Learning Properties of Feedforward Neural Networks Using Dual Numbers

Yuto Okawa* and Tohru Nitta*

*Rikkyo University, Tokyo, Japan

***Abstract*— Real-valued neural networks (real-NN) with real numbers is a popular neural network model. Complex-valued neural network (complex-NN) is an extension of the real-NN to the complex domain where the inputs, outputs, weights, and biases are all complex numbers. Dual number is a two-dimensional number and is kind of a relative of complex numbers. In this paper, we investigate a feedforward neural network extended to dual numbers (dual-NN). It is found that the dual-NN has different properties from the real-NN and the complex-NN. Specifcally, for the input (two-dimensional information), the weights are trained with the constraint of the motion of shearing. Experimental results show that the generalization ability is higher than those of the real-NN and the complex-NNs for the training pattern with the shearing relation.**

I. INTRODUCTION

A dual number is one of the binarion expressed as

$$z = x + \varepsilon y \quad , \quad \varepsilon^2 = 0. \tag{1.1}$$

The complex number is also one of the binarion, so the dual number and the complex number have similar properties. As complex numbers, we call x the real-part and y the dual-part, and denote them by $Re\, z = x$ and $Du\, z = y$, respectively. The conjugate for a dual number $z = x + \varepsilon y$ is defined by $\bar{z} = x - \varepsilon y$. Also, a dual number z can be viewed as a point (x, y) in the dual plane. The absolute value of the dual number can be obtained by multiplying z by the conjugate dual number $\bar{z}$, just like the complex numbers [1]. The difference from complex numbers is that $z = 0 + \varepsilon y$ itself is zero divisors, and all zero divisors are on the y-axis (the dual-axis). If a function $f(z)$ is represented by $u + \varepsilon v$ in the dual plane, it is differentiable for all $z = x + \varepsilon y \in \mathbb{D}$ since it satisfies the Cauchy-Riemann equation, provided that $\partial u/\partial x = \partial v/\partial y, \partial u/\partial y = 0$ where $\mathbb{D}$ is the set of dual numbers [2].

Sven and Gerald compared the learning losses and test losses of the multi-layer perceptrons (MLP) using real numbers, complex numbers, hyperbolic numbers and dual numbers. As a result, they showed that the MLP using dual numbers do not learn some training data that uniformly drawn from $[0,1]^2$ well [3].

Nitta proposed a feedforward complex-valued neural network (complex-NN) in which the inputs, outputs, weights, and biases of the real-valued neural network (real-NN) are extended to complex numbers [4][5] and investigated its fundamental properties [6]. In this paper, in accordance with those researches, we formulate a feedforward dual-valued neural network (dual-NN), which is an extension of the real-NN to dual numbers, and clarify its learning properties.

II. PROPERTIES OF THE DUAL-NN

This section formulates a dual-NN and shows the properties of the weight parameters.

For simplicity, we consider a single-layer dual-NN, i.e., a single dual-valued neuron. Like the real-NN, the dual-NN multiplies each input with its corresponding weight and adds all the values together. Add a bias value to it, and pass it through the activation function to generate the output. The single-layer dual-NN is an NN that outputs a single dual $Z = X + \varepsilon Y \in \mathbb{D}$ given n dual-valued inputs $z_k = x_k + \varepsilon y_k (1 \le k \le n) \in \mathbb{D}$. The weights and biases are $w_k = w_k^{(r)} + \varepsilon w_k^{(\varepsilon)} (1 \le k \le n) \in \mathbb{D}$ and $\theta = \theta^{(r)} + \varepsilon\theta^{(\varepsilon)} \in \mathbb{D}$ respectively. The activation function $f_D: \mathbb{D} \to \mathbb{D}$ is defined as

$$f_D(z) = f_R(x) + \varepsilon f_R(y) \quad z = x + \varepsilon y, \tag{2.1}$$

where $f_R(u) = 1/(1 + \exp(-u)) \in \mathbb{R}$ where $\mathbb{R}$ is the set of real numbers. The output of the dual-NN $X + \varepsilon Y$ can be expressed as

$$\begin{aligned} &X + \varepsilon Y \\ &= f_D\left(\sum_{k=1}^{n}\left\{\left(w_k^{(r)} + \varepsilon w_k^{(\varepsilon)}\right)\left(x_k + \varepsilon y_k\right)\right\} + \left(\theta^{(r)} + \varepsilon\theta^{(\varepsilon)}\right)\right) \\ &= f_R\left(\sum_{k=1}^{n} w_k^{(r)} x_k + \theta^{(r)}\right) \\ &\qquad + \varepsilon f_R\left(\sum_{k=1}^{n}\left(w_k^{(\varepsilon)} x_k + w_k^{(r)} y_k\right) + \theta^{(\varepsilon)}\right). \end{aligned} \tag{2.2}$$

In this way, the dual-valued neuron with n inputs can be represented by a real-valued neuron with n inputs and a real-valued neuron with $2n$ inputs. Because of $\varepsilon^2 = 0$, the real-valued neuron corresponding to the real-part X has n fewer terms than the real-valued neuron corresponding to the dual-part Y.

The arguments of f_R for the two terms in (2.2) are expressed as

$$\begin{pmatrix} \begin{pmatrix} w_1^{(r)} & 0 \\ w_1^{(\varepsilon)} & w_1^{(r)} \end{pmatrix} & \cdots & \begin{pmatrix} w_n^{(r)} & 0 \\ w_n^{(\varepsilon)} & w_n^{(r)} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \\ \vdots \\ \begin{pmatrix} x_n \\ y_n \end{pmatrix} \end{pmatrix} + \begin{pmatrix} \theta^{(r)} \\ \theta^{(\varepsilon)} \end{pmatrix} \tag{2.3}$$

in a matrix form. And the first term in (2.3) can be rewritten as

$$\begin{pmatrix} |w_1| \begin{pmatrix} \cos\alpha_1 & 0 \\ \sin\alpha_1 & \cos\alpha_1 \end{pmatrix} & \cdots & |w_n| \begin{pmatrix} \cos\alpha_n & 0 \\ \sin\alpha_n & \cos\alpha_n \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \\ \vdots \\ \begin{pmatrix} x_n \\ y_n \end{pmatrix} \end{pmatrix}, \tag{2.4}$$

where $\alpha_k := \tan^{-1}(w_k^{(\varepsilon)}/w_k^{(r)})$, and $|w_k| \begin{pmatrix} \cos\alpha_k & 0 \\ \sin\alpha_k & \cos\alpha_k \end{pmatrix}$ is represented by

$$|w_k| \cos\alpha_k \begin{pmatrix} 1 & 0 \\ \tan\alpha_k & 1 \end{pmatrix} \begin{pmatrix} x_k \\ y_k \end{pmatrix} \tag{2.5}$$

for any $(1 \le k \le n)$. Equation (2.5) is a transformation of the input $(x_k \quad y_k)^T$ by a matrix $\begin{pmatrix} 1 & 0 \\ \tan\alpha_k & 1 \end{pmatrix}$ followed by the multiplication by $|w_k| \cos\alpha_k$. The transformation, represented by the matrix $\begin{pmatrix} 1 & 0 \\ \tan\alpha_k & 1 \end{pmatrix}$, corresponds to a motion called **shearing**. Shearing is the transformation of the vector $(x_k \quad y_k)^T$ by scalar multiplying y_k without moving x_k (Fig 1):

$$\begin{pmatrix} 1 & 0 \\ \tan\alpha_k & 1 \end{pmatrix} \begin{pmatrix} x_k \\ y_k \end{pmatrix} = \begin{pmatrix} x_k \\ x_k \tan\alpha_k + y_k \end{pmatrix}. \tag{2.6}$$

The net input to the dual-NN is a linear transformation of shearing, scalar multiplication, and translation of the input dual (2-dimentional information) as shown in Fig 1. In other words, the weights are updated under the constraint with respect to the movement of shearing.

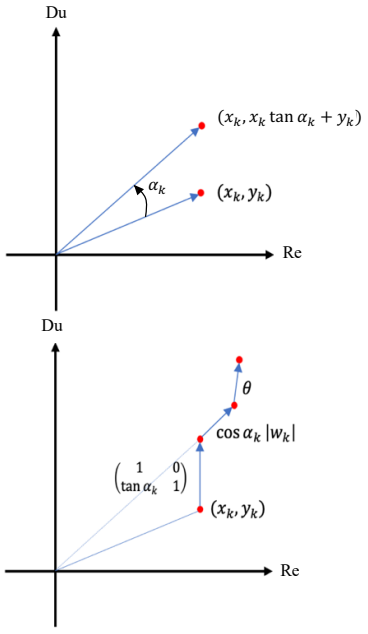

Figure 1. Images of shearing (top) and two-dimensional motion (bottom) for a dual signal

III. BACKPROPAGATION LEARNING ALGORITHM

In this section, we derive a backpropagation (BP) learning algorithm for a dual-NN. For simplicity, we assume that the network has three layers.

First, we formulate a three-layer dual-NN that has L input neurons, M hidden neurons and N output neurons. Let $\boldsymbol{X} = \boldsymbol{x} + \varepsilon\boldsymbol{y} = [x_1, \dots, x_L]^T + \varepsilon[y_1, \dots, y_L]^T$ denote the input to the dual-NN. We will use $w_{ji} = w_{ji}^{(r)} + \varepsilon w_{ji}^{(\varepsilon)}$ for the weight between the input neuron i and hidden neuron j, $v_{kj} = v_{kj}^{(r)} + \varepsilon v_{kj}^{(\varepsilon)}$ for the weight between the hidden neuron j and output neuron k, $\theta_j = \theta_j^{(r)} + \varepsilon\theta_j^{(\varepsilon)}$ for the bias of the hidden neuron j, and $\gamma_k = \gamma_k^{(r)} + \varepsilon\gamma_k^{(\varepsilon)}$ for the bias of the output neuron k. The activation function of each neuron is defined in (2.1). The net input to the hidden neuron j is

$$\begin{aligned} U_j &= U_j^{(r)} + \varepsilon U_j^{(\varepsilon)} \\ &= \sum_{i=1}^{L} w_{ji}^{(r)} x_i + \theta_j^{(r)} + \varepsilon\left(\sum_{i=1}^{L}\left(w_{ji}^{(\varepsilon)} x_i + w_{ji}^{(r)} y_i\right) + \theta_j^{(\varepsilon)}\right), \end{aligned} \tag{3.1}$$

and its output is

$$H_j = H_j^{(r)} + \varepsilon H_j^{(\varepsilon)} = f_R\left(U_j^{(r)}\right) + \varepsilon f_R\left(U_j^{(\varepsilon)}\right). \tag{3.2}$$

Then, the net input to the output neuron k is

$$\begin{aligned} S_k &= S_k^{(r)} + \varepsilon S_k^{(\varepsilon)} \\ &= \sum_{j=1}^{M} v_{kj}^{(r)} H_j^{(r)} + \gamma_k^{(r)} \\ &\quad + \varepsilon\left(\sum_{i=1}^{L}\left(v_{kj}^{(\varepsilon)} H_j^{(r)} + v_{kj}^{(r)} H_j^{(\varepsilon)}\right) + \gamma_k^{(\varepsilon)}\right), \end{aligned} \tag{3.3}$$

and its output is

$$O_k = O_k^{(r)} + \varepsilon O_k^{(\varepsilon)} = f_R\left(S_k^{(r)}\right) + \varepsilon f_R\left(S_k^{(\varepsilon)}\right). \tag{3.4}$$

Next, we derive the backpropagation learning algorithm for the three-layer dual-NN described above (called *dual-BP* here). The dual-BP is essentially the same as the Complex-BP, a backpropagation learning algorithm for the complex-NN [4, 5]. First, we initialize the weights and biases with random numbers. Next, we propagate the input forward and calculate the error between the actual output $\boldsymbol{O} = \boldsymbol{O}^{(r)} + \varepsilon\boldsymbol{O}^{(\varepsilon)}$ and the training data $\boldsymbol{T} = \boldsymbol{T}^{(r)} + \varepsilon\boldsymbol{T}^{(\varepsilon)}$ where, $\boldsymbol{O}^{(r)} = \left[O_1^{(r)}, \dots, O_N^{(r)}\right]^T$,

$\boldsymbol{O}^{(\varepsilon)} = [O_1^{(\varepsilon)}, \dots, O_N^{(\varepsilon)}]^T$, $\boldsymbol{T}^{(r)} = [T_1^{(r)}, \dots, T_N^{(r)}]^T$, and $\boldsymbol{T}^{(\varepsilon)} = [T_1^{(\varepsilon)}, \dots, T_N^{(\varepsilon)}]^T$. Let δ_k be the error between the actual output value $O_k = O_k^{(r)} + \varepsilon O_k^{(\varepsilon)}$ and the corresponding training data $T_k = T_k^{(r)} + \varepsilon T_k^{(\varepsilon)}$ of output neuron k. That is, $\delta_k = T_k - O_k$. We first considered the squared error $(1/2)\sum_{k=1}^{N}|T_k - O_k|^2$ as the error function to minimize. However, the absolute value of the dual number makes the real part disappear, that is, due to the property of the dual numbers: $\varepsilon^2 = 0$, the error function becomes $(1/2)\sum_{k=1}^{N}\left(\delta_k^{(r)}\right)^2$. Thus, we realized that the squared error is not suitable as the learning error function for the dual-BP. Therefore, we adopted here the Euclidean distance in the dual plane as the error function. That is, the error function E is defined as

$$E = \frac{1}{2}\sum_{k=1}^{N}\|T_k - O_k\|_E^2 = \frac{1}{2}\sum_{k=1}^{N}\left\|\delta_k^{(r)} + \varepsilon\delta_k^{(\varepsilon)}\right\|_E^2 = \frac{1}{2}\sum_{k=1}^{N}\left(\left(\delta_k^{(r)}\right)^2 + \left(\delta_k^{(\varepsilon)}\right)^2\right), \quad (3.5)$$

where $\|z\|_E = \sqrt{x^2 + y^2}, z = x + \varepsilon y$. Then, the error function E is partially differentiated with respect to each parameter to obtain the gradient of the parameter. We then multiply the gradient by a sufficiently small value $\eta > 0$ (learning rate) to obtain the update amount, and update it using the following algorithm.

$$\gamma_k^{new} = \gamma_k^{old} - \eta\Delta\gamma_k, \quad (3.6)$$

$$v_{kj}^{new} = v_{kj}^{old} - \eta\Delta v_{kj}, \quad (3.7)$$

$$\theta_j^{new} = \theta_j^{old} - \eta\Delta\theta_j, \quad (3.8)$$

$$w_{ji}^{new} = w_{ji}^{old} - \eta\Delta w_{ji}. \quad (3.9)$$

Each gradient is respectively expressed as

$$\Delta\gamma_k = \frac{\partial E}{\partial\gamma_k^{(r)}} + \varepsilon\frac{\partial E}{\partial\gamma_k^{(\varepsilon)}} = O_k^{(r)}\left(1 - O_k^{(r)}\right)\left(-2\delta_k^{(r)}\right) + \varepsilon\left(O_k^{(\varepsilon)}\left(1 - O_k^{(\varepsilon)}\right)\left(-\delta_k^{(\varepsilon)}\right)\right), \quad (3.10)$$

$$\Delta v_{kj} = \frac{\partial E}{\partial v_{kj}^{(r)}} + \varepsilon\frac{\partial E}{\partial v_{kj}^{(\varepsilon)}} = \left(H_j^{(r)}\Delta\gamma_k^{(r)} + H_j^{(\varepsilon)}\Delta\gamma_k^{(\varepsilon)}\right) + \varepsilon\left(H_j^{(r)}\Delta\gamma_k^{(\varepsilon)}\right), \quad (3.11)$$

$$\Delta\theta_j = \frac{\partial E}{\partial\theta_j^{(r)}} + \varepsilon\frac{\partial E}{\partial\theta_j^{(\varepsilon)}} = H_j^{(r)}\left(1 - H_j^{(r)}\right)\sum_k\left(v_{kj}^{(r)}O_k^{(r)}\left(1 - O_k^{(r)}\right)\left(-2\delta_k^{(r)}\right) + v_{kj}^{(\varepsilon)}O_k^{(\varepsilon)}\left(1 - O_k^{(\varepsilon)}\right)\left(-\delta_k^{(\varepsilon)}\right)\right) + \varepsilon\left(H_j^{(\varepsilon)}\left(1 - H_j^{(\varepsilon)}\right)\sum_k\left(v_{kj}^{(r)}O_k^{(\varepsilon)}\left(1 - O_k^{(\varepsilon)}\right)\left(-\delta_k^{(\varepsilon)}\right)\right)\right), \quad (3.12)$$

$$\Delta w_{ji} = \frac{\partial E}{\partial w_{ji}^{(r)}} + \varepsilon\frac{\partial E}{\partial w_{ji}^{(\varepsilon)}} = \left(x_i\Delta\theta_j^{(r)} + y_i\Delta\theta_j^{(\varepsilon)}\right) + \varepsilon\left(x_i\Delta\theta_j^{(\varepsilon)}\right). \quad (3.13)$$

IV. LEARNING PERFORMANCE

In this section, we evaluate the learning performance of the dual-BP via computer simulations.

We implemented the dual-BP using Python. The initial values of each parameter such as weights and biases used for training were set with random numbers generated from the standard normal distribution. We considered the sum of the learning errors for each pattern to be the learning error for one epoch, and if the learning error for one epoch was less than 0.1, we judged that learning was complete. The maximum number of the number of learning epochs was set to 10,000 epochs (1,000 epochs only for the first learning pattern). The learning rate η is varied from 0.1 to 1.0, and 50 trials are performed for each learning rate.

Pattern ①

First, we used the training set in Table 1. The inputs are points aligned on a line parallel to the real-axis in the dual plane, and the outputs are parallel translations (shearing) by the value of the dual-axis. If the input is $z = x + \varepsilon y$, the dual-part of the corresponding output is $(x^2 + y) + 2$. For example, if the input is a point $c = a + \varepsilon 0$ on the real-axis, then the expected output is $a + \varepsilon(a^2 + 0 + 2)$ (Fig. 2).

The convergence rates for the real-NN, the complex-NN and the dual-NN are shown in Table 2. We compared the generalization error of the dual-NN with those of the real-NN and the complex-NN. The experimental results are shown in Fig 3. We can find from Fig 3 that the generalization error of the dual-NN is smaller than those of the real-NN and the complex-NN at the learning rates ranging within 0.1 and 0.6. All the generalization errors of the real-NN, complex-NN and dual-NN were large at the learning rates ranging within 0.7 and 1.0.

Pattern No.	Input pattern	Output pattern
1	$-4 - 2\varepsilon$	$-4 + 16\varepsilon$
2	$-3 - 2\varepsilon$	$-3 + 9\varepsilon$
3	$-2 - 2\varepsilon$	$-2 + 4\varepsilon$
4	$-1 - 2\varepsilon$	$-1 + \varepsilon$
5	-2ε	0
6	$1 - 2\varepsilon$	$1 + \varepsilon$
7	$2 - 2\varepsilon$	$2 + 4\varepsilon$
8	$3 - 2\varepsilon$	$3 + 9\varepsilon$
9	$4 - 2\varepsilon$	$4 + 16\varepsilon$

Table 1. Learning patterns ①

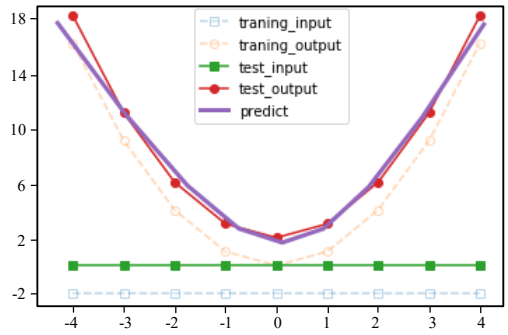

Figure 2. Learning patterns ①, input test patterns, output test patterns (i.e., expected outputs for the input test patterns). "predict" means the actual value output by the model learned the learning pattern ① for the "test input"

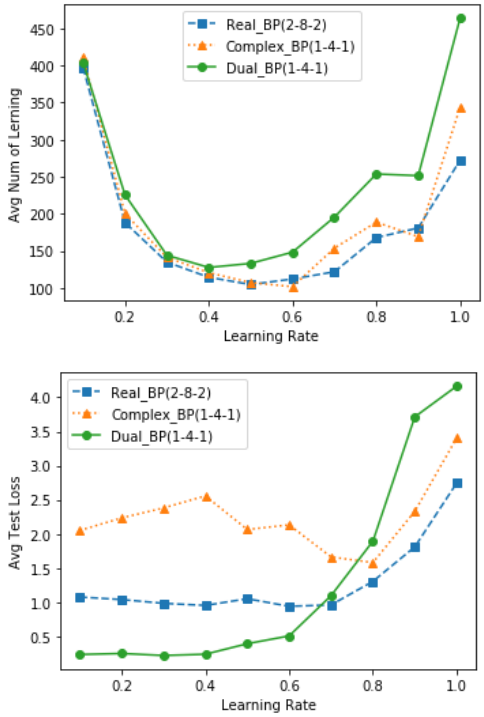

Figure 3. Average number of learning epochs in convergence (top) and average test error (bottom) for pattern ①

η	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Real	100	100	100	100	100	100	98	80	74	54
Complex	98	100	100	100	100	98	98	80	64	54
Dual	98	100	100	100	100	92	72	50	30	18

Table 2. The convergence rates (%) for the real-NN, the complex-NN and the dual-NN for pattern ①

Pattern ②

Next, we changed the training set from the one for learning quadratic functions to the one for learning sin functions. In the same way as the pattern ①, the points aligned on a line parallel to the real-axis in the dual plane are used as input to learn the fitting to the sin function. Specifically, when we input $z = x + \varepsilon y$, $x + \varepsilon(sin5x + y + 0.5)$ is output (Fig 4). The training set is shown in Table 3.

The experimental results show that the generalization error of the dual-NN is smaller than or nearly equal to those of the other two NNs at the learning rates $\eta = 0.1, \ldots, 0.7$, although the number of learning cycles of the dual-NN is somewhat larger than those of other two NNs (Fig 5). We excluded the experimental results for the learning rates ranging within 0.8 and 1.0 from discussions because the convergence rates were extremely low. The convergence rate for the real-NN, the complex-NN and the dual-NN are shown in Table 4. In the

Pattern No.	Input pattern	Output pattern
1	$-1.0 - 0.5\varepsilon$	$-1.0 + 0.96\varepsilon$
2	$-0.8 - 0.5\varepsilon$	$-0.8 + 0.76\varepsilon$
3	$-0.6 - 0.5\varepsilon$	$-0.6 - 0.14\varepsilon$
4	$-0.4 - 0.5\varepsilon$	$-0.4 - 0.91\varepsilon$
5	$-0.2 - 0.5\varepsilon$	$-0.2 - 0.84\varepsilon$
6	0.5ε	0
7	$0.2 - 0.5\varepsilon$	$0.2 + 0.84\varepsilon$
8	$0.4 - 0.5\varepsilon$	$0.4 + 0.91\varepsilon$
9	$0.6 - 0.5\varepsilon$	$0.6 + 0.14\varepsilon$
10	$0.8 - 0.5\varepsilon$	$0.8 - 0.76\varepsilon$
11	$1.0 - 0.5\varepsilon$	$1.0 - 0.96\varepsilon$

Table 3. Learning patterns ②

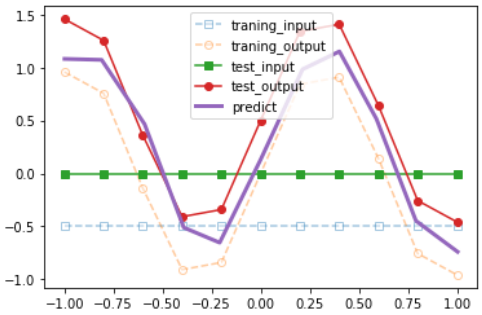

Figure 4. Learning patterns ②, input test patterns, output test patterns (i.e., expected outputs for the input test patterns). "predict" means the actual value output by the model learned the learning pattern ②for the "test input"

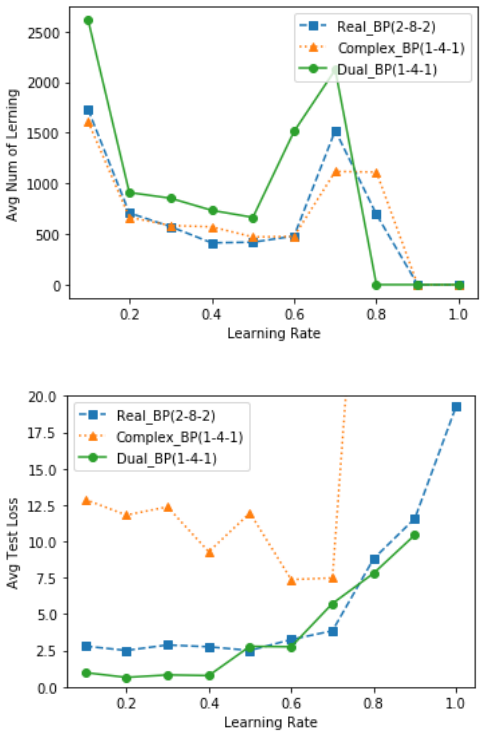

Figure 5. Average number of learning epochs in convergence (top) and average test error (bottom) for pattern ②

η	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Real	100	100	100	100	98	70	32	0	0	0
Complex	100	100	100	100	100	100	92	6	0	0
Dual	100	100	100	100	100	90	80	4	0	0

Table 4. The convergence rates (%) for the real-NN, the complex-NN, and the dual-NN for pattern ②

dual-NN, the convergence rate is nearly 0% for learning rates $\eta = 0.8, 0.9, 1.0$, but this is not a drawback unique to the dual-NN because the convergence rate of the real-NN is also nearly 0%.

Pattern ③

Finally, we used a more complicated function: a function that superimposed several sin functions. Specifically, we considered a function that outputs $x + \varepsilon(4/\pi\{\sin x + (1/3)\sin 3x + (1/5)\sin 5x + (1/7)\sin 7x\} + y + 0.5)$ for an input point $z = x + \varepsilon y$ aligned on a line parallel to the real-axis in the dual plane (Fig 6). The training set is shown in Table 5.

Fig 7 shows that the generalization error of the dual-NN is smaller than those of the other two NNs at almost all learning rates. In the case of the learning pattern ③, the number of learning cycles of the dual-NN needed to converge was almost the same as those of the real-NN and the complex-NN. The possible reason for this is that the function is difficult for the real-NN and the complex-NN to learn. However, this needs to be further investigated. As for the convergence rate, they are shown in Table 6. There was no significant difference in the convergence rate.

Pattern No.	Input pattern	Output pattern
1	$-1.0 - 0.5\varepsilon$	$-1.0 - 1.01\varepsilon$
2	$-0.8 - 0.5\varepsilon$	$-0.8 - 0.89\varepsilon$
3	$-0.6 - 0.5\varepsilon$	$-0.6 - 1.01\varepsilon$
4	$-0.4 - 0.5\varepsilon$	$-0.4 - 1.18\varepsilon$
5	$-0.2 - 0.5\varepsilon$	$-0.2 - 0.89\varepsilon$
6	0.5ε	0
7	$0.2 - 0.5\varepsilon$	$0.2 + 0.89\varepsilon$
8	$0.4 - 0.5\varepsilon$	$0.4 + 1.18\varepsilon$
9	$0.6 - 0.5\varepsilon$	$0.6 + 1.01\varepsilon$
10	$0.8 - 0.5\varepsilon$	$0.8 - 0.89\varepsilon$
11	$1.0 - 0.5\varepsilon$	$1.0 - 1.01\varepsilon$

Table 5. Learning patterns ③

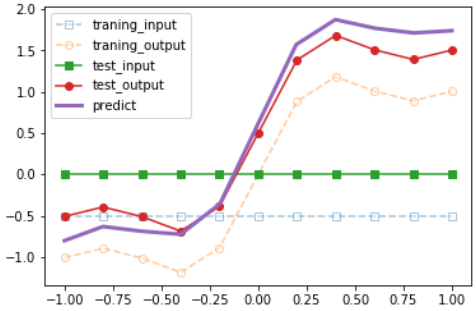

Figure 6. Learning patterns ③, input test patterns, output test patterns (i.e., expected outputs for the input test patterns). "predict" means the actual value output by the model learned the learning pattern ③ for the "test input"

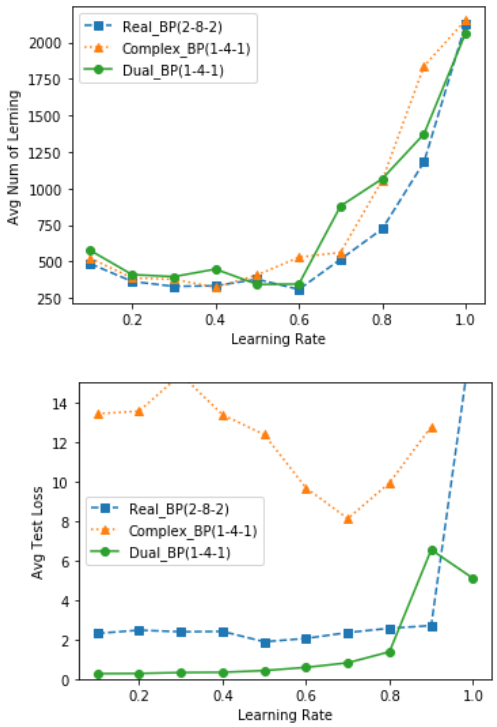

Figure 7. Average number of learning epochs in convergence (top) and average test error (bottom) for pattern ③

η	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Real	100	100	100	100	100	100	96	96	84	28
Complex	100	100	100	100	100	100	98	94	68	22
Dual	100	100	100	100	100	100	98	94	94	54

Table 6. The convergence rates (%) for the real-NN, the complex-NN, and the dual-NN for pattern ③

V. CONCLUSIONS

We have formulated a feedforward dual-valued neural network, derived a dual-valued backpropagation learning algorithm for it, and investigated their fundamental properties. As a result, we have found that the parameters of the dual-NN are learned under the constraint of shear motion. Simulation results show that when the learning pattern is one that learns shear motion (e.g., when inputting points aligned on a real axis, the pattern produces an output that fits a certain quadratic or trigonometric function), the generalization performance is better than those of the real-NN and the complex-NN.

In the future, we will study how to speed up the learning process. As other issues, we will extend the activation function ReLU, which is commonly used in NNs, to dual number to further stabilize the learning. We also plan to construct deep dual-NNs and study the differences in learning performance among the deep dual-NNs, deep real-NNs, and deep complex-NNs [7].

ACKNOWLEDGMENT

The authors would like to give special thanks to the members of the Cooperative Research Project of the Research Institute of Electrical Communication, Tohoku University for the stimulated discussion, and the anony-mous reviewers for valuable comments. This work was supported by JSPS KAKENHI Grant Number JP16K00347.

REFERENCES

[1] Anthony A. Harkin and Joseph B. Harkin, "Geometry of Generalized Complex Numbers", *Mathematics Magazine*, Vol. 77, No. 2, Permutations (Apr., 2004), pp. 118-129

[2] K. DenHartigh, and R. Flim, "Liouville theorems in the Dual and Double Planes." *Rose-Hulman Undergraduate Mathematics Journal* 12.2 (2011): 4.

[3] S. Buchholz, and G. Sommer. "On Clifford neurons and Clifford multi-layer perceptrons." *Neural Networks* 21.7 (2008): 925-935.

[4] T. Nitta, "An Extension of the Back-Propagation Algorithm to Complex Numbers", *Neural Networks,* Vol.10, No.8, pp.1391-1415 (1997).

[5] T. Nitta, "An analysis of the fundamental structure of complex-valued neurons." *Neural Processing Letters* 12.3 (2000): 239-246.

[6] T. Nitta and T. Furuya, "A Complex Back-propagation Learning", *Transactions of Information Processing Society of Japan*, Vol.32, No.10, pp.1319-1329 (1991) (in Japanese).

[7] C. Trabelsi, O. Bilaniuk, Y. Zhang, D. Serdyuk, S. Subramanian, J. Felipe Santos, S. Mehri, N. Rostamzadeh, Y. Bengio and C. J Pal, "Deep Complex Networks", ICLR 2018