Chapter II
Complex-Valued Neural Network and Inverse Problems

Takehiko Ogawa
Takushoku University, Japan

ABSTRACT
Network inversion solves inverse problems to estimate cause from result using a multilayer neural network. The original network inversion has been applied to usual multilayer neural networks with real-valued inputs and outputs. The solution by a neural network with complex-valued inputs and outputs is necessary for general inverse problems with complex numbers. In this chapter, we introduce the complex-valued network inversion method to solve inverse problems with complex numbers. In general, difficulties attributable to the ill-posedness of inverse problems appear. Regularization is used to solve this ill-posedness by adding some conditions to the solution. In this chapter, we also explain regularization for complex-valued network inversion.

INTRODUCTION
It is necessary to solve inverse problems for estimating causes from observed results in various engineering fields. In particular, inverse problems have been studied in the field of mathematical science (Groetsch, 1993). The inverse problem determines the inner mechanisms or causes of an observed phenomenon. The cause is estimated from the fixed model and the given result in the inverse problem, while the result is determined from the given cause by using a certain fixed mathematical model in the forward problem. As a solution of inverse problems, the neural network based method has been proposed while other method such as the statistical method (Kaipio & Somersalo, 2005) and parametric method (Aster, Borchers, & Thurber, 2005) have also been studied.

The idea of inverting network mapping was proposed by Williams (1986). Then, Linden and Kindermann proposed a method of network inversion (Linden & Kindermann, 1989). Also, the algorithms and applications of network inversion are summarized by Jansen et al. (1999). In this method, inverse problems are solved by the inverse use of the input-output relation of trained multilayer neural networks. In other words, the corresponding input is estimated from the provided output via fixed weights, after finding the forward relation by network train-
ing. The direction of the input-output relation between the training and the inverse estimation is important in this method. The estimation process in multilayer neural networks is considered from the viewpoint of forward and inverse problems. The usual estimation process of multilayer neural networks provides a solution for forward problems because the network estimates the output from the input provided by the forward relation obtained in the training. On the other hand, we can solve inverse problems using multilayer neural networks that learn the forward relation by estimating the input from the given output inversely. Network inversion has been applied to actual problems; e.g., medical image processing (Valova, Kameyama, & Kosugi, 1995), robot control (Lu & Ito, 1995; Ogawa, Matsuura, & Kanada, 2005), optimization problems, and so on (Murray, Heg, & Pohlhammer, 1993; Ogawa, Jitsukawa, Kanada, Mori, & Sakata, 2002; Takeuchi & Kosugi, 1994). Moreover, the answer-in-weights scheme has been proposed to solve the difficulty of ill-posed inverse problems, as a related model of network inversion (Kosugi & Kameyama, 1993).

The original network inversion method proposed by Linden and Kindermann solves an inverse problem by using a usual multilayer neural network that handles the relation between real-valued input and output. However, a network method for complex-valued input and output is required to solve the general inverse problem whose cause and result extend to the complex domain. On the other hand, there exists an extension of the multilayer neural network to the complex domain (Benvenuto & Piazza, 1992; Hirose, 2005; Nitta, 1997). The complex-valued neural network learns the relations between complex-valued input and output in the form of complex-valued weights. This complex-valued network inversion was considered to solve inverse problems that extended to complex-valued input and output. In this method, the complex-valued input is inversely estimated from the provided complex-valued output by extending the input correction of the original network inversion method to the complex domain. Actually, the complex-valued input is estimated from the complex-valued output by giving a random input to the trained network, back-propagating the output error to the input, and correcting the input (Ogawa & Kanada, 2005a).

In the forward problem, the existence, uniqueness, and stability of solution are guaranteed. When all these three conditions are satisfied, the problem is called a well-posed problem. In the inverse problem, it is often difficult to obtain a solution because of ill-posedness (Petrov & Sizikov, 2005; Tikhonov & Arsenin, 1977). Regularization imposes specific conditions on an ill-posed inverse problem to convert it into a well-posed problem. We consider introducing regularization to complex-valued network inversion. Tikhonov regularization, which is a conventional regularization method, uses a constant coefficient for the regularization term. On the other hand, dynamic regularization (Kosugi, Uemoto, & Ogawa, 1998) was proposed for changing the effect of regularization in each stage of the process.

The first objective of this chapter is to demonstrate the procedure of the complex-valued network inversion method to solve complex-valued inverse problems. The other objective is to show the effect of the regularization method. To achieve these two objectives, the following problems are examined: inverse estimation of complex-valued mapping, inverse Fourier transform problem, and inverse estimation of complex-valued mapping with ill-posedness.

**BACKGROUND**

**Inverse Problems**

An inverse problem determines the inner mechanism or cause of an observed phenomenon. For the relation $Kx = y$, where $K$ is some mathematical model, we consider the forward problem, which determines the result $y$ from the cause $x$. In this forward problem, generally, the definition of the operator $K$ is fixed and its mapping is supposed to be continuous. For this forward problem, we can define an inverse problem that determines the cause $x$ from the operator $K$ and result $y$ or one that estimates the operator $K$ from the cause $x$ and the result $y$. The relation between the forward and inverse problems is shown in Figure 1.

The relation of the given cause $x$ and the result $y$ is continuous in the forward problem. In the inverse problem, the existence or uniqueness of cause $x$ is not guaranteed for a given or observed result $y$. Even if the cause
$x$ is determined or estimated uniquely, it sometimes becomes unstable to minute perturbations of the result $y$. Existence, uniqueness, and stability are necessary conditions to ensure the well-posedness of a problem. In the inverse problem, it might be difficult to obtain a solution because the well-posedness of the problem is not always guaranteed. The difficulty is often referred as an ill-posedness. The concept of ill-posedness with regard to existence, uniqueness, and stability is shown in Figure 2.

In general, locating a source from observed data is known as an inverse problem. Two examples of such problems are the localization of active nerves from their evoked potential waveforms and the localization of objects from their echoes using an active sonar system.

**Example: Nerve Bundle Localization with Evoked Potentials.**

During neurosurgical operations, in addition to approaching and correcting malfunctioning parts, the localization of healthy nerves that must not be injured is an important and difficult task. Estimating the location of active nerves by observing evoked potentials during surgery has been proposed as a method to simplify this task. Experience with the standard approach of using electrodes on the surface of the scalp indicates that the determination of nerve location during surgery should satisfy two conditions: it should be performed in real time and the observation device (probe) should be introduced intracranially (Watanabe & Kosugi, 1991).

A percutaneous stimulus of the nerves generates evoked potentials that are synchronized with the stimulus. Since the nerve bundles consist of many axons with different transmission velocities, the pulse transmission in the nerve bundle $P$ is approximated as the sum of dipole moment $P_i$. For example, the potential $V$ at point $A$ in Figure 3 can be approximated by

$$V = \sum \frac{P_i}{4\pi e_{r}} \cos \theta, \quad P_i = \delta_i Q_i$$

(1)
Complex-Valued Neural Network and Inverse Problems

Figure 3. The physical principle of evoked potential generation. (a) The transmission of the pulse in the nerve bundle. (b) The dipole moment used to approximate the pulse.

Figure 4. The probe with four pairs of electrodes to detect each directional component of the evoked potential

where $Q_i$ is an electrical charge; $\varepsilon_0$, the permittivity of free space; and $\delta_i$, the separation between charges. The detecting probe, shown in Figure 4, consists of four pairs of electrodes that are connected in diagonal pairs to detect each directional component of the evoked potential. When the probe direction is changed relative to the nerves, each electrode pair detects the charge in the evoked potential. An example of evoked potential waveform of the median nerve is shown in Figure 5. The changes in the detected waveforms facilitate the observation of the location of the nerves with respect to the probe. The localization of the median nerve within the arm was attempted as a preliminary experiment to locate nerves within the brain for neurosurgical operations. There are some differences in the pulse width of the evoked potential, as well as the transmission characteristics in the surrounding tissues, between those in the brain and in the arm, but the principles of generation and transmission of the evoked potentials is essentially the same. Therefore, the experiment gives important insights prior to the location of nerves within the brain.

It is assumed that the detected evoked potential waveform is determined by the relation between the location of nerves (source location) and the probe position (sensor position). Thus, we can form an inverse problem
Complex-Valued Neural Network and Inverse Problems

Example: Source Localization with Bat-Like Sonar

Sonar systems using ultrasonic waves have been employed for navigation, registration, obstacle avoidance, and sonar map building, since they can determine the object location easily and inexpensively. Among these object recognition systems, the bat-like sonar system (Barshan & Kuc, 1992; Kuc & Barshan, 1991), shown in Figure 6 was implemented to simulate a biological system. In this system, the middle transducer \( T \) transmits an ultrasonic pulse, and two transducers \( R_1 \) and \( R_2 \) receive the echoes reflected by the objects. A typical received echo is shown in Figure 7. The pressure amplitude of the propagating pulse is given by

\[
p(r, \theta) \approx \frac{P_0 r}{r} e^{-\frac{r^2}{2\sigma_t^2}}, \quad r > r_0,
\]

where \( r \) is the radial distance from the transducer; \( \theta \), the azimuth; and \( P_0 \), the propagating pressure amplitude at range \( r_0 \) along the line-of-sight (\( \theta = 0 \) degree). For our sensor, \( r_0 \approx 100 \text{mm} \) and the beam-width parameter \( \sigma_t \) equals 30°.

In the ideal environment, we assume that no object other than the target object reflects waves. However, in the actual environment, there may be many objects that produce echoes. As a result, we may not be able to tell which
echoes belong to the desired object whose location we wish to determine. Bats living in a narrow cave receive many echoes, but they still manage to determine the location of a desired object, like a perch. In the bat-like sonar problem, the relation between the waveforms and the object location is determined by the relation between the object location (source location) and the detector position at which the sensors detect the waves reflected by the object (sensor position). In other words, we want to estimate the object location from a pair of physical quantities describing the detector position and the detected waveforms. Thus, we can form an inverse problem to estimate the target location to compose the environmental map from the detected waveforms (Ogawa et al., 1996).

**Network Inversion**

A conventional multilayer neural network is used to solve the forward problem by the learned forward relation. In a usual multilayer network whose training has completed, the input-output relation is given by

\[ y = f(w, x) \]  

(3)

where \( x \), \( y \), and \( f \) are the input vector, the output vector, and the function defined by the interlayer weights \( w \) of the network, respectively. For a given input vector \( x \), the network calculates the output vector \( y \).

Linden and Kindermann proposed the method of network inversion. In this method, we can apply the observed output data \( y \) with \( f \) fixed, after finding forward relation \( f \) by training. Then, the input \( x \) can be updated according to the calculated input correction signal, based on the duality of the weights and input in eqn. (3). Actually, the input is estimated from the output by an iterative update of the input based on the output error. By this method, the inverse problem for estimating the input \( x \) from the output \( y \) is solved by the multilayer neural network by using the forward relation inversely.
Complex-Valued Neural Network and Inverse Problems

The network is used in the two phases: the forward training and the inverse estimation to solve the inverse problem by network inversion. The two-step procedure is shown in Figure 8. In the training phase, we provide the training input $x$ and the training output $y$ and calculate the output error $E$. Then, the weight $w$ is updated by

$$w(t+1) = w(t) - \epsilon_t \frac{\partial E}{\partial w}$$

(4)

where $\epsilon_t$ represents the training gain, because the output error is due to the misadjustments of weights. By repeating this update procedure shown in Figure 9 (a), the forward relation is obtained by the distribution of weights. This is the procedure based on the usual back-propagation method. In the inverse estimation phase, we fix the relation obtained in the training, provide the random input $x$ and the test output $y$, and calculate the output error $E$. Then, the input $x$ is updated by

$$x(t+1) = x(t) - \epsilon_x \frac{\partial E}{\partial x}$$

(5)

where $\epsilon_x$ represents the input update gain, because the output error is due to the error of the input. By repeating this update procedure shown in Figure 9 (b), the input is estimated from the output.

Linden and Kindermann mentioned in their paper (Linden & Kindermann, 1989) that the inversion method was presented as a tool for understanding network behavior. The purpose of the original network inversion was not to serve as a solution of an inverse problem. However, in their paper, they posed the question of which input should be fed into the net to produce an output that approximates a given target vector. This is an inverse problem in itself. Network inversion has been applied to a number of inverse problems: impedance tomography, segmentation of brain MR images, inverse kinematics, and so on (Lu & Ito, 1995; Takeuchi & Kosugi, 1994; Valova et al., 1995).

Since network inversion solves inverse problems by a two-stage procedure of forward training and inverse estimation, this method needs an iterative procedure not only in forward training but also in inverse estimation. The iterative update of inputs in inverse estimation is done only to one provided an output pattern while the iterative update of weights in the forward training is performed on all the given training patterns. Therefore, the calculation time is much shorter than that by iteration in training, even though the calculation time for iteration in inverse estimation is definitely required.

However, the computing time is obviously long as compared to that in the case where there is no repetition. For instance, we can perform estimation without iteration by using the “inverse solver” network, which studies the inverse relation between inputs and outputs. However, we can use the inverse solver network only when the relation of a simple one-to-one correspondence is guaranteed between the input and output.

Therefore, the use of the network inversion method is significant in the case where it is used to solve an inverse problem with the possibility of many-to-one correspondence between the input and output. To solve an inverse problem with the many-to-one correspondence of the input-output relation by the usual feed-forward network, the network must learn the one-to-many correspondence of the input-output relation. Because it is difficult for a network to do this, network training does not converge, even if we attempt to make the network actually learn the relation. In the case of network inversion, it is possible to learn the many-to-one correspondence of the input-output relation by training. In other words, this relation signifies the ill-posed inverse problem. It is not necessary for well-posed inverse problems to be solved by network inversion methods. However, there are merits to using the network inversion method to solve inverse problems, because a one-to-one relation is often not guaranteed.

Consequently, it is significant to apply network inversion methods to solve ill-posed inverse problems. Therefore, the solution of ill-posedness is inevitably required. Generally, an ill-posed inverse problem can be converted into a well-posed one by adding an a priori given condition. Techniques such as the regularization method and the answer-in-weights scheme have been examined to provide a condition for network inversion.

**Regularization**

In general, inverse problems include difficulties with regard to ill-posedness. The ill-posedness problem is important in network inversion as well. Regularization, which has been studied in the field of mathematical science, is a solution to the problem of ill-posedness. The use of regularization has been studied for network inversion.
Figure 8. Two-step procedure to solve an inverse problem by network inversion:

- **Training phase**
  - Training input \( \rightarrow \) forward learning \( \rightarrow \) training output

- **Estimation phase**
  - Estimated input \( \rightarrow \) inverse estimating \( \rightarrow \) test output

- **by multilayer NN**

Figure 9. Weights and input update procedure. (a) Weight update based on error back-propagation learning, and (b) input update in inverse estimation based on network inversion:

- Training output \( Y' \)
- Output \( Y \)
- Output layer
- Weights
- Hidden layer
- Weights
- Input layer

- Training input \( X \)

- Error \( E \)
- Weights update

- Test output \( Y' \)
- Output \( Y \)
- Output layer
- Weights
- Hidden layer
- Weights
- Input layer

- Estimated input \( X \)
- Error \( E \)
- Fixed weights
We consider regularization for network inversion. The method is based on Tikhonov regularization, which provides the constraint condition to the solution. To estimate the input from a given output, network inversion minimizes the output error, as expressed in eqn. (5). To provide the constraint condition, we use the regularization functional to be minimized in accordance with the output error in the inverse estimation phase. The energy function $E(x)$ with the regularization functional $\Omega(x)$ is defined as

$$E = \|y - Kx\|^2 + \lambda \Omega(x)$$

where $\tilde{y}$, $K$, and $x$ are the network output, translation, and input, respectively. The first and second terms represent the output error of the network and the regularization functional. The parameter $\lambda$ is the regularization coefficient.

It is possible to consider a number of regularization functionals: maximum norm, minimum norm, maximum inclination, minimum inclination, and so on.

For example, we consider the case of regularization of the minimum squared norm solution. In this case, this regularization method solves the ill-posed inverse problem by finding the minimum squared norm solution. The functional to minimize the squared norm is added to the output error function of network inversion. The functional and the output error function are minimized at the same time in the inverse estimation phase of network inversion. The output error $E$ of the network inversion is defined as

$$E = \sum_{i} (y_i - \tilde{y}_i)^2 - \lambda \|\mathbf{x}\|^2$$

where $\tilde{y} = \{\tilde{y}_i\}$, $y = \{y_i\}$, $i = \{1, \ldots, M\}$ and $X = \{\mathbf{x}_j\}$, $j = \{1, \ldots, N\}$ are the network output, the given output, and the input to be estimated, respectively. The regularization functional is defined as the product of the squared norm $\|\mathbf{x}\|^2$ and regularization parameter $\lambda$. The network input is updated by minimizing the squared norm $\|\mathbf{x}\|^2$.

The input is corrected by

$$x_j(t+1) = \lambda x_j(t) + \eta \delta_j$$

where $\delta_j$ represents the error signal for the $j$-th input neuron that is generated by the output error and is propagated from the output layer. The parameter $\eta$ is the coefficient for the iterative update of the input.

The coefficient $\lambda$ is usually fixed and determined empirically or by trial and error. However, the balance between output error minimization and regularization is often uncertain. Actually, the regularization coefficient might be reduced with a decrease in the output error. We refer to this as dynamic regularization. For example, the regularization coefficient $\lambda(t)$ is reduced for each correction of input as

$$\lambda(t) = \lambda(0) \exp(-mt)$$

where $\lambda(0)$, $t$, and $m$, are the initial value of $\lambda$, the current epoch number, and a decay coefficient, respectively. The parameter $\lambda(t)$ decays from $\lambda(0)$ to zero with the epoch number $t$.

Another approach for solving ill-posedness is the answer-in-weights scheme (Kosugi & Kameyama, 1993). In the original network inversion approach, only one port is used as the input port because the input is used as the output port, corresponding to the answer-in-input scheme. However, it is difficult to efficiently embed the relation between two different data types such as the signal waveform and the sensor location into the network architecture. Therefore, the method that seeks an answer from neither the input nor the output but from the weights in between the layers of the network is considered. In this approach, we assign different data types to the input and output, and obtain an answer satisfying their relationship in weights. This strategy is referred to as the answer-in-weights scheme. In this method, the physical conditions that the solution of the inverse problem has to satisfy can be easily built into the network architecture using the relation between the input and output. We can reduce the difficulty due to the ill-posedness of inverse problems, since more comprehensive constraints, i.e., relations among three physical or mathematical quantities, can be applied to the answer-in-weights network than those in the conventional feed-forward network. The concept of the answer-in-weights scheme is shown in Figure 10.
Example: Inverse Kinematics of Robot Arm

The network inversion has been applied to the inverse kinematics problem to estimate joint angles from the given end effector’s coordinate. The inverse kinematics problem of multi degree-of-freedom (DOF) is an ill-posed inverse problem. For example, we consider the inverse kinematics of the three DOF robot arm. The joint angles to realize for the given end effector’s coordinate is not unique in the inverse kinematics, while an end effector’s coordinate is uniquely decided from the given joint angles in the forward kinematics, as shown in Figure 11.

To show the procedure of the network inversion, we examine the inverse kinematics of the three DOF robot arm. We attempt to estimate the joint angles to realize the given end effector’s coordinate. The network architecture used in the simulation is shown in Figure 12. The joint angles \((\theta_1, \theta_2, \theta_3)\) and the end effector’s coordinate \((x, y)\) are provided as the input and output, respectively. The parameters of the network and the robot arm are shown in Table 1. In the learning phase, the network learns the forward relation between input and output. The error for forward epoch is shown in Figure 13(a). The joint angles and end effector’s coordinate are provided as the training input and output, respectively. In the inverse estimation phase, the joint angles \((\theta_1', \theta_2', \theta_3')\) are estimated from the given end effector’s coordinate \((x', y')\). The initial joint angles are set to random values. The regularization functional and its parameter used here is the same as the equations (7) and (9). We prepared the five test output data of the end effector’s coordinate arranged on the arc. By repeating this input update procedure, the input is estimated from the output.

The error for inverse epoch is shown in Figure 13(b). From the decrease in the error, we confirm that the inputs are updated correctly. The obtained pose of robot arm and end effector’s coordinates are shown in Figure 14(a). The estimated coordinates are almost corresponding to the target. Also, we tried the inverse estimation five times. The results are shown in Figure 18(b). The estimated five trajectories of the end effector’s coordinates are almost corresponding to the target. From the above results, the estimated joint angles with the regularization functional are almost similar to the target angles. This means the network inversion is able to solve the ill-posed inverse problems with regularization functional. As a result, we confirm the correct solution of inverse kinematics by the network inversion with regularization.
Figure 11. An example of three DOF robot arm, (a) joint angles and end effector’s coordinate, (b) combination of joint angles for an end effector’s coordinate

![Three DOF robot arm diagram](image)

Figure 12. Network architecture for inverse kinematics of three DOF robot arm

![Network architecture diagram](image)

Table 1. Network and robot arm parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of input neurons</td>
<td>3</td>
</tr>
<tr>
<td>Number of hidden neurons</td>
<td>60</td>
</tr>
<tr>
<td>Number of output neurons</td>
<td>2</td>
</tr>
<tr>
<td>Training gain $\varepsilon_t$</td>
<td>0.001</td>
</tr>
<tr>
<td>Estimation gain $\varepsilon_e$</td>
<td>0.001</td>
</tr>
<tr>
<td>Maximum number of training epochs</td>
<td>10000</td>
</tr>
<tr>
<td>Maximum number of estimation epochs</td>
<td>10000</td>
</tr>
</tbody>
</table>
| Joint angles for learning data (degree)  | $\theta_1$: 0, 10, …, 90  
                                              $\theta_2$: 0, 15, …, 345  
                                              $\theta_3$: 0, 30, …, 330 |
| Length of arms (cm)                      | 30             |
Figure 13. Error curves for (a) learning epoch and (b) inverse epoch

![Error curves](image)

Figure 14. Estimated results (a) pose of robot arm (b) five trajectories of end effector’s coordinate figured by estimated joint angles

![Estimated results](image)
COMPLEX-VALUED NETWORK INVERSION

The network inversion method proposed by Linden and Kindermann solves an inverse problem by a usual multilayer neural network that handles the relation between a real-valued input and output. However, a network method for complex-valued input and output is required to solve the general inverse problem whose cause and result expand into the complex domain. On the other hand, the extension of the multilayer neural network to the complex domain has already been proposed.

In this section, a complex-valued network inversion that handles complex input and output is explained. The method uses a multilayer neural network that has complex weights and complex neurons. In this method, the complex-valued neural network estimates the complex input from the complex output using the trained network. It is an extension of the input correction principle of usual network inversion to the complex domain. Actually, the complex input is estimated from the complex output by providing a random input to the trained network, back-propagating the output error to the input, and repeating the input.

We consider the effectiveness of the complex-valued neural network and complex-valued network inversion. If only complex-valued input and output need to be handled, it can be done by a real-valued neural network whose input and output are allocated to two neurons. It is considered that complex-valued network inversion can be substituted by real-valued network inversion through a similar allocation. One of the most significant features of the complex-valued neural network is the restriction of flexibility based on the fundamental arithmetic rule of multiplication of complex numbers (Hirose, 2005). A neural network has to construct this relation from the data provided by learning when a complex-valued neural network is substituted with a real-valued one. On the other hand, we consider that there is merit in using this relation as a restraint condition in learning in the complex-valued neural network.

Similar to real-valued network inversion, the network is used in two phases: forward training and inverse estimation to solve the inverse problem in complex-valued network inversion. In the training phase, we provide the training input \( x = x_r + ix_i \) and training output \( y = y_r + iy_i \), and calculate the output error \( E = E_r + iE_i \) where \( i \) is the imaginary unit. Then, the weight \( w = w_r + iw_i \) is updated by

\[
w_{x}(t+1) = w_{x}(t) - \varepsilon_{w} \left( \frac{\partial E_r}{\partial w_x} + \frac{\partial E_i}{\partial w_x} \right), \quad w_{i}(t+1) = w_{i}(t) - \varepsilon_{w} \left( \frac{\partial E_r}{\partial w_i} + \frac{\partial E_i}{\partial w_i} \right)
\]

where \( \varepsilon_{w} \) represents the training gain, because the output error is due to the misadjustment of weights. By repeating this update procedure, the forward relation is obtained in the distribution of weights. This is a procedure based on the usual back-propagation method. In the inverse estimation phase, we fix the relation obtained in the training, provide the random input \( x = x_r + ix_i \) and the test output \( y = y_r + iy_i \), and calculate the output error \( E = E_r + iE_i \). Then, the input \( x = x_r + ix_i \), is updated by

\[
x_{x}(t+1) = x_{x}(t) - \varepsilon_{x} \left( \frac{\partial E_r}{\partial x_x} + \frac{\partial E_i}{\partial x_x} \right), \quad x_{i}(t+1) = x_{i}(t) - \varepsilon_{x} \left( \frac{\partial E_r}{\partial x_i} + \frac{\partial E_i}{\partial x_i} \right)
\]

where \( \varepsilon_{x} \) represents the input update gain, because the output error is due to the error of the input. By repeating this update procedure, the input is estimated from the output.

As an actual example, we consider the training and inverse estimation procedures of a three-layer complex-valued neural network shown in Figure 15. In this network, a complex-valued neuron is used for the hidden layer and the output layer. The complex input becomes a complex output through complex weights between each layer. The neuron uses a complex sigmoid function that applies the sigmoid function to the real and imaginary parts independently; this function is defined as follows:

\[
f_c(x) = f(x_r) + if(x_i), \quad f(u) = \frac{1 - e^{-u}}{1 + e^{-u}}
\]

where \( s = s_r + is_i \) represents the weighted sum of the neuron input. This complex-valued neuron model, which is shown in Figure 16, is referred to as a real-imaginary-type activation function (Benvenuto & Piazza, 1992; Hirose,
Here, we consider the activation function of the hyperbolic tangent type \( f(u) = \frac{1 - e^{-u}}{1 + e^{-u}} \) for the real and imaginary parts. But other activation functions, e.g., \( f(u) = \frac{1}{1 + e^{-u}} \), are also available for complex-valued network inversion (Ogawa & Kanada, 2005a; 2005b).

**Training Phase**

The training of the network is done in the same way employed for a usual complex-valued network. The output error \( E = E_R + iE_I \) is defined by the squared error

\[
E = \frac{1}{2} \sum_r \left( (y_r - y'_r)^2 + (E_I)^2 \right), \quad E_I = \frac{1}{2} \sum_r \left( (y_r' - y'_r)^2 \right),
\]

(13)
where \( y'_i = y'_{i+} + y'_{i-} \) and \( y_i = y_{i+} + y_{i-} \) are \( r \)-th tutorial output and network output. First, the weight update procedure between the hidden and output layers is formulated. The error signal from the output layer is calculated by

\[
\delta_{oi} = (y'_{oi} - y_{oi})(1 - y_{oi}) + y_{oi}(1 + y_{oi}) \quad \delta_{oi} = (y'_{oi} - y_{oi})(1 - y_{oi})(1 + y_{oi})
\]

which represents the gradient of the output error for the weight \( w_{oi} = w_{oi+} + iw_{oi-} \) between the hidden and output layer. Therefore, the weights are updated by

\[
w_{oi}(t+1) = w_{oi}(t) - \varepsilon_i(\delta_{oi}v_{oi} + \delta_{oi}v_{oi}) \quad \delta_{oi} = (y'_{oi} - y_{oi})(1 - y_{oi})(1 + y_{oi})
\]

where \( v_{oi} = v_{oi+} + iv_{oi-} \) and \( \varepsilon_i \) mean the input from \( k \)-th hidden neuron and a training gain, respectively.

Next, the weight's update procedure between input and hidden layer is formulated. The error signal from the hidden layer is calculated by

\[
\delta_{ih} = (1 - v_{ih})(1 + v_{ih}) \sum (\delta_{oh}w_{oh} + \delta_{oh}w_{oh}) \quad \delta_{ih} = (1 - v_{ih})(1 + v_{ih}) \sum (\delta_{oh}w_{oh} - \delta_{oh}w_{oh})
\]

which represents the gradient of the output error for the weights \( w_{ih} = w_{ih+} + iw_{ih-} \) between the input and hidden layer. Therefore, the weights are updated by

\[
w_{ih}(t+1) = w_{ih}(t) - \varepsilon_i(\delta_{ih}v_{ih} + \delta_{ih}v_{ih}) \quad \delta_{ih} = (1 - v_{ih})(1 + v_{ih}) \sum (\delta_{oh}w_{oh} - \delta_{oh}w_{oh})
\]

where \( \varepsilon_i \) is a training gain. The input-output relation is learned by correcting each complex weight according to the above equations.

**Inverse Estimation Phase**

In the inverse estimation phase, the input is estimated from the provided output. In other words, the provided initial random input is repeatedly updated by the output error that is back-propagated to the input via the fixed weights. This procedure is similar to error back-propagation training.

The output error for the input is defined similar to eqn. (13). In addition, the error signals from the output layer to hidden layer are formulated similar to eqns. (14) and (16). The error signal to the input layer is calculated by

\[
\delta_{xa} = (1 - x_{a})(1 + x_{a}) \sum (\delta_{oh}w_{oh} + \delta_{oh}w_{oh}) \quad \delta_{xa} = (1 - x_{a})(1 + x_{a}) \sum (\delta_{oh}w_{oh} - \delta_{oh}w_{oh})
\]

Then, the input can be updated by the error signal, which is expressed by eqn. (18), instead of the weight update, which is described by

\[
x_{a}(t+1) = x_{a}(t) - \varepsilon_i \delta_{xa}, \quad x_{a}(t+1) = x_{a}(t) - \varepsilon_i \delta_{xa}
\]

where \( \varepsilon_i \) is the inverse estimation gain. Thus, the input is corrected iteratively. When the error reaches the target, the input correction is terminated and the obtained complex input becomes a solution.

As a result, a complex input can be inversely estimated from a complex output by using the complex weight distribution obtained by training. This is similar to correcting the weights or the input iteratively, during training and inverse estimation. However, inverse estimation is iterative correction for a provided pattern, and it differs from training by repeated correction for plural patterns.

**Regularization**

In this section, we explain the introduction of regularization to complex-valued network inversion for solving ill-posed inverse problems. We examine static and dynamic regularization.
The error function of a complex-valued network inversion with static regularization is expressed by
\[
E_x = \frac{1}{2} \sum y'_a - y_a e^2 + \lambda \| f_x \|^2, \quad E_y = \frac{1}{2} \sum (y'_a - y_a) e^2 + \lambda \| f_y \|^2, \tag{20}
\]
where the first and second terms in each equation represent the usual error term and the regularization term. In this study, we use the regularization term defined as
\[
\| f_x \|^2 = \frac{1}{2} \sum x^2, \quad \| f_y \|^2 = \frac{1}{2} \sum y^2. \tag{21}
\]

This regularization serves to minimize the input value. The input of complex-valued network inversion with regularization is also updated based on eqn. (19). In contrast to real-valued network inversion, the input must be updated in both real and imaginary parts during complex-valued network inversion, which is expressed by
\[
x_{nR}(t + 1) = x_{nR}(t) - \varepsilon_n \delta_{nR} - \lambda x_{nR}(t), \quad x_{nI}(t + 1) = x_{nI}(t) - \varepsilon_n \delta_{nI} - \lambda x_{nI}(t). \tag{22}
\]
Ill-posedness in the complex domain can be reduced by the iterative correction of the input by eqn. (22). While the regularization parameter is constant in static regularization, it changes as a function of the epoch number \( t \) in dynamic regularization. The regularization parameter is defined as
\[
\lambda(t) = \lambda(0) \exp(-mt) \tag{23}
\]
where \( m \) is the decay coefficient. The parameter \( \lambda(t) \) decays from \( \lambda(0) \) to zero with the epoch number \( t \).

**SAMPLE PROBLEMS FOR COMPLEX-VALUED NETWORK INVERSION**

The first objective of this section is to demonstrate the complex-valued network inversion method to solve complex-valued inverse problems. To attain this objective, two problems—inverse estimation of complex-valued mapping and inverse Fourier transform—are examined. The second objective is to show the effect of the regularization method. To attain this objective, inverse estimation of complex-valued mapping with ill-posedness is examined.

**Example: Inverse Estimation of Complex-Valued Mapping**
The inverse complex-valued mapping problem is examined as a simple example of the inverse problem. In this problem, the mapping between the points on the complex plane is learned by a complex-valued neural network. Then, the point before mapping is estimated from the given point after mapping by using learned mapping inversely. In other words, the inverse mapping of an arbitrary point on the complex plane is estimated by inversely using the forward mapping obtained in the training. The network is assumed to have an input and an output.

First, we examine the inverse use of enlargement mapping. In the training phase, tutorial input and output patterns are provided to the network. The tutorial patterns are 11 points that satisfy the conditions \( y_a = 3x/a, x_{nR} = x_{nI} = \{-0.5, -0.4, \ldots, 0.4, 0.5\} \) for the input \( x_{nR} = x_{nI} + ix_{nI} \) and output \( y_{nR} = 3x_{nR} + iy_{nI} \). In other words, the network learns the relation mapped to 3/2 times coordinates, which implies enlargement mapping in the direction of the real and imaginary axes. In the inverse estimation phase, the weights obtained in the training phase are fixed to estimate the complex input from a complex test output. The complex test output patterns are 36 points that satisfy the condition \( y_{nR}^2 + y_{nI}^2 = 0.75^2 \). To provide the test output patterns, the corresponding input patterns are inversely estimated by the network. Because the inverse relation of the tutorial patterns results in reduction mapping, the input patterns to be estimated are expected to satisfy the condition \( y_{nR}^2 + y_{nI}^2 = 0.5^2 \).

In addition, we examine the inverse use of rotation mapping. The tutorial patterns are 11 points that satisfy the conditions \( y_{nR} = x_{nR}/2, x_{nI} = y_{nI} = \{-1.0, -0.8, \ldots, 0.8, 1.0\} \) for the input \( x_{nR} = x_{nI} + ix_{nI} \) and output \( y_{nR} = y_{nR} + iy_{nI} \). In other words, the network learns the relation mapped to 1/2 times in imaginary coordinates, which implies rotation mapping. In the inverse estimation phase, the complex test output patterns are 36 points that satisfy the
Complex-Valued Neural Network and Inverse Problems

To provide the test output patterns, the estimated input points are expected to lie on an inversely rotated ellipse, because the inverse relation of the tutorial patterns results in rotation mapping.

The network architecture and the allocation of the input and output patterns for training and inverse estimation are shown in Figure 17. The parameters of the network are shown in Table 2.

First, we show the simulation result of the inverse use of enlargement mapping. The plots of the tutorial and test input and outputs are shown in Figure 18 as the simulation result. The inverse estimated inputs are distributed as points on a circle of radius 0.5, and the test outputs are distributed as points on a circle of radius 0.75, according to the inverse estimated result of the input. This indicates that the input was correctly estimated from the test output by using the relation between the training input and output inversely. In other words, the reduction mapping of $2/3$ is achieved by using an enlargement mapping of $3/2$ inversely. Next, we show the simulation result of the inverse use of rotation mapping. The plots of the tutorial and test input and output are shown in Figure 19. The inverse estimated inputs are distributed as points on a rotated ellipse. Inverse rotation mapping is realized by using learned rotation mapping. To show the convergence of learning and inverse estimation, the learning and inverse estimation errors of the enlargement mapping are shown in Figure 20.

The fundamental inverse estimation operation of complex-valued network inversion was confirmed from these simulation results. In this method, a complex-valued input is estimated from the provided complex-valued output by using the trained forward complex-valued relation.

$y_{nt}^2 + 4y_{nt}^2 = 0.5^2$. Condition $y_{nt}^2 + 4y_{nt}^2 = 0.5^2$. To provide the test output patterns, the estimated input points are expected to lie on an inversely rotated ellipse, because the inverse relation of the tutorial patterns results in rotation mapping.

The network architecture and the allocation of the input and output patterns for training and inverse estimation are shown in Figure 17. The parameters of the network are shown in Table 2.

First, we show the simulation result of the inverse use of enlargement mapping. The plots of the tutorial and test input and outputs are shown in Figure 18 as the simulation result. The inverse estimated inputs are distributed as points on a circle of radius 0.5, and the test outputs are distributed as points on a circle of radius 0.75, according to the inverse estimated result of the input. This indicates that the input was correctly estimated from the test output by using the relation between the training input and output inversely. In other words, the reduction mapping of $2/3$ is achieved by using an enlargement mapping of $3/2$ inversely. Next, we show the simulation result of the inverse use of rotation mapping. The plots of the tutorial and test input and output are shown in Figure 19. The inverse estimated inputs are distributed as points on a rotated ellipse. Inverse rotation mapping is realized by using learned rotation mapping. To show the convergence of learning and inverse estimation, the learning and inverse estimation errors of the enlargement mapping are shown in Figure 20.

The fundamental inverse estimation operation of complex-valued network inversion was confirmed from these simulation results. In this method, a complex-valued input is estimated from the provided complex-valued output by using the trained forward complex-valued relation.

Table 2. Network parameters for the inverse estimation problem of complex mapping

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of input neurons</td>
<td>1</td>
</tr>
<tr>
<td>Number of hidden neurons</td>
<td>5</td>
</tr>
<tr>
<td>Number of output neurons</td>
<td>1</td>
</tr>
<tr>
<td>Training gain $\varepsilon_t$</td>
<td>0.01</td>
</tr>
<tr>
<td>Estimation gain $\varepsilon_e$</td>
<td>0.01</td>
</tr>
<tr>
<td>Final training error level</td>
<td>0.0001</td>
</tr>
<tr>
<td>Final estimation error level</td>
<td>0.001</td>
</tr>
<tr>
<td>Maximum number of training epochs</td>
<td>20000</td>
</tr>
<tr>
<td>Maximum number of estimation epochs</td>
<td>10000</td>
</tr>
<tr>
<td>Number of training sets</td>
<td>11</td>
</tr>
<tr>
<td>Number of test sets</td>
<td>36</td>
</tr>
</tbody>
</table>

Figure 17. Network architecture for the inverse estimation problem of complex-valued mapping
Example: Inverse Fourier Transform Problem
The inverse Fourier transform is solved by complex-valued network inversion by considering it as an inverse
problem. Fourier transform and its inversion are considered to be a mapping between a signal and a complex-
valued spectrum. The mapping from the signal to spectrum is learned by a complex-valued neural network. Then,
the signal is estimated from the given spectrum by using the learned mapping inversely. In fact, we consider the
discrete Fourier transform (DFT) and its inversion. The signal and the spectrum in DFT are assumed to be input
and output, respectively, and the relation is learned by the network. Using the network learned by complex-valued
network inversion, the signal is inversely estimated from the given spectrum. In learning phase, the network is
studied by nine kinds of sinusoidal signal input and their corresponding spectrum outputs. While the weights
obtained in learning are fixed for inverse estimation, the input is estimated from the test output pattern of the
complex-valued spectrum.
Complex-Valued Neural Network and Inverse Problems

The inverse Fourier transform problem is examined to show the procedure of the complex-valued network inversion. Though the significance as a practical problem are not so large, the inverse Fourier transform problem is more concrete than the complex mapping problem as an example with the complex-valued input and output. It is possible to enhance the solution of the inverse Fourier transform problem to nonlinear mappings between the complex-valued input and output.

In the simulation, we use the complex-valued network with eight inputs and eight outputs. The input and output are the waveform and its spectrum, respectively. Both waveform and spectrum are sampled at eight points to be provided to the network. After several-time trials, we decided the number of hidden layer neurons. Because the number of training sets is small, we used the smaller number of hidden layer neurons. The tutorial inputs for the network training are the sinusoidal waveforms expressed by

\[ x_n = a \cos \left(2\pi \frac{n}{N} b - \frac{\pi}{4} c\right), \quad (b = 1, 2, 3, c = 0, 1, 2) \]  \hspace{1cm} (24)

where \(a\) and \(N\) are the amplitude and number of sampling points, respectively. The input waveforms are magnified and translated versions of the sinusoidal waveform \(x_n = a \cos (2\pi n/N), \ (a = 0.8, N = 8, n = 0, 1, \ldots, 7)\). The tutorial outputs are the complex-valued spectrum \(y_n = F(n, \omega), \ (n = 0, \ldots, 7)\) of the input waveforms. In the inverse estimation phase, a random input pattern is given to the input layer while fixing the weights obtained in the training phase, and the input is iteratively corrected from the given test output pattern (complex-valued spectrum). The network parameters are summarized in Table 3. The network architecture is shown in Figure 21.

The three estimated input patterns—\(x_n = a \cos (2\pi n/N), x_n = a \cos (4\pi n/N-\pi/2),\) and \(x_n = a \cos (6\pi n/N-\pi/2)\)—are shown in Figure 22. These are three of nine training patterns shown in eqn. (24). According to the results, the original shape of the waves is almost estimated for a given test output of the complex-valued spectrum. In the
signal with different phases and same cycle, the ratio of the imaginary part to the real part of the spectrum is different. The signals with different phases can be estimated by complex-valued network inversion. The reason for this result is that the crossover among the input and the weights was correctly realized by the complex-valued neural network. Consequently, the inverse estimation of the input from a given output by complex-valued network inversion can be confirmed by the inverse Fourier transform of the signals corresponding to the given complex-valued spectrum.

In general, the estimated result is not guaranteed for untrained data in network inversion. However, inverse estimation is performed on test data that are distributed on a complex plane for training data distributed within a limited range in the above-mentioned result. In other words, it seems that inverse estimation is carried out from untrained data. This feature is explained with relation to the identity theorem in complex analysis. However, it is necessary that training data are distributed over the entire area in the real and imaginary parts. The inverse estimation of the input by complex network inversion was confirmed from these results. In this section, two kinds of conversion were examined for testing complex-valued network inversion. It is necessary to confirm various conversion abilities to verify the feature of complex-valued network inversion.

In the training phase, the network learns the relation between the input and output from the provided tutorial input and output patterns. The relations to be learned are enlargement, reduction in real and imaginary axis, and rotation at the origin. In the inverse estimation phase, the complex-valued input is estimated from the provided complex-valued output while fixing the complex-valued weights obtained in the training phase. After the network
Figure 22. Simulation results of inverse Fourier transform. (a) $x_n = a \cos \left( \frac{2\pi n}{N} \right)$, (b) $x_n = a \cos \left( \frac{4\pi n}{N} - \frac{\pi}{2} \right)$, (c) $x_n = a \cos \left( \frac{6\pi n}{N} - \frac{\pi}{2} \right)$
learns the forward relation from the input to the output, it estimates the corresponding input from the given test output by using the relation inversely. According to the result of inverse estimation, we obtained an almost correct input value. This means that the input was correctly estimated from the given test output by using the forward relation between the training input and output inversely. Inverse estimation of the input by complex-valued network inversion was confirmed by these simulations.

Example: Inverse estimation of Complex-Valued Mapping with Ill-Posedness
We examined complex-valued network inversion with regularization to solve the ill-posed inverse complex mapping problem. In this problem, the mapping between the points on the complex plane is learned by the complex-valued neural network. The complex-valued mapping concerning the absolute value is inversely estimated as an ill-posed inverse problem. To demonstrate the effect of regularization, three inverse estimations by complex-valued network inversion were examined. These estimations were (i) without regularization, (ii) with regularization directed to the maximum value, and (iii) with regularization directed to the minimum value. Here, we use the dynamic regularization method, which dynamically changes the coefficient, because it provides an advantage in that parameter setting of regularization is easier in dynamic regularization than in static regularization. Consequently, the solution that satisfies a specific condition is estimated almost correctly by this regularization.

In the training phase, the tutorial input and output patterns are provided to the network. The tutorial patterns are 31 points that satisfy the conditions $x_nR = x_nI$, $y_nR = x_nR$, and $y_nI = |x_nI|$ for the input $x_n = x_nR + ix_nI$ and output $y_n = y_nR + iy_nI$. In the inverse estimation phase, the weights obtained in the training phase are fixed to estimate the complex input from the complex test output. Both the real and imaginary parts of the initial input are set to random values from $-1.0$ to $1.0$. The complex test output patterns are the same as those for the tutorial input. The input and output patterns for training are shown in Figure 23. The output patterns are also used for inverse estimation. In inverse estimation, there are two possible answers for data in the second and third quadrants, because the input is estimated as an inversion of the absolute value. Therefore, this problem contains the non-uniqueness of ill-posedness. The network architecture is the same as that shown in Figure 17. The parameters of the network are shown in Table 4.

The plot of the estimated input is shown in Figure 24 as the simulation result. The input estimated without regularization is shown in Figure 24 (a). Two correct solutions exist in the area of $y_R < 0$ while the solution is decided uniquely in the area of $y_R > 0$, where $y_R$ is the value of the real part. The estimated inputs divide in the solutions of $y_I > 0$ and $y_I < 0$ and settle in the area of $y_R < 0$. It is considered that this division of solutions depends on the value of random numbers given as input initial values in complex-valued network inversion. On the other
hand, the input estimated by adding the regularization is shown in Figure 24 (b) and (c). The former is regularization directed at the maximum value, and it is expressed by

\[
\begin{align*}
x_{at}(t+1) &= x_{at}(t) - \varepsilon_x \cdot \delta_{at} - \lambda x_{at}(t), \quad x_{at}(t+1) = x_{at}(t) - \varepsilon_x \cdot \delta_{at} - \lambda x_{at}(t). 
\end{align*}
\]

The latter is regularization directed at the minimum value,

\[
\begin{align*}
x_{at}(t+1) &= x_{at}(t) + \varepsilon_x \cdot \delta_{at} + \lambda x_{at}(t), \quad x_{at}(t+1) = x_{at}(t) + \varepsilon_x \cdot \delta_{at} + \lambda x_{at}(t),
\end{align*}
\]

where \( \varepsilon_x \) and \( \lambda \) represent the input correction coefficient and regularization coefficient, respectively.

According to these results, we can estimate the directed solution from the possible solutions by complex-valued network inversion with regularization. In other words, regularization solves ill-posed inverse problems with regard to uniqueness. The effectiveness of the regularization method in the complex-valued network inversion was depicted from this result. The effect on ill-posedness concerning existence and stability is currently under research. We expect that the problem of existence is solved by the generalization ability of the neural network. Moreover, we consider that the problem of stability is reduced by regularization that minimizes the fluctuations of the solution.

As described previously, the practical significance of network inversion is in the solution of ill-posed inverse problems. In the case of a well-posed problem with one-to-one correspondence between the input and output, we can estimate the inputs by the inverse solver network that reverses the input and the output. In other words, we can solve a well-posed inverse problem by forward learning and forward estimation. However, the inverse solver network does not solve ill-posed problems with many-to-one correspondence by forward learning and forward estimation, because the learning itself is unsuccessful.

To depict this feature, we consider three training data sets with one-to-one, many-to-one, and one-to-many correspondences of the complex-valued mapping problem. The network is assumed to have one input and one output, and the parameters are shown in Table 5. The first data set indicates a well-posed problem, which can be solved by the inverse solver network with reversed input and output. The second one indicates a problem that

<table>
<thead>
<tr>
<th>Table 4. Network parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Number of input neurons</td>
</tr>
<tr>
<td>Number of hidden neurons</td>
</tr>
<tr>
<td>Number of output neurons</td>
</tr>
<tr>
<td>Training gain ( \varepsilon_t )</td>
</tr>
<tr>
<td>Estimation gain ( \varepsilon_e )</td>
</tr>
<tr>
<td>Final training error level</td>
</tr>
<tr>
<td>Final estimation error level</td>
</tr>
<tr>
<td>Maximum number of training epochs</td>
</tr>
<tr>
<td>Maximum number of estimation epochs</td>
</tr>
<tr>
<td>Number of training sets</td>
</tr>
<tr>
<td>Number of test sets</td>
</tr>
<tr>
<td>Regularization coefficient ( \lambda )</td>
</tr>
<tr>
<td>Decay coefficient ( m )</td>
</tr>
</tbody>
</table>
Figure 24. Estimated results of complex-valued mapping with ill-posedness. (a) Without regularization, (b) with regularization for positive direction, (c) with regularization for negative direction.
Complex-Valued Neural Network and Inverse Problems

becomes ill-posed in inverse estimation. The forward training of this problem is to be confirmed. The third one indicates forward training for solving the ill-posed problem by the inverse solver network. These three data sets used in the simulation are shown in Table 6.

The error curve obtained when these data are learned is shown in Figure 25. The input-output relation is learned sufficiently in the first and second data sets because the error decreases. From the result of the first data set, this implies that the forward relation of the well-posed problem is correctly learned. In addition, the forward relation of the ill-posed problem is correctly learned from the result of the second data set. On the other hand, the inverse input-output relation cannot be learned because the error does not decrease in the third data sets. This result indicates that the inverse solver network cannot handle the ill-posed inverse problem. Consequently, the inverse solver network in which the input-output relation was reversed cannot solve the ill-posed inverse problem because the learning does not succeed; however, the problem can be solved by network inversion with appropriate regularization.

Because the inverse problem can be solved by the inverse solver network if data without ill-posedness are prepared, the practical significance of network inversion is reduced. However, it is difficult to exclude ill-posedness completely in the general problem. Therefore, it is considered that there is practical significance of network inversion as well as complex-valued network inversion.

CONCLUSION

In this chapter, we introduced a method involving a complex-valued neural network to solve inverse problems that extend to the complex domain. This method extends the network inversion method to the complex domain by solving inverse problems on a multilayer neural network. In this method, the complex-valued input is estimated from the provided complex-valued output by using a trained complex-valued multilayer neural network. Actually,
this method estimates the complex-valued input by propagating the complex-valued output error inversely and by correcting the complex-valued input value repeatedly.

An inverse problem, which is one by which the cause is estimated from the given result, is an important problem in the field of science and engineering fields. In this chapter, we explained the inverse problems as a background and introduced two examples: nerve bundle localization by evoked potentials and source localization using a bat-like sonar system. Next, we explained the principles of network inversion and regularization. The main focus of this chapter is on the introduction of the complex-valued network inversion method and the explanation of the regularization method for reducing ill-posedness. To confirm the procedure of complex-valued network inversion, we carried out two simulations: inverse estimation of complex-valued mapping and inverse Fourier transform. In both simulations, inverse estimated results with almost correct complex-valued inputs obtained from complex-valued outputs were observed, and the operation of complex-valued network inversion was confirmed according to the results of each simulation.

Moreover, complex-valued network inversion with regularization was applied to the ill-posed inverse problem of complex-valued mapping. The solution was confirmed to converge to a directed solution by the regularization method. Therefore, the effectiveness of the complex-valued network inversion method was confirmed. As future problems, we will consider applying complex-valued network inversion with regularization to actual engineering problems such as medical signal analysis and image recognition. Moreover, we will examine the effects of reducing ill-posedness by various regularizations.

**FUTURE RESEARCH DIRECTIONS**

In this chapter, the solution of inverse problems by complex-valued network inversion was clarified. In addition, the effect of the addition of regularization to solve ill-posedness, which poses a difficulty in actual inverse problems, was examined and illustrated by simulation.

We are considering three directions of future research. The first one involves studying the characteristics of the complex-valued network inversion thoroughly and researching its feature to solve inverse problems theoretically. The purpose of this would be to examine possible conversion and mapping by complex-valued network
inversion, and to examine the characteristics experimentally and theoretically. As a result, it is necessary to clarify theoretical and methodological features of complex-valued network inversion. The second study involves applying complex-valued network inversion to actual problems. For example, we believe that this inversion can be applied to actual problems such as image processing and sonar signal analysis. In this case, because the problem of ill-posedness often arises in the actual problem, it is necessary to develop a method of regularization that corresponds to an actual case and examine this method. We consider that enhancement of the answer-in-weights scheme to the complex domain is a solution to the problem. The third study involves examining various methods of regularization. It is necessary to examine the regularization functional, including theoretical effects, because various situations of a problem are possible. Moreover, we consider that the method of providing the restraint condition is effective according to a method such as the answer-in-weights scheme. We think that this line of research will help make the complex-valued network inversion method a step in the development of complex-valued neural networks.

REFERENCES


**ADDITIONAL READING**


