Neural Processing Letters (2022) 54:559–580
https://doi.org/10.1007/s11063-021-10644-1

Complex Valued Deep Neural Networks for Nonlinear System Modeling

Mario Lopez-Pacheco[1] · **Wen Yu**[1]

Accepted: 9 September 2021 / Published online: 23 September 2021
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2021

Abstract
Deep learning models, such as convolutional neural networks (CNN), have been successfully applied in pattern recognition and system identification recent years. But for the cases of missing data and big noises, CNN does not work well for dynamic system modeling. In this paper, complex valued convolution neural network (CVCNN) is presented for modeling nonlinear systems with large uncertainties. Novel training methods are proposed for CVCNN. Comparisons with other classical neural networks are made to show the advantages of the proposed methods.

Keywords Convolutional neural networks · Complex valued · System modeling

1 Introduction

Predicting future behaviors in dynamic systems is a complicated task. Many advanced control techniques could be used to achieve this goal. The system modeling can be done through different methods, applying the laws of physics that govern the behavior of the system is one of the most used [1]. In order to apply this method, it is necessary to have sufficient knowledge of the system to determine the parameters of physic equations, which in most cases is difficult to achieve. In general, modeling or identification of systems is to find relationships between the inputs and outputs of the system. These relationships can often be represented as a convolution operation, such as time-invariant linear systems. Time series forecasting methods can be used for the system identification, but the inputs and outputs of the system are joined into one time series, and the convolution relation lost [2].

Neural networks are black box models, because they only need to know input and output data. This data-driven modeling method still has problems. There are still not effective methods for determining the hyper-parameters of the neural network, such as the number of hidden layers and the number of nodes in each of the layers [3]. The universal approximation theorem ensures that a neural model can approximate virtually any continuous system using some number of nodes in one hidden layer. However, increasing the number of nodes in this

✉ Wen Yu
yuw@ctrl.cinvestav.mx

[1] Departamento de Control Automático, CINVESTAV-IPN, Ciudad de México, Mexico

layer leads to overfitting problem [4]. As an alternative, instead of increasing the number of nodes in the hidden layer, it is recommended to add more hidden layers. This is the deep neural networks [5]. In recent years the models of deep learning have been performed in many areas, and they archive better results than existing models theoretically and practically [6].

One of the most popular deep learning models is convolutional neural network (CNN) [7], which is used mainly in images processing. CNNs are used for image classification in [8–10]. The CNNs give almost the highest performances, and they have been applied in medical images [11,12]. There are also applications in other areas, such as time series prediction [13,14]. Many variation architectures of CNNs have been presented. [15] presents a CNN in the frequency domain, which differs from the classical CNNs from the 90's [3,16].

A very common problem of systems modeling are large uncertainties affecting them, such as unknown parts of the data, which are commonly up to 50% of total data, It can be produced by any cause, a fault, a human error, an accidental or provoked deletion, etc. Data loss is most often caused by physical failure of sensors, followed by human error. The others large uncertainties are the external disturbance,such as the measurement noise which can be around 20% of the output amplitude, this type of noise is generated mainly by a malfunction of the sensors used in the system and it can be periodic, intermittent, or random. Data processing can reduce these effects on the process. We can also reduce these influences through our neural networks, for example multilayer perceptions (MLP) can increase theirs nodes and layer numbers. But we have to train a greater number of parameters, and there is overfitting problem. CNNs can reduce noise directly in the convolutional layers. CNN are use more frequently in the others tasks than nonlinear dynamic system modeling [17].

System identification can be formed into time series prediction [18,19]. In [20], the time series is in the self-regulating model to improve performance. [21] presents a model based on neural networks to estimate COVID-19 cases in the following months. In [22] proposes a new LSTM network architecture for modeling dynamic systems. In this same area, very few publications are presented in which a CNN is used for this purpose. In [23], the identification of the nonlinear system is transferred to time series using ARMAX model. In our previous work [24], a modeling scheme for nonlinear systems is proposed using CNN with real values. In [25], CNN is trained to model uncertainties in the dynamic system. These uncertainties are over passed with the CNN because of the properties of share weights and sparse connectivity.

Most of deep learning algorithms work with real values, such as multilayer networks (MLP), support vector machine (SVM), long-short term memories (LSTM) and CNN. Complex valued neural networks have many advantage over classical NNs, such as complex valued MLP [26], complex valued RBF networks [27] and complex valued convolutional neural networks (CVCNN) [28,29].

In order to use complex numbers, the normal neural networks have to be modified. The modifications are in the structure of NNs and in learning methods [30]. A general training algorithm for the complex valued NN is introduced in [28]. In [31], the complex valued NN is regarded as two real valued NNs, which expand the learning information. It is more difficult to give theoretical analysis for complex values [32]. For image processing, CVCNN is given for image denoising [33] and for image classification [34]. In [35] CVNN is proposed as classifier based on range-beam-Doppler tensor. In [36] classification of human activities are proposed using CVCNN, where the motion of human are captured and transformed to frequency domain. To the best of our knowledge, there are not published results using CVCNN for dynamic system modeling [37].

From our point of view, the main advantage of the CVCNN is that we can have same structure as the real valued CNN, but for the cases of missing data and big noises, CVCNN

will give better performances. In order to create a neural network who can model nonlinear systems with missing data and large uncertainties with CNN, in this paper, we make the following contributions:

(1) Novel architecture of CVCNN for non linear dynamic system modeling is proposed.
(2) Learning method of the CVCNN is given.

To show the advantages of the CVCNN, comparisons with the other state-of-art methods are made by using several benchmarks. The comparison results show that: (1) the novel model has better modeling performances than the other algorithms in the cases of large uncertainties; (2) the proposed method has fast convergence speed, and it can be realized easily.

2 Nonlinear Dynamic System Modeling with Complex Valued CNN

2.1 Deep Nneural Networks for Dynamic System Modeling

Let consider an unknown discrete-time nonlinear system

$$\bar{x}(k+1) = f\left[\bar{x}\left(k\right), u\left(k\right)\right], \quad y(k) = g\left[\bar{x}\left(k\right)\right] \tag{1}$$

where $u\left(k\right)$ is the input vector, $\bar{x}\left(k\right)$ is the internal state vector, $y\left(k\right)$ is the output vector. f and g are general nonlinear smooth functions, $f, g \in C^{\infty}$. Denoting

$$Y(k) = \left[y\left(k\right), y\left(k+1\right), \cdots y\left(k+n-1\right)\right]$$
$$U(k) = \left[u\left(k\right), u\left(k+1\right), \cdots u\left(k+n-2\right)\right]$$

if $\frac{\partial Y}{\partial \bar{x}}$ is non-singular at $\bar{x} = 0$ and $U = 0$, leads to the following nonlinear auto regressive exogenous model (NARX)

$$y(k) = \Upsilon\left[x\left(k\right)\right] \tag{2}$$

where $\Upsilon\left(\cdot\right)$ represent an unknown nonlinear difference equation corresponding to the plant dynamics,

$$x\left(k\right) = \left[y\left(k-1\right), \cdots y\left(k-m_y\right), u\left(k\right), \cdots u\left(k-m_u\right)\right]^T \tag{3}$$

$x\left(k\right) = \left[x_1 \cdots x_l\right]^T$, $l = m_y + m_u + 1$, $k = 1 \cdots N$, N is the total data for this particular system.

To model the nonlinear system (2) two types of models can be used:

(1) The first type is

$$\begin{aligned} &\hat{y}(k) = N[u\left(k\right), \cdots, u\left(k-n_u\right)] \\ &\text{or } \hat{y}(k) = N[\hat{y}(k-1), \cdots, \hat{y}(k-n_y), \\ &u\left(k\right), \cdots, u\left(k-n_u\right)] \end{aligned} \tag{4}$$

where $\hat{y}(k)$ is the model output, $N\left[\cdot\right]$ is the complex valued CNN. n_y and n_u are the regression orders. These representation is called the parallel model [38]. Here the input to the CNN does not include the real output $y\left(k-1\right), \cdots$

(2) The second type is

$$\begin{aligned} &\hat{y}(k) = N[y\left(k-1\right), \cdots, y\left(k-n_y\right), \\ &u\left(k\right), \cdots, u\left(k-n_u\right)] \end{aligned} \tag{5}$$

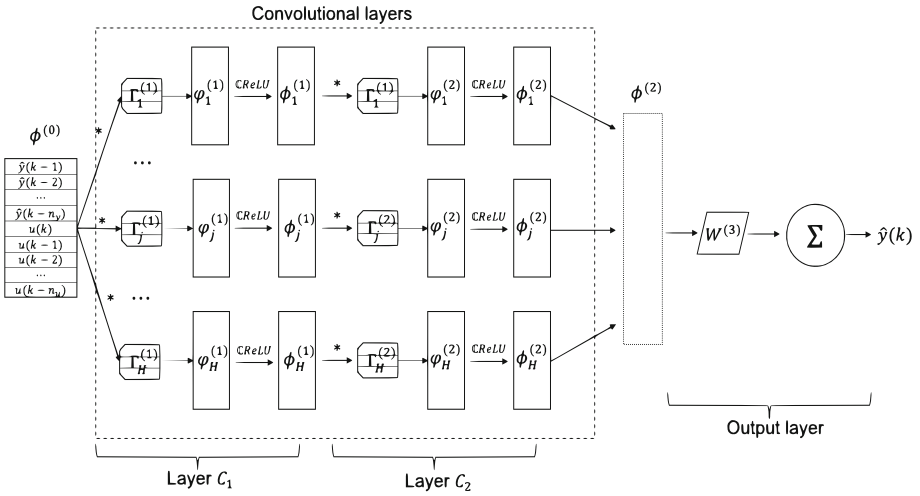

Fig. 1 Three-layer complex valued CNN

It is the series-parallel model [38]. The data regression is the same as the NARX model (3), only $n_u \neq m_u$, $n_y \neq m_y$.

Because the parallel model (4) do not use the output $y\,(k-i)$, the missing data and the disturbances in the output $y\,(k-i)$ does not affect the neural model. In this paper, we will use the parallel model (4).

The biggest difficult of the parallel model (4) is that it cannot model complex nonlinear system, such as missing data and large disturbances, this is because it is not enough to know only the input of the system. By not knowing the output data, even when the data is presented smoothly, the models used tend to diverge [39]. But we will use complex valued CNN to overcome this problem, the fact of adding the imaginary part to the network, leads to being able to extract different features of the system helping a better understanding by the network.

2.2 Complex Valued CNN

The architecture of the convolutional neural networks (CNN) for system identification is shown in Fig. 1. It is in cascade connection. Each CNN cell includes convolution and subsampling operations. The last layer is fully connected with synaptic weights. For complex valued CNN (CVCNN), all parameters (weights) are complex-value.

CVCNN is an efficient modeling option for nonlinear systems, because the convolution operation in CNN is the same as the input-output relationship of dynamic system [24], and the complex-based representation of the neural network increases the potential of the algorithm. The advantages of using CVCNN for modeling dynamic systems are:

(1) To model a complex nonlinear system, large-scale neural networks are needed. The CNN model uses sparse connectivity and shared weight to reduce the number of model parameters. This property is one of the main advantaged of real valued CNN. A CNN does not depend on the type of data used, but on the structure of the network.
(2) Each convolutional filter scans the entire data regardless of where the information is in the data. When acquiring features is in the complex domain, the created feature maps are much richer with system information, even the input data are real. Each of these

feature maps has some particular system property through the convolution operation with complex values, enlarge by the imaginary part, This can be seen as if more filters are added to the network, corresponding to the imaginary part of them, causing more features to be acquired from the data, providing more information for the modeling of the system [40].

(3) The multi-level grouping, such as the maximum operation on the sub sampling layer, has tolerance to the noise in the data. This makes CNN's model very robust. The calculations carried out in the convolutional layers by the CVCNN obtain more information from the physical system, this due to the fact that the filters have complex components, which allows different features to be generated when performing these operations, which implies that the probability of presenting over fitting, which is a very frequent problem in these types of methods [41].

(4) The activation function has the problem of gradient vanish [42], which occurs in deep neural networks. CVCNN also uses modified rectified linear units (ReLU) function to reduce this effect. With this activation function, the gradient passes through these layers without any alteration allowing the deep models to be more reliable. It also reduces the number of parameters to be updated in each iteration, since it is a conditional function which does not depend on any parameter, except for the input to function.

2.3 Complex Valued CNN for Dynamic System Modeling

To model complex nonlinear systems, we design the following operations for the complex valued CNN.

(1) Convolutional operation. The parameters of the complex valued CNN, such as the filters and the synaptic weights, are complex values The convolution operation of CVCNN is modified as

$$\varphi_j^{(\ell)} = \phi_j^{(\ell-1)} * \Gamma_j^{(\ell)} \tag{6}$$

where φ is the $j-$th feature map in the layer ℓ, produced by the convolution of the input $\phi^{(\ell-1)}$, $\Gamma_j^{(\ell)}$ is the filter in current layer, $\ell = 1, 2, \cdots, m$, m is the number of convolutional layers. In each layer, there are n filters with the length f_ℓ. Each convolutional layer is denoted by C_ℓ.
In the first layer, $\phi^{(\ell-1)} = \phi^{(0)}$. It is the input to the CVCNN. Because convolution is a combined operation, the sum of the product can also be interpreted as a point product between two vectors.

(2) Activation function. After the feature maps are created by the convolutional layer, an activation function is applied. For the complex valued CNN, the ReLU function is modified as $\mathbb{C}ReLU$ function,

$$\begin{aligned} &\mathbb{C}\text{ReLU}(x) = \text{ReLU}(x_\Re) + i\text{ReLU}(x_\Im) \\ &x = x_\Re + ix_\Im \in \mathbb{C} \end{aligned} \tag{7}$$

The $\mathbb{C}ReLU$ satisfies the Cauchy-Riemann equations, when both real and imaginary are strictly positives or negatives. This condition is also satisfied when $\theta_x \in \left[0, \frac{\pi}{2}\right]$ or $\theta_x = \in \left[\pi, \frac{3}{2}\pi\right]$, $\theta_x = arg(x)$, θ_x is the phase or argument of the complex number x.
Applying the $\mathbb{C}ReLU$ activation function to the feature map (6),

$$\phi_j^{(\ell)} = \mathbb{C}ReLU(\varphi_j^{\ell}) \tag{8}$$

where $\phi_j^{(\ell)}$ is the output of the current convolutional layer.

(3) Pooling. An additional layer is included to the CVCNN for nonlinear dynamic system modeling. This layer is called pooling layer. Because the complex number has more properties than the real number, the pooling operations, such as maximum grouping and average grouping, have different meanings. Because of cycling property, we cannot only evaluate complex numbers by their phase. Magnitude is a more reliable property in the pooling, which is defined by

$$M(x) = |x| \quad M : \mathbb{C} \to \Re \tag{9}$$

The pooling uses the magnitude of the complex number as

$$\arg \max_{x \in set} M(x) \tag{10}$$

(10) allows us to maintain a complex number through the complex valued CNN, because the output of the max-by-magnitude grouping belongs to the same input set.

(4) Full connection. The output layer of the CVCNN is a fully connected layer with synaptic weights $W^{(\ell)} \in \mathbb{C}^n$. The output of the CVCNN is defined as:

$$\hat{y}(k) = W^{(\ell)} \phi^{(\ell-1)} \tag{11}$$

3 Complex Valued CNN Training

The normal training algorithms of CNN cannot be applied to CVCNN, because:

(1) The ReLU of CNN is modified as $\mathbb{C}ReLU$ (7), which uses complex value. The back-propagation operation has to be changed in complex domain.
(2) The convolution operation in complex domain can be only applied to the real counterpart. The image part has to be redefined to obtain more information from the data.
(3) CNN uses the gradient descent algorithm to update the parameters. But for CVCNN, the gradient descent algorithm has to be derived in complex domain.

In this paper, we propose a novel training method of CVCNN for system identification.

3.1 Backpropagation in Complex Domain

The object of nonlinear dynamic system modeling using the complex valued deep neural networks (11) is to update the weights $W^{(\ell)}$, such that the output of the CNN $\hat{y}(k)$ convergence to the system output $y(k)$ in (2),

$$\arg \min_{W^{(\ell)}, V} \sum_{k=1}^{N} \left[\hat{y}(k) - y(k)\right]^2, \quad \ell = 1, 2, \cdots, m$$

where N the training data number.

In this paper, we use the stochastic gradient descent (SGD). The object is

$$\arg \min_{W^{(\ell)}, V} \left[\hat{y}(k) - y(k)\right]^2$$

where $\ell = 1, 2, \cdots, m$, $k = 1, 2, \cdots, N$.

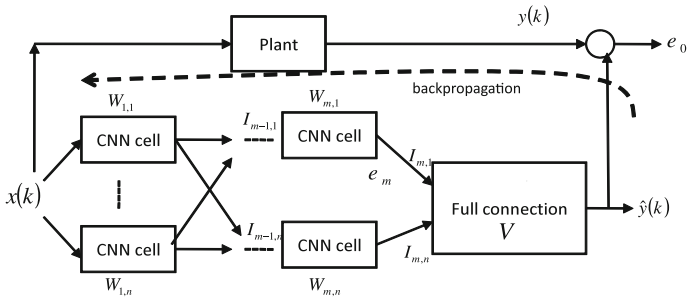

Fig. 2 The hierarchical structure of CNN for system identification

The performance index is defined as

$$J(k) = \frac{1}{2} e_o(k)^2, \quad e_o(k) = \hat{y}(k) - y(k) \tag{12}$$

where $e_o(k)$ is the modeling error.

There are m convolutional layers, each layer has n filters. So there are $m \times n$ CNN cells. We should update the weigh of each filter $W^{(\ell)} \in \mathbb{C}^n$ and the weight in the full connection layer V, see Fig. 2.

We first discuss the gradient descent algorithm in complex domain. For real values, the weights in the output layer is updated as

$$V(k+1) = V(k) - \eta \phi^{(\ell)} e_o(k) \tag{13}$$

where $\phi_m^{(\ell)}(k) = \left[\phi_{m,1} \cdots \phi_{m,n}\right]^T$ is the output of the convolution operation of the CNN model (11), $\eta > 0$ is the learning rate.

For complex values, we need to calculate the gradient descent of a complex function, which is defined as $f : \mathbb{C} \to \mathbb{C}$.. The derivatives of complex valued function can be regard as the derivative of a multivariate function, defined as $f : \Re^2 \to \Re^2$.

If $f(x)$ and $f(x, y)$ are differentiable at points $a \in \Re^1$ and $(a, b) \in \Re^2$,

$$\begin{aligned} &\lim_{x \to \alpha} \frac{f(x) - f(\alpha)}{x - \alpha} = \beta \\ &\lim_{(x,y) \to (a,b)} \frac{\|f(x, y) - f(a, b) - F(x - a, y - b)\|}{\|(x - a, y - b)\|} = 0 \end{aligned} \tag{14}$$

where $x, \alpha, \beta \in \Re$.

If $f(x, y) = f[u(x, y), v(x, y)]$ and the derivative of f is denoted by Df,

$$Df = \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix} \tag{15}$$

From Cauchy-Riemann Theorem [43]. A complex function $f(x) = r(x) + is(x)$, $r, s : \Re \to \Re$ is differentiable if and only if $f(x_\Re, x_\Im) = (r(x_\Re, x_\Im), s(x_\Re, x_\Im))$ is differentiable in $\Re^2 \to \Re^2$, and its partial derivatives satisfy

$$\frac{\partial r}{\partial x_\Re} = \frac{\partial s}{\partial x_\Im}, \quad \frac{\partial s}{\partial x_\Re} = -\frac{\partial r}{\partial x_\Im} \tag{16}$$

At the point $a + ib$, the Wirtinger derive is

$$\lim_{x_\Re + ix_\Im \to a+ib} \frac{f(x_\Re + ix_\Im) - f(a+ib)}{x_\Re + ix_\Im - (a+ib)} = \varsigma_\Re + i\varsigma_{Im} \tag{17}$$

where $x_\Re$ and $x_\Im$ are the variables in real and complex parts. The complex number $\varsigma = \varsigma_\Re + i\varsigma_{Im}$. (17) becomes

$$\lim_{\substack{x_\Re + ix_\Im \\ \to a+ib}} \frac{\left[\begin{array}{l} f(x_\Re + ix_\Im) - f(a+ib) \\ -(\varsigma_\Re + i\varsigma_{Im})(x_\Re + ix_\Im - (a+ib)) \end{array}\right]}{x_\Re + ix_\Im - (a+ib)} = 0 \tag{18}$$

So the complex derivative of function f can be regarded as a linear transformation, which is equivalent to the derivative of the function in $\Re^2 \to \Re^2$. The complex derivatives are special linear functions with orthogonal matrices and with positive determinant.

The differential of a complex valued function $f(x) : S \to \mathbb{C}, S \subseteq \mathbb{C}$ can be expressed by

$$df = \frac{\partial f(x)}{\partial x} dx + i \frac{\partial f(x)}{\partial x^*} dx^* \tag{19}$$

where x^* is in complex domain. Here its derivative of real function has a orthogonal matrix with positive determinant.

In complex domain,

$$\frac{\partial J}{\partial V} = \frac{\partial J}{\partial e} \frac{\partial e}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial V} = e_o \left[V_{j,\Re} + i V_{j,\Im} \right] \tag{20}$$

The Cauchy-Riemann equation are satisfy as $r = \hat{y}_\Re - y$ and $s = \hat{y}_\Im$:

$$\left| \begin{array}{ll} \frac{\partial r}{\partial \hat{y}_\Re} = 1 & \frac{\partial r}{\partial \hat{y}_\Im} = 0 \\ \frac{\partial s}{\partial \hat{y}_\Re} = 0 & \frac{\partial s}{\partial \hat{y}_\Im} = 1 \end{array} \right| > 0, \quad \frac{\partial e_0}{\partial \hat{y}} = 1 \tag{21}$$

For the fully connected layer, two gradients are required. Since the Cauchy-Riemann equations hold for $r = \hat{y}_\Re$ and $s = \hat{y}_\Im$,

$$\left| \begin{array}{ll} \frac{\partial r}{\partial V_\Re} = \Phi_\Re^{(m)} & \frac{\partial r}{\partial V_\Im} = -\Phi_\Im^{(m)} \\ \frac{\partial s}{\partial V} = \Phi_\Im^{(m)} & \frac{\partial s}{\partial V_\Im} = \Phi_\Re^{(m)} \end{array} \right| > 0 \tag{22}$$

the partial derivative of $\hat{y}$ respect to V is

$$\frac{\partial \hat{y}}{\partial V} = \frac{\partial r}{\partial V_\Re} + i \frac{\partial s}{\partial V_\Re} = \Phi_\Re^{(m)} + i\Phi_\Im^{(m)} = \Phi^{(m)} \tag{23}$$

We can calculate the gradient of cost function respect V, and for the rest of the hyper-parameters in the output layer using chain rule

$$\frac{\partial J}{\partial V} = \frac{\partial J}{\partial e} \frac{\partial e}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial V} = (e_o)(1)\Phi^{(m)}) = e\Phi^{(m)} \tag{24}$$

The gradient of $\hat{y}$ respect to $\Phi^{(m)}$ is obtain with aid of the Cauchy-Riemann equation, $r = \hat{y}_\Re$ and $s = \hat{y}_\Im$:

$$\left| \begin{array}{ll} \frac{\partial r}{\partial \Phi_\Re^{(m)}} = V_\Re & \frac{\partial r}{\partial \Phi_\Im^{(m)}} = -V_\Im \\ \frac{\partial s}{\partial \Phi_\Re^{(m)}} = V_\Im & \frac{\partial s}{\partial \Phi_\Im^{(m)}} = V_\Re \end{array} \right| > 0 \tag{25}$$

So

$$\frac{\partial \hat{y}}{\partial \Phi^{(m)}} = \frac{\partial r}{\partial \Phi_{\Re}^{(m)}} + i\frac{\partial s}{\partial \Phi_{\Re}^{(m)}} = V_{\Re} + iV_{\Im} = V \tag{26}$$

The updating law for the output layer (13) is

$$V(k+1) = V(k) - \eta\phi^{(\ell)}\left[V_{j,\Re} + iV_{j,\Im}\right]e_o(k) \tag{27}$$

The updating law for the parameters in the convolution operation is

$$W_{ij}(k+1) = W_{ij}(k) - \eta\frac{\partial J}{\partial W_{ij}}$$

where J is defined in (12), W_{ij} is parameter of the convolution operation.

3.2 Backpropagation for Complex Valued Convolution Operation

The error is back-propagated from the layer $\ell - 1$ to the layer ℓ as

$$e_\ell(k) = e_{\ell-1}(k)\frac{\partial \phi^{(\ell-1)}}{\partial t}W^{(\ell-1)} \tag{28}$$

In backward order, the next layer is the activation layer with $\mathbb{C}ReLU$. It does not have parameters,

$$\frac{\partial J}{\partial \varphi^{(\ell)}} = \frac{\partial J}{\partial \phi^{(\ell)}}\varphi^{(\ell)} \tag{29}$$

After activation layer, the convolutional layer with filters is updated.

For classical CNN,

$$\frac{\partial J}{\partial \gamma_J^{(\ell)}} = \sum_{a=0}^{N-f_\ell}\varphi_a^{(\ell)}\phi_{J+a}^{(\ell-1)} \tag{30}$$

where N is the size of the $\phi^{(\ell-1)}$, $\gamma_J^{(\ell)}$ is one of the elements in the filter $\Gamma^{(\ell)}$. The training law is

$$\begin{aligned}\gamma_J^{(\ell)}(k+1) &= \gamma_J^{(\ell)}(k) - \eta\frac{\partial J}{\partial \gamma_J^{(\ell)}}\\ \frac{\partial J}{\partial \phi_J^{(\ell-1)}} &= \sum_{a=0}^{f_\ell-1}\frac{\partial J}{\partial \varphi_{J-a}^{(\ell)}}\frac{\partial \varphi_{J-a}^{(\ell)}}{\partial \phi_J^{(\ell-1)}} = \sum_{a=0}^{f_\ell-1}\frac{\partial j}{\partial \varphi_{J-a}^{(\ell)}}\gamma_a^{(\ell)}\end{aligned} \tag{31}$$

The training object of [29] is to improve image classification, while our object is nonlinear dynamic system modeling with respect to measurement noise and missing data. Unlike the CNN and [29], we apply the convolution operation only in sum of the products of cells. We also use two real-value equations for each layer training, we do not use complex-value to train each layer as in [29]. These two differences between the proposed CVCNN and [29] help us to simplify the training and the analysis, at the same time they do not affect modeling accuracy.

3.2.1 Feedforward Operation

The filters and the weights of CVCNN are defined as

$$\Gamma^{(\ell)} = \begin{bmatrix} \gamma_{1,\Re}^{(\ell)} + i\gamma_{1,\Im}^{(\ell)} \\ \vdots \\ \gamma_{p,\Re}^{(\ell)} + i\gamma_{p,\Im}^{(\ell)} \end{bmatrix} \in \mathbb{C} \tag{32}$$

and

$$W^{(\ell)} = \left[W_{1,\Re}^{(\ell)} + i\,W_{1,\Im}^{(\ell)} \cdots W_{m,\Re}^{(\ell)} + i\,W_{m,\Im}^{(\ell)} \right]^T \in \mathbb{C}^m \tag{33}$$

The input of CVCNN is $\phi^{(0)} = \begin{bmatrix} \phi_1^{(0)} \\ \phi_2^{(0)} \\ \cdots \\ \phi_l^{(0)} \end{bmatrix}$. In the first layer, the convolution operation leads into the feature map $\varphi^{(1)}$,

$$\begin{aligned} \varphi_1^{(1)} &= \phi^{(0)} * \Gamma_1^{(1)} = \begin{bmatrix} \Psi_{1,1}^{(1)} \\ \vdots \\ \Psi_{1,q}^{(1)} \end{bmatrix} \\ &= \begin{bmatrix} \phi_1^{(0)} \\ \vdots \\ \phi_q^{(0)} \end{bmatrix} * \begin{bmatrix} \gamma_{1,\Re}^{(1)} + i\gamma_{1,\Im}^{(1)} \\ \vdots \\ \gamma_{p,\Re}^{(1)} + i\gamma_{p,\Im}^{(1)} \end{bmatrix} \end{aligned} \tag{34}$$

where each element $\Psi^{(1)} \in \mathbb{C}$ is obtained by the convolution

$$\Psi_{1,j}^{(1)} = \sum_{a=0}^{q} \phi_a^{(0)} \gamma_{j-a} \tag{35}$$

Then the $\mathbb{C}ReLU$ is applied to the feature maps,

$$\phi_1^{(1)} = \mathbb{C}\mathrm{ReLU}(\varphi_1^{(1)}) \tag{36}$$

The elements of this vector are

$$\begin{aligned} \phi_1^{(1)} &= \begin{bmatrix} \Phi_{1,1}^{(1)} \\ \vdots \\ \Phi_{1,q}^{(1)} \end{bmatrix} = \begin{bmatrix} \mathbb{C}\mathrm{ReLU}(\Psi_{1,1}^{(1)}) \\ \vdots \\ \mathbb{C}\mathrm{ReLU}(\Psi_{1,q}^{(1)}) \end{bmatrix} \\ &= \begin{bmatrix} \mathrm{ReLU}(\Psi_{1,1,\Re}^{(1)}) + i\mathrm{ReLU}(\Psi_{1,1,\Im}^{(1)}) \\ \vdots \\ \mathrm{ReLU}(\Psi_{1,3,\Re}^{(1)}) + i\mathrm{ReLU}(\Psi_{1,p,\Im}^{(1)}) \end{bmatrix} \end{aligned} \tag{37}$$

The last layer of the convolution operation is

$$\hat{y} = W^{(m)} \phi^{(m-1)} = \left[W_1^{(m)} \cdots W_p^{(m)} \right] \begin{bmatrix} \Phi_1^{(m-1)} \\ \cdots \\ \Phi_p^{(m-1)} \end{bmatrix} \tag{38}$$

The output of CVCNN is

$$\hat{y} = \hat{y}_{\Re} + i\,\hat{y}_{\Im}$$

where $\hat{y}_{\Re}$ and $\hat{y}_{\Im}$ are defined as:

$$\begin{aligned}
\hat{y}_{\Re} &= W_{1,\Re}^{(m)} \Phi_{1,\Re}^{(m-1)} - W_{1,\Im}^{(m)} \Phi_{1,\Im}^{(m-1)} + \cdots \\
&\quad + W_{p,\Re}^{(m)} \Phi_{p,\Re}^{(m-1)} - W_{p,\Im}^{(m)} \Phi_{p,\Im}^{(m-1)} \\
&\quad \vdots \\
\hat{y}_{\Im} &= W_{1,\Re}^{(m)} \Phi_{1,\Im}^{(m-1)} + W_{1,\Im}^{(m)} \Phi_{1,\Re}^{(m-1)} + \cdots \\
&\quad + W_{p,\Re}^{(m)} \Phi_{p,\Im}^{(m-1)} + W_{p,\Im}^{(m)} \Phi_{p,\Re}^{(m-1)}
\end{aligned} \tag{39}$$

here $\Phi_{1,\Re}^{(m-1)}$ and $\Phi_{1,\Im}^{(m-12)}$ are the real and imaginary part of the element $\Phi_1^{(m)}$, respectively.

3.2.2 Backpropagation of CVCNN

The backpropagation of the convolution operation needs chain rule and partial derivatives to calculate the gradient of the complex element. In the activation layer, the gradient is

$$\begin{gathered}
r = \mathrm{ReLU}(\Psi_{\Re})^{(m)}, \quad s = \mathrm{ReLU}(\Psi_{\Im}^{(m)}) \\
\begin{vmatrix}
\frac{\partial r}{\partial \Psi_{\Re}^{(m)}} = \Psi_{\Re}^{(m)} & \frac{\partial r}{\partial \Psi_{\Im}^{(m)}} = 0 \\
\frac{\partial s}{\partial \Psi_{\Re}^{(m)}} = 0 & \frac{\partial s}{\partial \Psi_{\Im}^{(m)}} = \Psi_{\Im}^{(m)}
\end{vmatrix} > 0
\end{gathered} \tag{40}$$

This leads to

$$\frac{\partial \Phi^{(m)}}{\partial \Psi^{(m)}} = \Psi_{\Re}^{(m)} \tag{41}$$

The gradient through a activation layer is determinate by adding $\Psi_{\iota j}^{(m)}$ into the corresponding vector $\varphi_j^{(m)}$,

$$\frac{\partial J}{\partial \varphi_\iota} = \frac{\partial J}{\partial \phi_\iota^{(m)}} \varphi_\iota^{(m)} \tag{42}$$

The convolutional layer is similar to the fully connected layer. The convolution operation can be regarded as the sum of products. Similar with (31),

$$\frac{\partial J}{\partial \gamma_J^{(\ell)}} = \sum_{a=0}^{N-f_\ell} \varphi_a^{(\ell)} \phi_{J+a}^{(\ell-1)} \tag{43}$$

and the gradient through a convolutional layer is

$$\frac{\partial J}{\partial \phi_J^{(\ell-1)}} = \sum_{a=0}^{f_\ell - 1} \frac{\partial J}{\partial \varphi_{J-a}^{(\ell)}} \frac{\partial \varphi_{J-a}^{(\ell)}}{\partial \phi_J^{(\ell-1)}} = \sum_{a=0}^{f_\ell - 1} \frac{\partial j}{\partial \varphi_{J-a}^{(\ell)}} \gamma_a^{(\ell)} \tag{44}$$

4 Simulations

In this section, we use three benchmarks to show the effectiveness of the complex valued CNN (CVCNN) compared with the classical CNN, classical neural network (MLP), and some other recent methods. The architecture of the CNN is the same as the CVCNN, but the filters and the weights are different, they are real value and complex values.

CNN has two convolutional layer followed by a ReLU activation function and a max-pooling layer. For each benchmark, the number of filters in the convolutional layers are different. We use random walk to find the best possible combination. MLP has one hidden layer, the activation function is $tanh(\cdot)$. The hidden nodes are different according to each benchmark. The initial filters of CNN and CVCNN are chosen randomly in the range $[-1, 1]$. The weights of MLP are also chosen in $[-1, 1]$.

In order to show the advantages of CVCNN for dynamic system modeling, we use the following two neural network models:

(1) Series-parallel model as (5)

$$\hat{y}(k) = NN[y(k-1), \cdots u(k), \cdots] \tag{45}$$

where both the input $u(k)$ and the output $y(k-1)$ of identified system are fed to the neural network $NN[\cdot]$.

(2) Parallel model as (4)

$$\hat{y}(k) = NN[\hat{y}(k-1), \cdots + u(k), \cdots] \tag{46}$$

where only the input $u(k)$ of identified system is fed to the neural network $NN[\cdot]$. We use the output of the neural networks $\hat{y}(k-1)$ as the other part of neural network input. In the case of noise, (45) becomes

$$\hat{y}(k) = NN[y(k-1) + \rho, \cdots + u(k), \cdots] \tag{47}$$

where ρ is the random noise. (46) becomes

$$\hat{y}(k) = NN[\hat{y}(k-1) + \rho, \cdots u(k), \cdots]$$

In the case of missing data, (45) becomes

$$\hat{y}(k) = NN[\bar{y}(k-1), \cdots \bar{u}(k), \cdots] \tag{48}$$

where $\bar{y}(k-1)$ and $\bar{u}(k)$ are from the data sets $\{y(1), \cdots y(N)\}$ and $\{u(1), \cdots u(N)\}$. (46) becomes

$$\hat{y}(k) = NN[\tilde{y}(k-1), \cdots \tilde{u}(k), \cdots]$$

where $\tilde{y}(k-1)$ and $\tilde{u}(k)$ are from the data sets $\left\{\hat{y}(1), \cdots \hat{y}(N)\right\}$ and $\{u(1), \cdots u(N)\}$.

In this paper, we select 30% of the data as missing data.

4.1 Searching Hyper-parameters

There are not optimal methods to define the neural structure. Most of them use trial and error to find a good structure. We will use random search method to decide the hyper-parameters of the neural model. It is similar with [44].

We first randomly select the combinations of the hyper-parameters in total set. Then we search the best score, i.e. minimizing a hyper-parameter response function Υ. The algorithm is as follows for CNN:

1. Choose the number of convolutional layers in CNN from (1,2 or 3 layers).
2. Define the number of filters in each layer (1–100 filters) and its dimension(3-6 of length).
3. Initialize the filters and synaptic weights of the output layer (range: -1 to 1).
4. Carry out the simulation and obtain the score of the function Υ
5. Repeat previous steps 15 times with randomly hyper-parameter settings.
6. Choose the structure with the best score.

In case of MLP, the algorithm is as follows:

1. define a two-layer MLP.
2. Define the number of neurons or nodes in the hidden layer (1–100 nodes).
3. Initialize the synaptic weights (range: -1 to 1).
4. Carry out the simulation and obtain the score of the function Υ
5. Repeat previous steps 10 times with randomly hyper-parameter settings.
6. Choose the structure with the best score.

In both cases, data set is used without preprocessing. With these algorithms, the hyper-parameters are decided and shown in the following sections, which are the best score in the hyper-parameter response function Υ according to the random search algorithm.

4.2 Gas Furnace Modeling

The data set of gas furnace is a benchmark example for nonlinear system modeling [45]. The input signal $u(k)$ is the flow rate of the methane gas. The output signal $y(k)$ is the concentration of CO_2. There are 296 samples in 9 seconds. We use 200 samples for training, the other 96 samples for testing. We compare our CVCNN with CNN and MLP.

For the series-parallel models (45), (47) and (48), the input vector to the neural models are

$$[y(k-1), \cdots y(k-5), u(k), \cdots u(k-4)]$$

For the parallel model (46), the input vector to the neural models are $[\hat{y}(k-1), u(k), \cdots u(k-10)]$. Each convolutional layer of CVCNN and CNN has 3 filters, the size of each filter is 3. The MLP has 50 nodes in the hidden layer. The amplitude of the noise is about 10% of the output amplitude. The modeling errors of testing phase for gas furnace with the parallel model (46) are shown in Fig. 3.

The performance of the MLP becomes worse if we reduce the percentage of training data from 60% to 40%, i.e, data number is from 200 to 120. The root mean square error (RMSE) goes from 0.1257 to 0.2435. The same occurs for CNN, increasing from 0.1466 to 0.1935 and for the CVCNN, the RMSE also increase, from 0.0829 to 0.1458. Reducing the amount of training data does increase the RMSE value. Both CNNs have less problems in deducing training data compared with MLP. The quantity of 200 data is chosen, because this quantity is commonly used for this benchmark.

4.3 First Order Nonlinear System

The following discrete-time first-order nonlinear system is another popular benchmark [38],

$$y(k+1) = \frac{y(k)}{1+y^2(k)} + u^3(k) \tag{49}$$

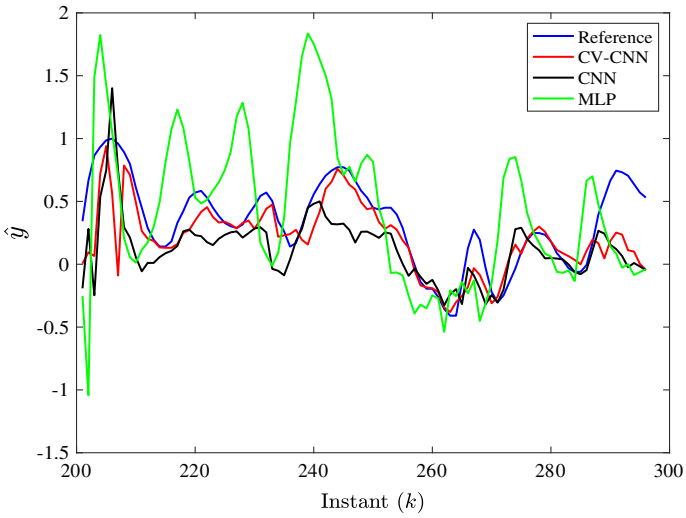

Fig. 3 Modeling errors of parallel model for gas furnace

The control input $u(k)$ is periodic, $u(k) = B\sin\left(\frac{\pi k}{50}\right) + C\sin\left(\frac{\pi k}{20}\right)$. In training phase, $B = C = 1$. In testing phase, $A = 0.9$, $B = 1.1$. We use 5000 data generated by (49) to train the neural models, and use 100 data for the testing.

For the series-parallel models (45), (47) and (48), the input vector to the neural models is

$$[y(k-1), \cdots y(k-10), u(k), \cdots u(k-13)]$$

For the parallel model (46), the input vector to the neural models are $\left[\hat{y}(k-1), u(k), \cdots u(k-12)\right]$. The CVCNN and CNN have 8 filters in each convolutional layer. MLP has 35 nodes in the hidden layer. The amplitude of the noise is around 10% of the output amplitude.

The system identification of the testing results for the first-order nonlinear system with the parallel model (46) are shown in Fig. 4.

4.4 Wiener-Hammerstein System

Wiener-Hammerstein system is the series connection of a linear system, a static non linearity and another linear system. It is a electrical circuit consisting of three cascade blocks [46]. This benchmark system has 14, 000 samples. We use 1, 000 for testing, the other part data are used for training. For the series-parallel models (45), (47) and (48), the input vector to the neural models are $[y(k-1), \cdots y(k-4), u(k), \cdots u(k-5)]$, with noise amplitude around 10% of output amplitude For the parallel model (46), the input vector to the neural models are $\left[\hat{y}(k-1), \cdots \hat{y}(k-3), u(k), \cdots u(k-80)\right]$. For CVCNN and CNN, each convolutional layer has 15 filters, the seize is 6. The MLP is the same as the model of the gas furnace, using 80 nodes in the hidden layer. The modeling errors of the series-parallel model with noise data (47) are shown in Fig. 5. The modeling errors of the parallel model (46) are shown in Fig. 6. We can see that the parallel model with the CVCNN is alternative in the context of system modeling, due to its better performances compared to other methods.

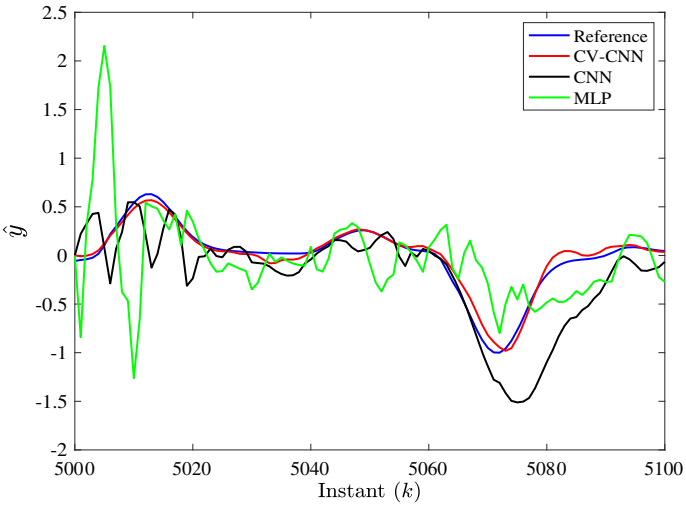

Fig. 4 The modeling errors of parallel model for first-order system

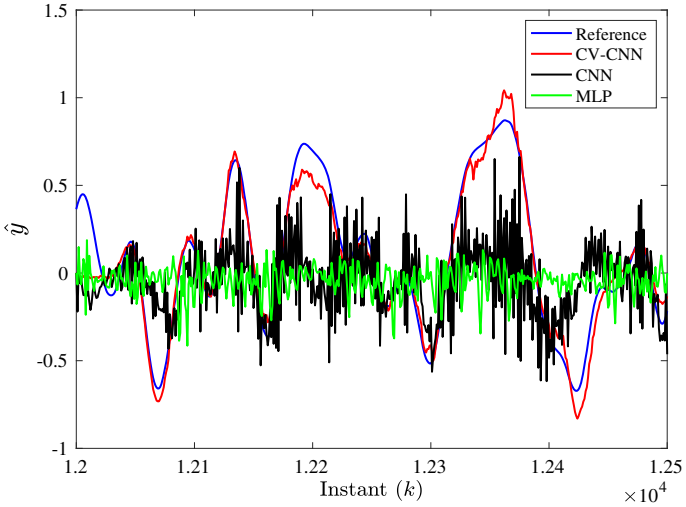

Fig. 5 Modeling errors of series-parallel model with noisy data for the Wiener-Hammerstein system

4.5 Discussion

The main metric to evaluate performance is the mean square error (MSE) defined by

$$\frac{1}{N}\Sigma_{k=1}^{N}\left[y(k)-\hat{y}(k)\right]^2$$

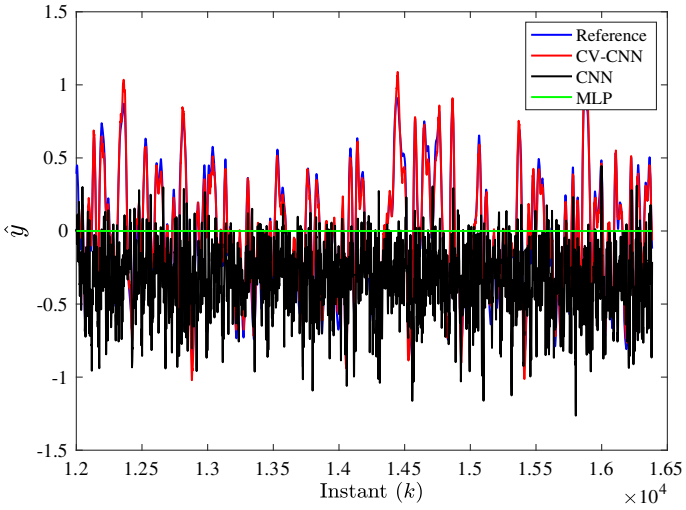

Fig. 6 Modeling errors of parallel model for the Wiener-Hammerstein system

Table 1 Performance metrics for gas furnace benchmark using series parallel model

	R^2	WI	MSE	MAE	RMSE
CVCNN Filters: 3	0.8784	0.9669	0.0069	0.0729	0.0829
MLP Nodes: 50	0.7207	0.9166	0.0158	0.1279	0.1257
CNN Filters: 3	0.6200	0.8833	0.0215	0.1313	0.1466

We also use the other metrics as

$$\mathrm{R}^2 : 1 - \frac{\frac{1}{N}\Sigma_{k=1}^{N}\left[y(k) - \hat{y}(k)\right]^2}{\frac{1}{N}\Sigma_{k=1}^{N}\left[y(k) - \bar{y}(k)\right]^2}$$

$$\text{Mean absolute error(MAE): } \frac{1}{N}\Sigma_{k=1}^{N}\left|y(k) - \hat{y}(k)\right|$$

$$\text{Root mean squared error (RMSE): } \sqrt{\frac{1}{N}\Sigma_{k=1}^{N}\left[y(k) - \hat{y}(k)\right]^2}$$

For the above three benchmarks, we use five metrics to compare our CVCNN with CNN and MLP. The MLP has one hidden layer. CNN and CVCNN have the same structure as above.

Tables 1, 2, 3, 4 show the comparison results of the three models. We can see that in the most cases, the proposed CVCNN performance better than the others

Table 5 shows the modeling errors of different recent methods. In order to do a fair comparison, we use the same models as the Arima model and PEC-WNN in [47]. We form the problems of prediction, modeling and identification into the same sense: we use the trained model to estimate the next value in the time series, and the objects are the same. We can see that our CVCNN has better performances than the other methods in missing data

Table 2 Performance metrics for gas furnace benchmark using parallel model

	R^2	WI	MSE	MAE	RMSE
CVCNN Filters: 3	0.4042	0.8270	0.0337	0.1914	0.1835
MLP Nodes: 50	−1.1887	0.0983	0.1237	0.3222	0.3518
CNN Filters: 3	0.5052	0.8657	0.0280	0.1742	0.1672

Table 3 Performance metrics for gas furnace benchmark using series parallel model with noise data

	R^2	WI	MSE	MAE	RMSE
CVCNN Filters: 3	0.8785	0.9669	0.0074	0.0980	0.0860
MLP Nodes: 50	−3.4390	0.4211	13.9291	1.7413	3.7322
CNN Filters: 3	0.8159	0.7255	0.9420	0.1015	0.1293

Table 4 Performance metrics for gas furnace benchmark using series parallel model with missing data

	R^2	WI	MSE	MAE	RMSE
CVCNN Filters: 3	0.9305	0.9841	0.0042	0.0736	0.0650
MLP Nodes: 50	−0.6885	−0.3736	0.0953	0.3530	0.3087
CNN Filters: 3	0.6803	0.9454	0.0195	0.1041	0.1395

case. PEC-WNN is better than ours, but it has to use complete data set. None of the other methods considers missing data.

For this benchmark we can conclude that:

- In the cases of big noise and missing data, CVCNN gives good modeling accuracy than CNN and MLP.
- If the parallel model is used, both CVCNN and CNN work well.
- If the series-parallel model can be used, both CNN and MLP work well.

Tables 6, 7, 8, 9 shown the comparison results of Wiener-Hammerstein system. We can see that similar with the gas furnace modeling, our method gives better results for Wiener-Hammerstein system. It should be noted that our method has better performances in the cases of missing data and big disturbances. CVCNN, CNN and MLP give good modeling accuracy with series-parallel. For the cases of noise and missing data, the CVCNN and CNN work well, but MLP cannot model correctly.

Table 10 presents the results of other four methods for dynamic system modeling: LSTM and SVM are popular sadistical learning methods, BLA uses Spearman correlation for model optimization, PNLSS uses polynomial nonlinear state-space model. All of these methods use the trained model to estimate the next value in the time series, and the modeling objects

Table 5 Performances of the other recent methods for the gas furnace modeling

Model	RMSE
Arima	0.843
Tong's model	0.685
Xu's model	0.573
Sugeno's model	0.596
Surmans's model	0.4
ANFIS	0.405
Generalized fuzzy NN	0.273
PEC-WNN	0.0589
OSELM-K [48]	0.3341
LSTM [49]	0.9730
CVCNN	0.0829

Table 6 Performance metrics for Wiener-Hammerstein benchmark using series parallel model

	R^2	WI	MSE	MAE	RMSE
CVCNN Filters: 15	0.9991	0.9998	0.000076	0.0.0018	0.0087
MLP Nodes: 50	0.9551	0.9884	0.0038	0.0641	0.0617
CNN Filters: 15	0.9975	0.9994	0.00021	0.0127	0.0146

Table 7 Performance metrics for Wiener-Hammerstein benchmark using parallel model

	R^2	WI	MSE	MAE	RMSE
CVCNN Filters: 15	0.9503	0.9867	0.0042	0.0684	0.0649
MLP Nodes: 80	0.3588	0.7217	0.0544	0.2458	0.2322
CNN Filters: 15	0.8747	0.9734	0.0106	0.0992	0.1031

Table 8 Performance metrics for Wiener-Hammerstein benchmark using series parallel model with noisy data

	R^2	WI	MSE	MAE	RMSE
CVCNN Filters: 15	0.9734	0.993	0.00018	0.0155	0.0138
MLP Nodes: 80	−0.0350	−0.0237	0.0878	0.3445	0.2963
CNN Filters: 15	0.1471	0.5112	0.0723	0.3082	0.2690

Table 9 Performance metrics for Wiener-Hammerstein benchmark using series parallel model with missing data

	R^2	WI	MSE	MAE	RMSE
CVCNN Filters: 15	0.8753	0.9702	0.00089	0.0100	0.0298
MLP Nodes: 80	0.7890	0.9344	0.0015	0.0430	0.0388
CNN Filters: 15	0.7129	0.9341	0.0020	0.0106	0.0453

Table 10 Performance of Wiener-Hammerstein system modeling using the other methods

Model	RMSE
LSTM [49]	0.011
BLA [50]	0.00607
SVM [51]	0.047
PNLSS [52]	0.00042
CVNN	0.0087

Table 11 Performance metrics for nonlinear system benchmark using series parallel model

	R^2	WI	MSE	MAE	RMSE
CVCNN Filters: 8	0.9993	0.9998	3.97×10^{-5}	0.0061	0.0063
MLP Nodes: 35	0.9501	0.9884	0.0030	0.0585	0.0551
CNN Filters: 8	0.9936	0.9984	3.90×10^{-4}	0.0222	0.0198

Table 12 Performance metrics for nonlinear system benchmark using parallel model

	R^2	WI	MSE	MAE	RMSE
CVCNN Filters: 8	0.9179	0.9829	0.0050	0.0799	0.0707
MLP Nodes: 35	−1.3991	0.7553	0.1461	0.3007	0.3822
CNN Filters: 8	0.5012	0.9167	0.0304	0.2033	0.1743

are the same. We can see that our CVCNN has very similar results with the other for this benchmark. But for the cases of big noise and missing data, they did not show results.

Tables 11, 12, 13, 14 give the modeling results of the nonlinear system. Clearly, CVCNN performs better than the other methods. We can see that for the first-order nonlinear system, CVCNN, CNN and MLP give good modeling accuracy with series-parallel. For the cases of noises and missing data, CVCNN and CNN work well, but MLP cannot.

Table 13 Performance metrics for nonlinear system benchmark using series parallel model with noise data

	R^2	WI	MSE	MAE	RMSE
CVCNN Filters: 8	0.9122	0.9758	0.3111	0.6106	0.5577
MLP Nodes: 35	0.8710	0.9645	0.4572	0.7495	0.6761
CNN Filters: 8	0.8915	0.9710	0.3846	0.6767	0.6201

Table 14 Performance metrics for nonlinear system benchmark using series parallel model with missing data

	R^2	WI	MSE	MAE	RMSE
CVCNN Filters: 8	0.9760	0.9940	0.0850	0.2423	0.2915
MLP Nodes: 35	−2.1640	0.5257	11.2123	1.0461	3.3485
CNN Filters: 8	0.9560	0.9895	0.1561	0.1512	0.3951

5 Conclusion

Measurement noise and missing data are important disturbances in system identification, which can affect directly modeling accuracy. CVCNN has much power and better modeling accuracy than the other classical modeling methods for these cases, although the mathematical analysis of CVCNN is more difficult. So CVCNN is efficient and robust model compare to classic ones for nonlinear dynamic system modeling with bid uncertainties. Our further work will be on CVCNN based robust control.

References

1. Noel JP, Kerschen G (2017) Nonlinear system identification in structural dynamics: 10 more years of progress. Mech Syst Signal Process 83:2–35
2. Lopez M, Morales J, Yu W (2020) Frequency domain CNN and disipate energy approach for damage detection in building structures. Soft Comput 24:15821–1584051
3. LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, Jackel LD (1989) Backpropagation applied to handwritten zip code recognition. Neural Comput 1(4):541–551
4. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958
5. Bengio Y, Lamblin P, Popovici D, Larochelle H (2007) Greedy layer-wise training of deep networks. In: Proceedings of the advances in neural information processing systems (NIPS'06), pp. 153–160
6. Hinton G, Osindero S, Teh Y (2006) A fast learning algorithm for deep belief nets. Neural Comput 18(7):1527–1554
7. Sermanet P, LeCun Y (2011) Traffic sign recognition with multi-scale convolutional networks. In: Proceedings of the 2011 international joint conference on neural networks (IJCNN)
8. Mao L, Li X, Yang D et al (2021) Convolutional feature frequency adaptive fusion object detection network. Neural Process Lett. https://doi.org/10.1007/s11063-021-10560-4
9. Soleymanpour S, Sadr H, Beheshti H (2020) An efficient deep learning method for encrypted traffic classification on the web. In: 2020 6th International conference on web research (ICWR), IEEE, pp. 209–216

10. Sultana F, Sufian A, Dutta P (2018). Advancements in image classification using convolutional neural network. In: 2018 Fourth international conference on research in computational intelligence and communication networks (ICRCICN), IEEE, pp. 122–129
11. Yadav SS, Jadhav SM (2019) Deep convolutional neural network based medical image classification for disease diagnosis. J Big Data 6(1):1–18
12. Altan G, Yayik A, Kutlu Y (2021) Deep learning with ConvNet predicts imagery tasks through EEG. Neural Process Lett 53:1–16
13. Ismail FH et al (2019) Deep learning for time series classification: a review. Data Min Knowl Discov 33(4):917–963
14. Hatami N, Gavet Y, Debayle J (2017) Classification of time-series images using deep convolutional neural networks. arXiv:1710.00886v2
15. Wang Y et al. (2016) CNNpack: Packing convolutional neural networks in the frequency domain. In: NIPS
16. Lee H, Groose R, Ranganath R, Ng A (2009) Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In: Proceedings of the 26th international conference on machine learning (ICML09), pp. 609–616
17. Kiranyaz S, Avci O, Abdeljaber O, Ince T, Gabbouj M, Inman DJ (2021) 1D convolutional neural networks and applications: a survey. Mech Syst Signal Process 151:107398
18. Borovykh A, Bohte S, Oosterlee CW (2017) Conditional time series forecasting with convolutional neural networks. arXiv preprint arXiv:1703.04691
19. Javedani SH et al (2019) Short-term load forecasting by using a combined method of convolutional neural networks and fuzzy time series. Energy 175:365–377
20. Binkowski M, Marti G, Donnat P (2017) Autoregressive convolutional neural networks for asynchronous time series. arXiv:1703.04122
21. Namasudra S, Dhamodharavadhani S, Rathipriya R (2021) Non-linear neural network based forecasting model for predicting COVID-19 cases. Neural Process Lett. https://doi.org/10.1007/s11063-021-10495-w
22. Wang Y (2017) A new concept using LSTM neural networks for dynamic system identification. In: 2017 American control conference (ACC), IEEE, pp 5324–5329
23. Genc S (2017) Parametric system identification using deep convolutional neural networks. In: Proceedings of the 2017 international joint conference on neural networks (IJCNN17), pp. 2112–2119
24. Lopez M, Yu W (2019) Impact of random weights on nonlinear system identification using convolutional neural networks. Inf Sci 477(1):1–14
25. Kang Y, Chen S, Wang X, Cao Y (2018) Deep convolutional identifier for dynamic modeling and adaptive control of unmanned helicopter. IEEE Trans Neural Netw Learn Syst 30:524–538
26. Virtue P, Yu SX, Lustig M (2017) Better than real: complex-valued neural nets for MRI fingerprinting. IEEE Int Conf Image Process (ICIP) 2017:3953–3957. https://doi.org/10.1109/ICIP.2017.8297024
27. Xiong T et al (2015) Forecasting interval time series using a fully complex-valued RBF neural network with DPSO and PSO algorithms. Inf Sci 305:77–92
28. Dramsch J, Lüthje M, Christensen AN (2021) Complex-valued neural networks for machine learning on non-stationary physical data. Comput Geosci 146:104632 (**Elsevier**)
29. Guberman N (2016) On complex valued convolutional neural networks. arXiv preprint arXiv:1602.09046
30. Benvenuto N, Piazza F (1992) On the complex backpropagation algorithm. IEEE Trans Signal Process 40(4):967–969
31. Hirose A, Yoshida S (2011) Comparison of complex-and real-valued feedforward neural networks in their generalization ability. In: International conference on neural information processing. Springer, Berlin, Heidelberg
32. Hirose A (2003) Complex-valued neural networks: theories and applications, vol 5. World Scientific, Singapore
33. Quan Y, Chen Y, Shao Y, Teng H, Xu Y, Ji H (2021) Image denoising using complex-valued deep CNN. Pattern Recognit 111:107639
34. Zhang Z, Wang H, Xu F, Jin YQ (2017) Complex-valued convolutional neural network and its application in polarimetric SAR image classification. IEEE Trans Geosci Remote Sens 55(12):7177–7188
35. Meyer M, Kuschk G, Tomforde, S (2020) Complex-valued convolutional neural networks for automotive scene classification based on range-beam-doppler tensors. In: 2020 IEEE 23rd International conference on intelligent transportation systems (ITSC), IEEE, pp. 1–6
36. Yao X, Shi X, Zhou F (2020) Human activities classification based on complex-value convolutional neural network. IEEE Sens J 20(13):7169–7180
37. Zimmermann HG et al (2011) Comparison of the complex valued and real valued neural networks trained with gradient descent and random search algorithms. In: Proc. of ESANN 2011

38. Narendra K, Parthasarathy K (1990) Identification and control of dynamical systems using neural networks. IEEE Trans Neural Netw 1(1):4–27
39. Ruta D, Gabrys B (2007) Neural network ensembles for time series prediction. Int Joint Conf Neural Netw 2007:1204–1209
40. Gu H, Qing G, Wang Y, Hong S, Gacanin H, Adachi F (2020) Deep complex-valued convolutional neural network for drone recognition based on RF fingerprinting
41. Tan X, Li M, Zhang P, Wu Y, Song W (2019) Complex-valued 3-D convolutional neural network for PolSAR image classification. IEEE Geosci Remote Sens Lett 17(6):1022–1026
42. Hongyo R, Egashira Y, Yamaguchi K (2018) Deep neural network based predistorter with ReLU activation for Doherty power amplifiers. Asia-Pacific Microw Conf (APMC) 2018:959–961
43. James G, Burley D (2002) Matematicas avanzadas para ingenieria. Pearson Educacion, London
44. Bergstra J, Bengio Y (2011) Random search for hyper-parameter optimization. J Mach Learn Res 13:281–305
45. Box G, Jenkins G, Reinsel G (2008) Time series analysis: forecasting and control, 4th edn. Wiley, Hoboken
46. Schoukens J, Suykens J, Ljung L (2009) Wiener-Hammerstein benchmark. In: Proceedings of the 5th IFAC symposium on system identification
47. Ustundag BB, Kulaglic A (2020) High-performance time series prediction with predictive error compensated wavelet neural networks. IEEE Access 8:210532–210541
48. George K, Mutalik P (2017) A multiple model approach to time-series prediction using an online sequential learning algorithm. IEEE Trans Syst, Man, Cybern: Syst 49(5):976–990
49. Gonzalez J, Yu W (2018) Non-linear system modeling using LSTM neural networks. IFAC-PapersOnLine 51(13):485–489
50. Shaikh MAH, BarbÃ K (2019) Wiener Hammerstein system identification: a fast approach through spearman correlation. IEEE Trans Instrum Measurement 68(5):1628–1636
51. Marconato A, Schoukens J (2009) Identification of Wiener-Hammerstein benchmark data by means of support vector machines. IFAC Proc Vol 42(10):816–819
52. Paduart J, Lauwers L, Pintelon R, Schoukens J (2012) Identification of a WienerHammerstein system using the polynomial nonlinear state space approach. Control Eng Pract 20(11):1133–1139

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.