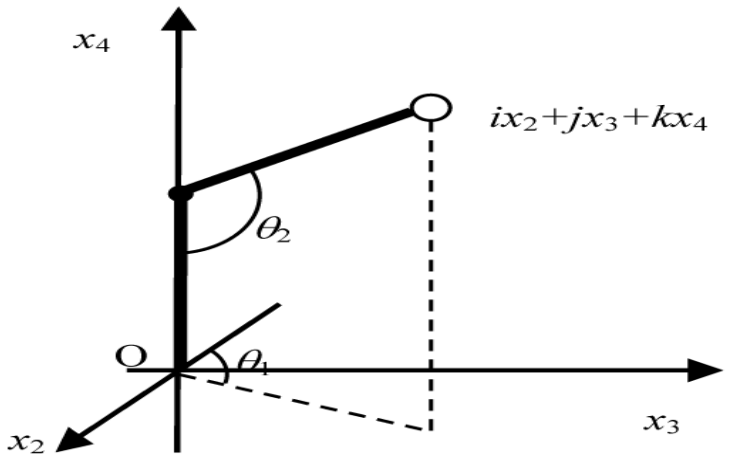

Fig. 5 Two-degree-of-freedom arm in the 3D space.

given end-effector coordinates output, inversely using the forward relationship obtained in learning. In order to confirm the inverse estimation and the effectiveness of the proposed method, we perform six simulations through three types of learning data (the interval of the joint angles π/3, π/6, and π/12) and two types of networks (NN and QNN).

First, we prepare the three types of learning data by setting the joint angles as follows:

- π/3 interval: 10 data that satisfy θ_1 = {0, π/3, ..., π}, θ_2 = {0, π/3, ..., π}(except (θ_1, θ_2) = (0, π/3), … , (0, π), (π, 0), …, (π, 2π/3));

- $\pi/6$ interval: 37 data that satisfy $\theta_1 = \{0, \pi/6, ..., \pi\}$, $\theta_2 = \{0, \pi/6, ..., \pi\}$(except $(\theta_1, \theta_2) = (0, \pi/6), \dots, (0, \pi), (\pi, 0), \dots, (\pi, 5\pi/6)$);
- $\pi/12$ interval: 145 data that satisfy $\theta_1 = \{0, \pi/12, ..., \pi\}$, $\theta_2 = \{0, \pi/12, ..., \pi\}$(except $(\theta_1, \theta_2) = (0, \pi/12), \dots, (0, \pi), (\pi, 0), \dots, (\pi, 11\pi/12)$).

We set the previous angles of (θ_1, θ_2) as the input coordinates on the unit circle in the complex plane, and calculate the output of the end-effector coordinates. As the estimation data, we use the 36 data of $(\theta_1, \theta_2) = (0, 0), (\pi/36, \pi/36), \dots, (\pi, \pi)$. The end-effector coordinates for the learning data of the $\pi/6$ interval and for the estimation data are shown in Fig. 6.

In addition, we consider the two types of the network (the NN and the QNN), to compare the effectiveness of the proposed method. The architecture of the former is eight inputs and four outputs, and of the latter it is two inputs and one output. The parameters of the networks are listed in Table 3.

For the learning phase in each case, we confirm the convergence of errors and the completion of the learning phase. The mean squared errors between the inverse-estimated inputs and the correct values are listed in Table 4 and shown in Fig. 7.

According to the results, we find that the angular interval of the learning data becomes smaller when the estimation error is reduced in the QNN. In addition, we find that the estimation errors in the QNN become smaller than the NN. We consider that the QNN has learned the relationship of the coordinates from the

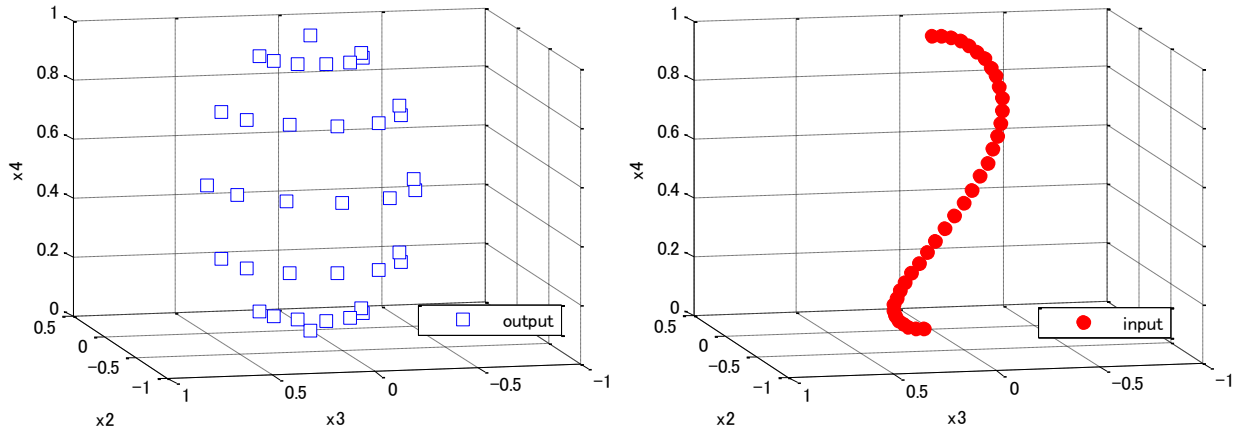

Fig. 6 End-effector coordinates (a) for learning data of $\pi/6$ interval and (b) for estimation data.

Table 3 Network parameters for inverse kinematics simulation.

Parameters		Values	
		NN	QNN
Number of input neurons		8	2
Number of hidden neurons		60	15
Number of output neurons		4	1
Training rate ε_t	Case of $\pi/3$ and $\pi/6$ interval data	0.001	0.001
	Case of $\pi/12$ interval data	0.0001	0.0001
Input correcting rate ε_e		0.001	0.001
Max. number of training epoch		20000	20000
Training error to be attained		0.0001	0.0001
Number of estimating epoch		10000	10000

Table 4 Estimated network input error in each inverse kinematics simulation.

Learning points (interval of angle)	Inverse kinematics	
	NN	QNN
(a) 60 deg.	0.5245	0.4290
(b) 30 deg.	0.6098	0.2704
(c) 15 deg.	0.3262	0.1567

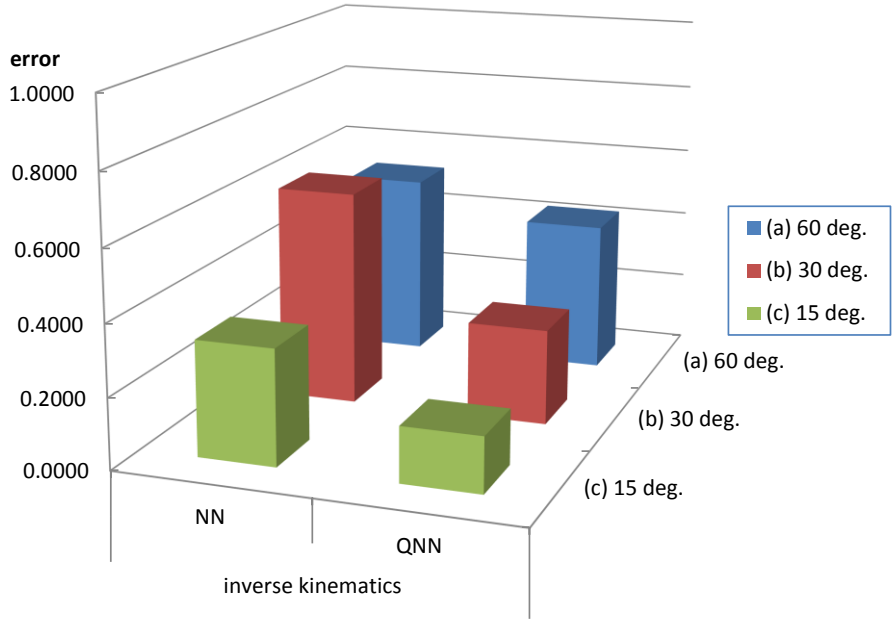

Fig. 7 Plot of estimated network input error in each inverse kinematics simulation.

angle sufficiently, even from the sparse training data, similarly to the simulation of the inverse mapping. Thus, the estimated error is slightly larger, but it is considered that the QNN estimates the joint angles that correspond to the given end-effector coordinates. However, the accuracy of the estimated results is still insufficient; therefore, further examination is required. We consider devising encoding for the joint angles and examining the parameters of the network as an improvement for future work.

5. Conclusions

In this paper, we proposed a quaternion network inversion method for the inverse problem solution that has been extended to the quaternion domain. Our simulation of the inverse mapping problem in the 3D space and the inverse kinematics problem of the robot arm demonstrated inverse estimation by the proposed method. Moreover, we showed the effectiveness of the quaternion network inversion by comparing our proposed method with the usual real-valued neural network. In the inverse estimation from learning by sparse learning data, we verified that the quaternion network inversion was more effective than the usual real-valued network. We consider that this occurs because the quaternion transforms are structurally realized in the quaternion network. As future work, it is necessary to consider a regularization method for the solution of ill-posed inverse problems and an application to actual problems.

Acknowledgment

This work was partly supported by a Grant-in-Aid for Scientific Research (#26330284) from the Japan

Society for the Promotion of Science.

References

[1] Groetsch, C. W. 1993. "Inverse Problems in the Mathematical Sciences." *Informatica International*.

[2] Neto, F. D. M., and da Silva Neto, A. J. 2011. *An Introduction to Inverse Problems with Applications*. Springer.

[3] Linden, A., and Kindermann, J. 1989. "Inversion of Multilayer Nets." In *Proceedings of the International Joint Conference on Neural Networks*, 425-30. Washington, D.C.

[4] Valova, I., Kameyama, K., and Kosugi, Y. 1995. "Image Decomposition by Answer-in-Weights Neural Network." *IEICE Transactions on Information and Systems* E78-D (9): 1221-4.

[5] Lu, B., and Ito, K. 1995. "Regularization of Inverse Kinematics for Redundant Manipulators Using Neural Network Inversions." In *Proceedings of IEEE International Conference on Neural Networks*, 2726-31. Perth, WA.

[6] Murray, W. R., Heg, C. T., and Pohlhammer, C. M. 1993. "Iterative Inversion of a Neural Network for Estimating the Location of a Planar Object." *Proceedings of the World Congress on Neural Networks* 3: 188-93.

[7] Hirose, A. 2006. *Complex-Valued Neural Networks.* Springer.

[8] Nitta, T. 2009. *Complex-Valued Neural Networks: Utilizing High-Dimensional Parameters*. IGI-Global.

[9] Nitta, T. 1996. "An Extension of the Back-Propagation Algorithm to Quaternions." *Proceedings of the International Conference on Neural Information Processing* 1: 247-50. Hong Kong.

[10] Matsui, N., Isokawa, T., Kusamichi, H., Peper, F., and Nishimura, H. 2004. "Quaternion Neural Network with Geometrical Operators." *Journal of Intelligent and Fuzzy Systems* 15 (3-4): 149-64.

[11] Kuroe, Y. 2011. "Models of Clifford Recurrent Neural Networks and Their Dynamics." *Proceedings of the 2011 International Joint Conference on Neural Networks*, 1035-41. San Jose.

[12] Ogawa, T. 2009. "Complex-Valued Neural Network and Inverse Problems." In *Complex-Valued Neural Networks: Utilizing High-Dimensional Parameters*, edited by T. Nitta. IGI-Global, Chapter 2, 27-55.

[13] Fukami, S., Ogawa, T., and Kanada, H. 2008. "Regularization for Complex-Valued Network Inversion." *Proceedings of the SICE Annual Conference*, 1237-42. Tokyo.

[14] Conway, J. H., and Smith, D. A. 2003. *On Quaternions and Octonions—Their Geometry, Arithmetic, and Symmetry.* A. K. Peters, Ltd.

[15] Garling, D. J. H. 2011. *Clifford Algebras: An Introduction.* Cambridge University Press.

[16] Kristiansen, R., and Nicklasson, P. J. 2005. "Satellite Attitude Control by Quaternion-Based Backstepping." *Proceedings of the 2005 American Control Conference* 2: 907-12.

[17] Vince, J. 2010. *Geometric Algebra for Computer Graphics.* Springer.