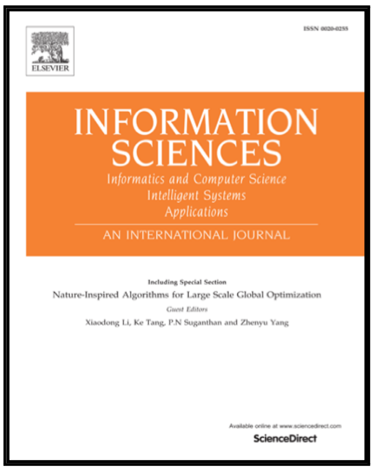

Impact of random weights on nonlinear system identification using convolutional neural networks

Wen Yu, Mario Pacheco

PII: S0020-0255(18)30816-8
DOI: https://doi.org/10.1016/j.ins.2018.10.019
Reference: INS 14000

To appear in: *Information Sciences*

Received date: 7 June 2018
Revised date: 8 October 2018
Accepted date: 14 October 2018

Impact of random weights on nonlinear system identification using convolutional neural networks

Wen Yu, Mario Pacheco

Departamento de Control Automatico

CINVESTAV-IPN (National Polytechnic Institute)

Av.IPN 2508, Mexico City, 07360, Mexico

yuw@ctrl.cinvestav.mx

Abstract

Randomized algorithms have been successfully applied in modelling dynamic system. How do random weights affect system identification and why do they sometimes work well? In this paper, we use the convolutional neural network (CNN) as an identification model to answer these questions.

Since the convolution operation is an important property of the dynamic system and in the frequency domain it becomes the product, the CNN model is analyzed in the frequency domain. We first modify the CNN model, so that it can model both the input and the output series. Then we analyze the impact of the random weights of CNN in the frequency domain. We prove the existence of optimal weights and analyze the modeling accuracy under optimal weights and random weights. Through theoretical analysis, we propose a two-step training method and compare it with the random weight algorithm. The proposed CNN model with random weights is validated with three benchmark problems.

Keywords: convolutional neural network, random algorithms, frequency domain, deep learning.

1 Introduction

To predict the future behavior of the dynamic system, to apply model-based control or to understand the physical process, system identification is needed. Neural networks are black box models, which only use input and output data. These data-based modeling methods still have many problems. The hyper-parameters must be defined before they are applied, for example, how many hidden layers and hidden nodes are needed [21]. The universal approximation theorem guarantees that the neural model can approach almost all continuous systems with enough hidden nodes, however, the fact of increasing the hidden nodes causes the over-fit problem [34].

The alternative method to increase the hidden nodes is to use more hidden layers. It is a basic idea of deep neural networks. Theory and application show that deep learning models achieve better training precision [2], and get impressive results on many difficult tasks [14]. The convolutional neural network (CNN) is one of the most important deep learning models [22][33]. The main difference between CNN and normal neural networks is the convolution operation. The first successful CNN is at early 90s for the classification of digits [23], which includes the invariances in two-dimensional forms using local connections and weight restrictions. The training uses the maximum likelihood estimation and the modified backpropagation algorithm [26]. To give a better understanding of how convolutional networks work, [39] proposed the de-convolutional technique to visualize the operations of the hidden layer. By using GPU (Graphic Processing Unit), faster learning and testing are obtained by [20]. It has been proven that CNNs have great performances in classification tasks, such as image processing [30]. CNNs are also applied for data regression and time series modeling. In [4], the time series is formed in the autoregressive model. In [12], CNN is applied to classify the time series.

There are two correlated time series in dynamic systems, the input $x(k)$ and the output $y(k)$. The identification of the system is to find the relationship between these two time series. This relationship is sometimes convolution, for example, the time-invariant linear system. The input and the output series can be joined in a time series, the normal time series prediction method is applied in [25]. However, the convolution relation between input and the output disappears.

To the best of our knowledge, there are few published results on the application of CNN to the dynamic system modeling. In our previous paper [25], we gave a possible scheme for modeling nonlinear systems with CNN. In [19], the CNNs are trained to model uncertainties in the dynamic system. It is the extended version of classic neural control. In [9], the identification of the nonlinear system is transferred to the time series model using the ARMAX model. However, all the above papers do not analyze why CNN works well for the system identification.

Random algorithms were initially proposed in [31] and were studied in depth in [17]. Their hidden weights are chosen at random and the pseudoinverse approach (or the least squares method) is applied to calculate the output weights. They show that the optimization of the hidden layer parameters does not significantly improve the generalization behavior, while the update of the output weights is more effective [29]. The importance of the scope for the random assignment of hidden parameters is described in [24]. The neural model can be generated by the stochastic configuration method, this stochastic configuration networks [37] needs less human intervention and fast learning. Random algorithms have been applied successfully to the identification of nonlinear systems in [36]. In [8], the random hidden weights are adjusted by the statistical characteristics of the system input through restricted Boltzmann machines. The combination of random algorithms and CNN has good results in classification tasks. [7] shows that CNN has a significant degree of robustness in the classification of images with random filters. For image processing, [28] explains why the square pooling in CNN works well with random weights.

In this paper, we will study if the CNN with random weights also generates surprising results for the system identification. We take advantage of CNN and the randomized algorithm for the modeling of nonlinear systems. The behaviors of the proposed model are analyzed in the frequency domain. We show that there are optimal inputs (or optimal filters), and the precision of modeling with random filters is closed to the optimal filter. These theoretical results help us to design a two-step training method for the CNN model: pre-training for CNN filters and fine training for the full connection layer. Three benchmark examples are applied to show that the randomized algorithm with the CNN model is effective for the identification of nonlinear systems.

Nomenclature

Meaning	**symbol**
system input	$u(k)$
system output	$y(k)$
internal state	$\bar{x}(k)$
input to NARMA model	$x(k)$
unknown NARMA function	$\Gamma[\cdot]$
unknown nonlinear functions	f, g
modelling error	e_m, e_o

Meaning	**symbol**
weights of the last layer	V
CNN operations	$\Phi[\cdot]$
filter parameters	W_{ij}
CNN output	$\hat{y}(k)$
discrete Fourier transform	$\mathcal{F}$
convolution operation	$*$
normalization of x	X

This paper is organized as follows: Section 2 explains how to use CNN for the identification of nonlinear systems and shows why CNN is favorable for system modelling. Section 3 gives our theoretical analysis of the impact of random weights in the frequency domain. Section 4 provides the training algorithms of the CNN model. The random filters are pre-trained. Then we discuss the significance of random weights. Section 5 reports the results of the simulation with three benchmark problems and shows the impact of random weights for the identification of nonlinear systems. Section 5 concludes the paper.

2 Convolutional neural networks for system modeling

CNN can be an extremely efficient model for the modelling of nonlinear systems, because the convolution operation in CNN is the same as the input-output relation of the linear time invariant system. For two integers k and i, if the input is the pulse function $u(k) = \delta(k-i)$[1], then the output is the pulse response $y(k) = h(k-i)$. The relation between $y(k)$ and $u(k)$ is

$$y(k) = \sum_{i=-\infty}^{k} u(i)\, h(k-i) = u(k) * h(k)$$

where $*$ is defined as the convolution operation.

Consider the following unknown discrete-time nonlinear system

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

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

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

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

$$y(k) = \Gamma[x(k)] \tag{2}$$

where

$$x(k) = [y(k-1), \cdots y(k-n_y), u(k), \cdots u(k-n_u)]^T \tag{3}$$

$\Gamma(\cdot)$ is the unknown nonlinear difference equation representing the plant dynamics, $u(k)$ and $y(k)$ are measurable input and output, $x(k) = [x_1 \cdots x_l]^T$, $l = n_y + n_u + 1$, $k = 1 \cdots N$, N is the data number.

The unknown nonlinear system (2) can be modeled by the following two types of models:

1. Simulation model,

$$\hat{y}(k) = N[u(k), \cdots, u(k-n)] \tag{4}$$

where $\hat{y}(k)$ is the output of the model, $N[\cdot]$ is the model structure, for example it is the neural network. n is the regression order for the input $u(k)$. Since n_u in (2) is unknown, $n \neq n_u$. (4) is also called nonlinear finite impulse response (FIR) model.

[1] $\delta(\cdot)$ is the Dirac delta function, $\delta(0) = \infty$, $\delta(x) = 0$ when $x \neq 0$, and $\int_{-\infty}^{\infty} \delta(x)\, dx = 1$.

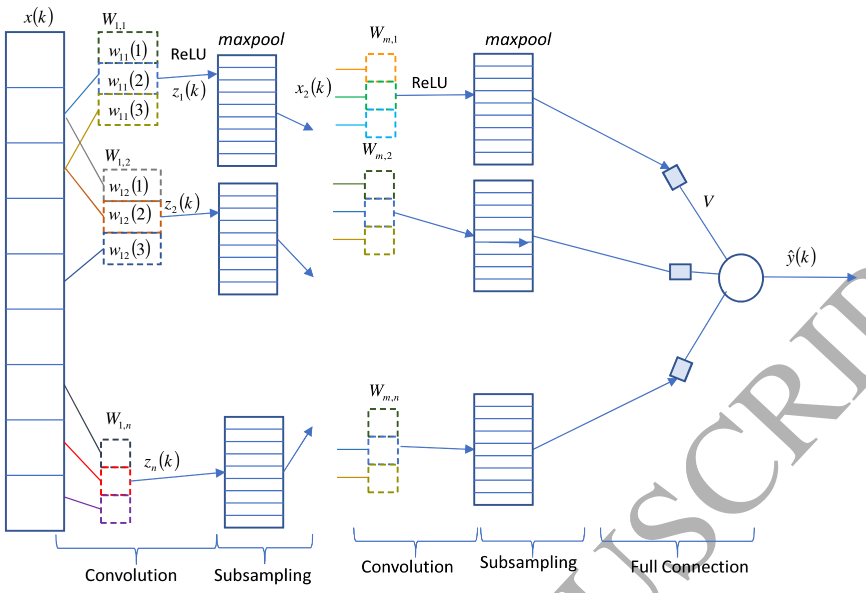

Figure 1: CNN model for system identification

In general form, (4) can be written as

$$\hat{y}(k) = N[\hat{y}(k-1), \cdots, \hat{y}(k-m), u\left(k\right), \cdots, u\left(k-n\right)] \tag{5}$$

where m is the regression order for the model output $\hat{y}\left(k\right)$, $m \neq n_y$. The difference between the simulation model (5) and the NARX model (2) is that the simulation model can not use the actural output $y\left(k-m\right)$ in its regression model, it can only use its own output $\hat{y}(k-m)$. In this paper, we use this model. It is the parallel identification model of [27].

2. Prediction model,

$$\hat{y}(k) = N[y\left(k-1\right), \cdots, y\left(k-m\right), u\left(k\right), \cdots, u\left(k-n\right)] \tag{6}$$

where m is the regression order, $y\left(k\right)$ is the output of the plant (2), $m \neq n_y$. The data regression form of (6) is the same as the NARX model (2). It is the series-parallel identification model of [27].

In many modeling applications, we have to use the simulation model (4), which has a static input-output mapping. For example, multi-step prediction and neural control based on offline models can not use recent actural outputs $y\left(k-i\right), i = 1, 2 \cdots$, to update models.

The nonlinear system identification scheme based on CNN is shown in Figure 1. CNN has a cascade connection structure. Each CNN cell has two layers: convolution layer and sub-sampling layer. Each convolution layer has n convolution operators, which are called filters, $C_{1,1}\cdots C_{1,n}$. There are m levels. The last layer is fully connected. The output of the CNN model is

$$\widehat{y}\left(k\right)=V\Phi\left[x\left(k\right)\right] \tag{7}$$

where $\widehat{y}\left(k\right)$ is the output of the CNN model, Φ represents the operation of $m-$level, $V\in\Re^{n}$ is the weight vector of the final layer. In each convolutional layer, there are n filters, the convoluted output is

$$z=x*W_{ij},\quad z=\left[zz_{1}\cdots zz_{N-p}\right],\quad zz_{q}=\sum_{k=1}^{p}x\left[p(q-1)+k\right]w_{ij}\left[k\right] \tag{8}$$

where p is the filter length, N is the length of $x\left(k\right)$, $z\left(k\right)=\left[zz_{1}\cdots zz_{N-p}\right]$ is the feature map generated in this layer. In this paper, we let all the filters have the same length. w_{ij} are the parameters of the filters, they are the hidden weights of CNN.

After the convolution (8), the signal $z\left(k\right)$ goes through an activation function. In this paper, we use the rectified linear unit (ReLU) as

$$Z\left(k\right)=\max\left[0,z\left(k\right)\right],\quad z=\left[z_{1}\cdots z_{n}\right] \tag{9}$$

We use the max-pooling method [13] in the sub-sampling layer

$$x_{2}\left(k\right)=\max_{p}\left[Z\left(k\right),s\right] \tag{10}$$

where s is the shrink parameter, which depends on the layer, $x_{2}\left(k\right)$ is a new input for the second convolutional layer, see Figure 1.

The following special properties of CNN are favorable to the modeling of nonlinear systems:

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. In Figure 1, the width of the receptive field is p, the neuron in layer m only receives p nodes from layer $m-1$.

2) Each convolutional kernel scans the complete data, and the input features are mapped to the output, no matter where the receptive fields are located. This means that the same properties of the system have the same weights. The convolution calculation in CNN reduces the risk of over-fitting by sharing weights. In Figure 1, the nodes of the layer m belong to the same feature map. This means that CNN learns more meaningful features of the dynamic system.

3) The multiple-level pooling, such as the maximum operation in the sub-sampling layer, generates tolerance to the noise in the data. This makes the CNN model very robust.

CNNs have the following special technical issues related to the modeling of nonlinear systems:

1) In general, the backpropagation learning in deep networks has the gradient vanish problem. The ReLU (rectified linear units) in CNN does not face this gradient problem.

2) The pre-training can extract properties of the dynamic system. It can learn something intrinsic about the data. They extract the patterns from the input-output data and store the information learned in the weights of the networks.

3) The convolution operation used by CNN facilitates the analysis of the modeling of the system in the frequency domain.

The object of the system modeling is to update the weights W_{ij} and V of CNN, or let alone W_{ij} randomly and only update V, so that the output of CNN (7) convergence to the system output $y(k)$ in (2)

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

3 Frequency domain analysis of random weights

The motivations of the frequency domain analysis for CNN are: 1) The convolution operation in CNN becomes element-wise product, the analysis becomes easy and direct; 2) The changes of the convolution with random filters are less in the frequency domain. With these, we can explain why random weights perform well for nonlinear system identification.

If we use discrete Fourier transform (DFT), the convolution operation in (8) can be transformed into multiplication,

$$\mathcal{F}(x * W) = \mathcal{F}(x) \times \mathcal{F}(W) = \mathcal{F}(x)\,\mathcal{F}(W) \tag{11}$$

where F represents the DFT for the corresponding vector, $*$ represents the convolution operation, $F(x)$ and $F(W)$ are vectors in the frequency domain, $\times$ represents element-wise product.

Since the product requires the two vectors, $F(x)$ and $F(W)$, have the same length, the vector with smaller length is filled with zeros, that is the same elements as the larger one.

We consider the CNN model as in Figure 1, which includes (8) and (10)

$$x_2(s) = Sx^T W \tag{12}$$

where x_2 is the output of the maxpooling, W is the convolution matrix (filters), S is the

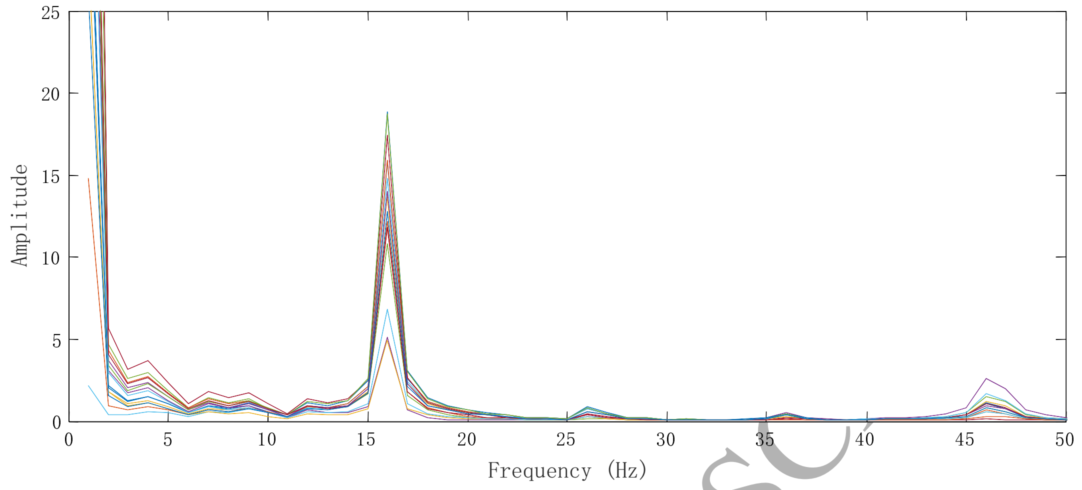

Figure 2: The frequency spectra of $x\,(k) * W_{i,j}$. The forms are not changed by the random filters $W_{i,j}$.

vector as $[0, \cdots 0, 1, 0, \cdots, 0]$ to obtain the maximum value of the convolution operation. Here we do not need the exact position of "1".

We first use an example to show that random hidden weights in CNN do not affect so much to the convolution operation. We use the gas furnace data set, which is the benchmark problem in [18]. The input $u\,(k)$ is the flow rate of the methane gas, while the output $y\,(k)$ is the concentration of CO_2 in the gas mixture under steady air supply. The data regression in (2) is, $x\,(k) = [y(k-1) \cdots y(k-4)\ u(k) \cdots u(k-5)]$. The data set has 296 samples with the fixed interval 9 seconds, they are added certain frequency noises. We use DFT to transform the data to the frequency domain, then apply convolution between $x(k)$ and filters. When the weights of the filters are random, the spectra of the convolution are shown in Figure 2.

Compare with the input frequency spectrum in Figure 3, some filters increase the frequency amplitude of $x\,(k)$, some filters decrease it. However, the form of $x\,(k)$ is not changed by random filters, that is, the features of $x\,(k)$ can be extracted by random weights when the convolution operation is applied. This property can not be seen in the time domain.

From the point of view of identification, we expect the amplitude to reach the highest, so that its features can be extracted with the greatest energy. In the time domain, we want to find the best filter $W_{i,j}$ such that

$$\max_{w} \left[x * W_{i,j} \right] \tag{13}$$

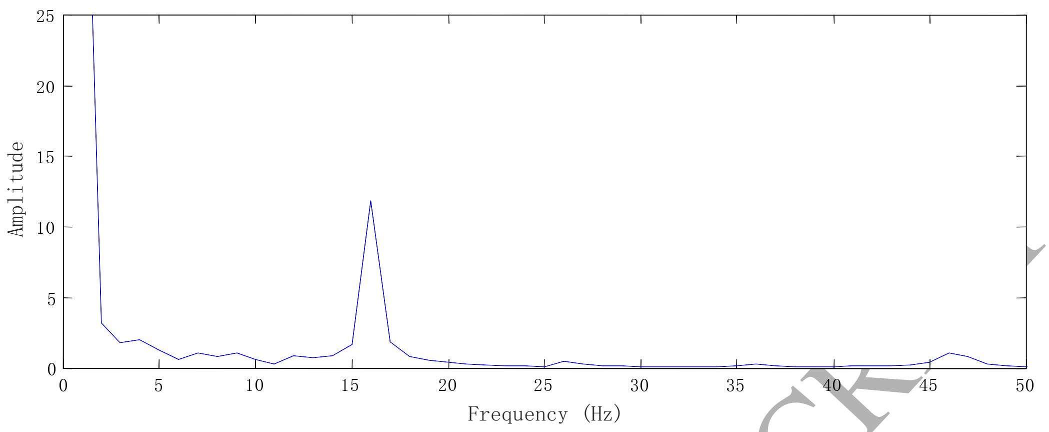

Figure 3: The frequency spectrum of the input $x(k)$. This form is similar with $x(k) * W_{i,j}$.

It is a difficult task. However, in the frequency domain, (13) becomes

$$\max_{w} \left[x(s)^T W(s) \right] \tag{14}$$

where $x(s) = F(x)$, $W(s) = F(W)$, F represents the discrete Fourier transform. Since the input data is known, it is easier to do

$$\max_{x} \left[x(s)^T W(s) \right] \tag{15}$$

(15) means we can find an optimal input instead of the optimal filter. The following theorem provides the optimal input, so that (13) is maximized.

Theorem 1 *The input data $x(k)$, $k = 1 \cdots N$, pass through the filter $W_{ij} = [w_{ij}(1), \cdots w_{ij}(p)]$, $p < N$. There is the optimal input, which maximizes the CNN cell, i.e. (13) or (15) with the ReLU (9) and the max-pooling (10) as*

$$x^*(k) = \frac{\sqrt{2}}{N} \cos\left(\frac{2\pi f_0}{N} k + \phi \right) \tag{16}$$

where f_0 is the optimal frequency, ϕ is the phase.

Proof. Each CNN cell includes the convolution operation (13), the linear operation ReLU (9), and the max-pooling (10). The optimization problem (14) is

$$S \max_{x} [Wx] \quad \text{or } S \max_{x} [WX] \tag{17}$$

where S is defined in (12), X is the normalization of x, $\|X\|^2 = 1$, W is the circulant matrix, i.e., each row is the circular shift of the previous row [11]. Then the optimization problem (17) can be describe as

$$S \max_{X, X \neq 0} \frac{(X^T W^T W X)^{\frac{1}{2}}}{X^T X} = S \max_{X, X \neq 0} J \tag{18}$$

Here $X \neq 0$ is under the assumption that the zero frequency can be the maximal frequency f_0. The optimal solution must be the eigenvector associated with the maximum value of $W^T W$. As we saw before, the discrete Fourier transform can be used to place the eigenvalues of $W^T W$ in the diagonal matrix, F is the matrix of the discrete Fourier transform. (18) is in a positive semi-definite quadratic form. The optimal solutions are the eigenvectors associated with the maximal value of $W^T W$. We use DFT to (18),

$$J = \frac{(X^T W^T W x)^{\frac{1}{2}}}{X^T X} = \frac{(X^T F^T F W^T W F^T F X)^{\frac{1}{2}}}{X^T F^T F X} \tag{19}$$

The discrete Fourier transform matrix satisfies $F^T F = I$. We define $q = Fx$, q is the discrete Fourier transform of x. Apply $F^T F$ inside $W^T W$,

$$J = \frac{(q^T F W^T F^T F W F^T q)^{\frac{1}{2}}}{q^T q} \tag{20}$$

Considering that W is the circulant matrix, then $FWF^T = \Lambda$. Finally the optimization problem (17) is

$$\begin{cases} S \max_q \frac{(q^T |\Lambda|^2 q)^{\frac{1}{2}}}{q^T q} \\ \text{Subject: } F^T q \in \Re \end{cases} \tag{21}$$

The constraint $F^T q \in \Re$ guarantees $x(k) \in \Re$. The matrix Λ is diagonal, $\Lambda_{ii} = \beta \left| FW \right|_i$, β is the scaling factor. Then, the eigenvalues are $\lambda_i = \beta |\Lambda|_{ii}$. Because $x(k)$ must be real, the DFT coefficients corresponding to the negative frequencies must be its complex conjugate, i.e. $q_{-i} = \bar{q}_i$ and $q_{-j} = \bar{q}_j$. To satisfy this, a possible way is

$$q_j = \begin{cases} v^{i\text{sign}(j) b_{|j|}} a_{|j|}, & \lambda_j \in \max(\lambda) \\ 0 & \text{otherwise} \end{cases} \tag{22}$$

with a and b are vectors of arbitrary coefficients, one of them is the maximal frequency f_0, $\|a\| = 1$, v is the eigenvector of (21). The reality condition is

$$a_{|j|} e^{i\text{sign}(-j)\phi_{|j|}} = \overline{a_{|j|} e^{i\text{sign}(j)\phi_{|j|}}} = \bar{q}_j \tag{23}$$

So for nonzero q_j and $\bar{q}_j$, the reality condition of x is fulfilled. Since (23) reaches the maximum without the restriction of reality, it is also the maximum for the complete problem. If we convert q back to the time domain, the maximum of x is (16). ■

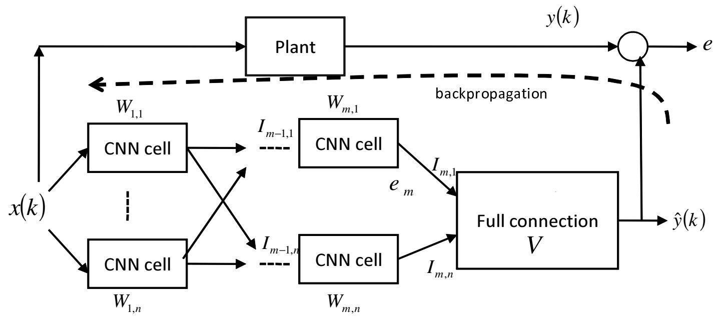

Figure 4: The hierarchical structure of CNN for system identification.

Remark 1 *This theorem shows that there is an optimal input for random filters. The index in the frequency domain (14) shows that for any inputs (random or determined), it corresponds to the optimal filter. The frequency of the optimum input is the frequency of maximum magnitude in the filter. Since the phase is not specified, the architecture of the filter is invariant in the translation. On the other hand, any random filter containing some frequencies of moderate magnitude can generate the maximum magnitude of the best input frequency, i.e., the optimal input places all its energy at the highest amplitude frequency in the random filter. This is why random weights sometimes do it so well.*

4 CNN training with random weights

In the previous session, we showed that the convolution operation in CNN with random filters (random weights) can very well represent the characteristics of the nonlinear system. From Figure 2 and Figure 3 we can see that although the optimum filter maximizes the amplitude of the frequency spectra, the effectiveness of this optimal filter is limit. In this section, we use the two-step training method, pre-training and fine tuning, to show the impact of the random weights.

In the pre-training, the weights of the convolutional filters W_{ij} are randomized first, then we use small data set and the backpropagation to update them.

In the fine tuning, the weights of the convolutional filters are set to the values of the pre-training, the weight V in the last layer is updated.

4.1 Pre-training

The identification scheme with CNN is shown in Figure 4. Each "CNN cell" has the convolution operation (13), the activation function ReLU (9), and the max-pooling (10). There are $m \times n$ CNN cells. The final block is fully connected.

The objective is to train the weights so that the error between the neural model (7) and the plant (2) is minimized. The performance index is defined as

$$J = \frac{1}{2}e_o^2 \quad e_o = \hat{y} - y \tag{24}$$

The last block is the classical single layer feedforward neural network, the training law is the following gradient descent

$$V(k+1) = V(k) - \eta I_m(k) e_o(k) \tag{25}$$

where $I_m(k) = [I_{m,1} \cdots I_{m,n}]^T$, which are the outputs of the last layer of the CNN model, $\eta > 0$ is the learning rate.

The error is back-propagated to the layer m as

$$e_m(k) = e_o(k) \frac{\partial \sigma}{\partial t} V = e_o(k) V(k) \tag{26}$$

where σ is the active function of the full connection layer. Here we use the linear function, $\frac{\partial \sigma}{\partial t} = I$.

For each CNN cell, we must update the filter weight W_{ij} in (8) as

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

where J is defined in (24).

For $m-$th CNN cell

$$W_{mj}^{(k+1)} = W_{mj}^{(k)} - \eta I_{m-1,j}(k) e_m(k)$$

When the errors are back propagated from block m to block $m-1$, e_m is the backpropagation error through the max-pooling (10), the activation function ReLU (9), and the convolution operation (13). Because ReLU is linear, the error goes directly through ReLU.

In the max-pooling, there is no more operation than the shrink of the data, the reverse operation is the following up-sampling

$$e_{mp}(k) = up[e_m(k)]$$

where the $up(\cdot)$ is the opposite function to the max-pooling. Only the elements that have the greatest contribution in the next stage are applied.

When the error e_{mp} goes through the convolution operation (13),

$$e_{mc} = e_{mp} * W_{ij}$$

we use the following discrete-time de-convolution

$$e_{mc} = e_{mp} * [rot180\,(W_{ij})]$$

where $rot180(\cdot)$ rotates the matrix W_{ij} to 180°, it is equivalent to performing the convolution of e_{mp} with the filter W_{ij},. Finally, from layer m to layer $m-1$, the error becomes

$$e_{m-1} = up\,[e_m\,(k)] * [rot180\,(W_{ij})] \tag{27}$$

For any CNN cell, the training law of the filters is

$$\begin{array}{l} W_{ij}^{(k+1)} = W_{ij}^{(k)} - \eta I_{i-1,j}\,(k)\,e_i\,(k) \\ e_i\,(k) = up\,[e_{i+1}\,(k)] * [rot180\,(W_{ij})] \end{array} \tag{28}$$

where $i = 1 \cdots m$, $j = 1 \cdots n$. Since the interior structures of the CNN cells are the same, we can use the same training law (28) to train the entire CNN model.

From the analysis of the frequency domain, we find that CNN with random weights is good for the identification of the system. There are optimal inputs (or optimal filters) in the frequency domain. The pre-training moves the random weights to obtain a good approximation capability.

4.2 Fine tuning

After the pre-training, the weights of the filters are fixed, because moving the filters in the fine tuning step does not improve significantly the modeling accuracy (see frequency domain analysis). In this fine tuning step, only the weights in the final layer are updated by the Moore-Penrose pseudoinverse.

Definition 1 *[16] The matrix $A^+ \in \Re^{n\times m}$ is the Moore-Penrose generalized inverse of $A \in \Re^{m\times n}$ if*

$$AA^+A = A, \quad A^+AA^+ = A^+, \quad \left(AA^+\right)^T = AA^+, \quad \left(A^+A\right)^T = A^+A \tag{29}$$

In particular, when A has full column rank,

$$A^+ = \left(A^TA\right)^{-1} A^T \tag{30}$$

When A has full row rank

$$A^+ = A^T \left(AA^T\right)^{-1} \tag{31}$$

Definition 2 *[16] $x_0 \in \Re^n$ is said to be a minimum norm least-squares solution of the linear system $y = Ax$ if*

$$\|x_0\| \leq \|x\|, \quad \forall x \in \{x : \|Ax - y\| \leq \|Az - y\|, \forall z \in \Re^n\} \tag{32}$$

where $y \in \Re^m$.

For a linear system $y = Ax$, x_0 is a least-squares solution if

$$\|Ax_0 - y\| = \min_x \|Ax - y\| \tag{33}$$

where $\|\cdot\|$ is a norm in Euclidean space. If By is a minimum norm least-squares solution of the linear system $y = Ax$, then it is necessary and sufficient that $B = A^+$. Here A^+ is the Moore-Penrose generalized inverse of matrix A, which is defined in (29). The object to train the weight V is

$$J = \min_V \sum_k \|y(k) - \hat{y}(k)\|^2 \tag{34}$$

where $\widehat{y}(k) = V\Phi[x(k)]$. The training data are $y(k)$ and $\Phi[k]$.

Consider all data $k = 1 \cdots N$,

$$\begin{array}{l} \hat{Y} = \left[\begin{array}{ccc} \hat{y}(1) & \cdots & \hat{y}(N) \end{array}\right] = \left[\begin{array}{ccc} V\Phi(1) & \cdots & V\Phi(N) \end{array}\right] = V\Psi \\ Y = \left[\begin{array}{ccc} y(1) & \cdots & y(N) \end{array}\right] = \left[\begin{array}{ccc} V\Phi(1) + e_o(1) & \cdots & V\Phi(N) + e_o(N) \end{array}\right] \end{array} \tag{35}$$

where $e_o(k)$ is the modeling error which defined in (24), $\Psi = [\Phi(1), \cdots, \Phi(N)]$. (35) in matrix form is

$$Y = V\Psi + E \tag{36}$$

where $E = [e_o(1), \cdots, e_o(N)]$. (36) is a linear-in-parameter system.

From Definition 2, when

$$\hat{V} = Y\Psi^T \left(\Psi\Psi^T\right)^{-1} = Y\Psi^+ \tag{37}$$

where Ψ^+ is the pseudoinverse of Ψ which is defined in Definition 1, (34) arrives the minimum value when $\frac{\partial J}{\partial \hat{V}} = 0$.

$\hat{V}$ is the least square solution of (36), it reaches the smallest approximation error on the training data set. The two-step training algorithm is as follows:

1. Construct the CNN model (7), the weights of the filters are randomly assigned in $[-\lambda, \lambda]$, the positive value λ represents the range of the random weights.

2. Use input and output data and (28) to pre-train the filters of the CNN.

3. Train the full connection layer $\hat{V}$ in (37) with Ψ in (35).

5 Simulations

In this section, we use three benchmark examples to show the effectiveness of CNN with random filters for the modeling of nonlinear systems. We use the following simulation model,

$$\begin{aligned} \hat{y}(k) &= CNN[x\left(k\right)] \\ x\left(k\right) &= \left[u\left(k\right), \cdots u\left(k-l\right)\right]^{T} \end{aligned} \tag{38}$$

where $\hat{y}(k)$ is the output of the CNN model, l is the regression order for the input $u\left(k\right)$. Normal neural networks work well with the prediction model (6) for the following three examples. However, they can not predict the output with the simulation model (38).

We will use the following three cases.

Case 1: random weights in $[0,1]$, without pre-training, with fine tuning;

Case 2: random weights in $[-1,1]$, without pre-training, with fine tuning;

Case 3: initial weights are in $[0,1]$, with pre-training, with fine tuning.

Case 1 and Case 2 have different ranges of random weights. We will also show that fine turning with random weights has good modeling accuracy, and avoid the time-consuming problem in the pre-training. This is the advantage of CNN in the identification of nonlinear systems.

5.1 First order nonlinear system

This benchmark example is proposed in [27]. It is a simple nonlinear system,

$$y(k+1) = \frac{y(k)}{1+y^{2}(k)} + u^{3}(k) \tag{39}$$

where $u(k)$ is a periodic input, which has a different form in the training and the testing processes, $u(k) = A\sin\left(\frac{\pi k}{50}\right) + B\sin\left(\frac{\pi k}{20}\right)$. In the training stage, $A = B = 1$. In the testing stage, $A = 0.9$, $B = 1.1$. This model has been used by many papers as the benchmark problem. Most of them used the prediction model (6). If the previous output $y(k-1), \cdots$ is not available, it is difficult to identify the nonlinear system. Here we use $x(k) = [u(k), \cdots, u(k-5)]^{T}$. The comparison with MLP is shown in Figure 5. We can see that the normal neural model, MLP, with two hidden layers and the hidden node numbers are 20 and 10, cannot model the system when the previous outputs are not available in the testing phase. While convolutional neural networks, CNN, with random weights can model it. Testing errors can be reduced by the pre-training as in Figure 6. Here the squared error is defined as $E = \frac{1}{N}\Sigma_{k=1}^{N}\left[y(k) - \hat{y}(k)\right]^{2}$. We see that the pre-training process in CNN can improve the modeling accuracy for this example.

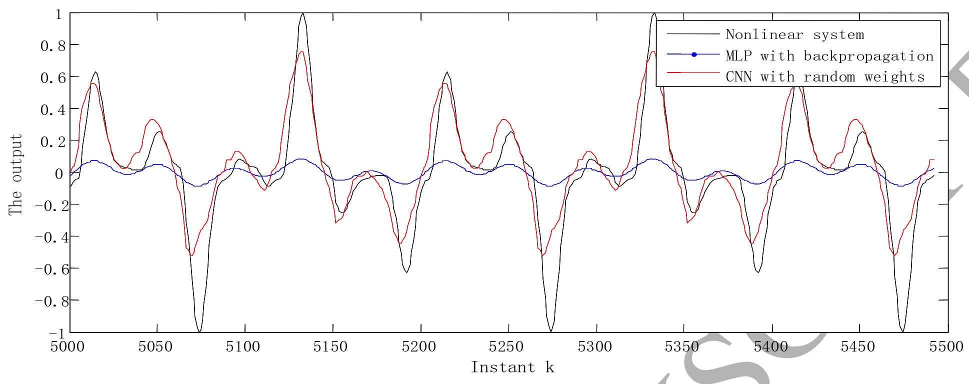

Figure 5: Testing results of nonlinear system modelling. CNNs with random weights are better than MLP.

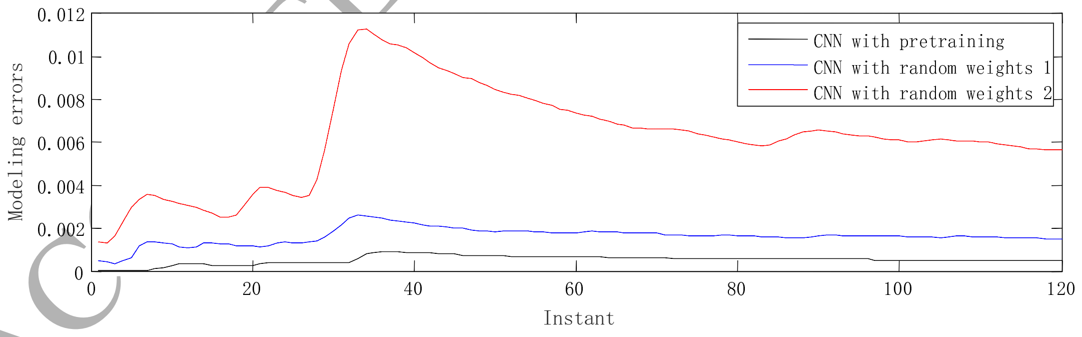

Figure 6: Squared testing errors of nonlinear system modelling. CNN with pre-training is better than CNN with random weight.

We change the weight distributions from $[0, 1]$ to $[-5, 5]$. Table 1 shows how the modelling accuracy is affected by random weights. We can see that different random distributions for CNN filters have good modeling performance. The range $[1, 3]$ of the random weights has better results.

Table 1. The testing squared errors of nonlinear system $(\times 10^{-3})$

Filter distributions	Random	Pre-training
$[-5, -3]$	4.11	5.98
$[-3, 0]$	3.26	3.57
$[0, 1]$	1.42	4.64
$[1, 3]$	1.37	1.02
$[-5, 5]$	3.51	5.95

5.2 Gas furnace data

The gas furnace data set is another benchmark problem [18]. The input $u\,(k)$ is the flow rate of the methane gas, while the output $y\,(k)$ is the concentration of CO_2 in the gas mixture under a steady air supply. The data set has 296 samples at the fixed interval of 9 seconds. [18] used a time-series based approach to develop a linear model. [35] and [38] use this dataset to evaluate their fuzzy modeling methods. In this paper, the recursive input data for the model is $x(k) = [u(k-1), \cdots u(k-10)]^T$, the output of the model is $\hat{y}(k)$. 200 samples are applied for training, the other 96 samples are used for testing. In the "pre-training+fine tuning" case, 100 samples are for pre-training, 100 samples are for fine tuning. The testing errors of different CNNs are shown in Figure 7. For this example, we can see CNN with the random weights is very good. The filters with the pre-training can not significantly improve modeling accuracy.

Similar to the previous example, the weight distributions of the filter are changed from $[0, 1]$ to $[-5, 5]$. Table 2 shows the modelling errors with different random weights. For this particular model, the range $[0, 1]$ of the random weights has better results.

Table 2. The testing squared errors of gas furnace $(\times 10^{-3})$

Filter distributions	Random	Pre-training
$[-5, -3]$	27.6	24.1
$[-3, 0]$	12.7	21.1
$[0, 1]$	9.5	8.9
$[1, 3]$	11.7	9.1
$[-5, 5]$	24.3	31.2

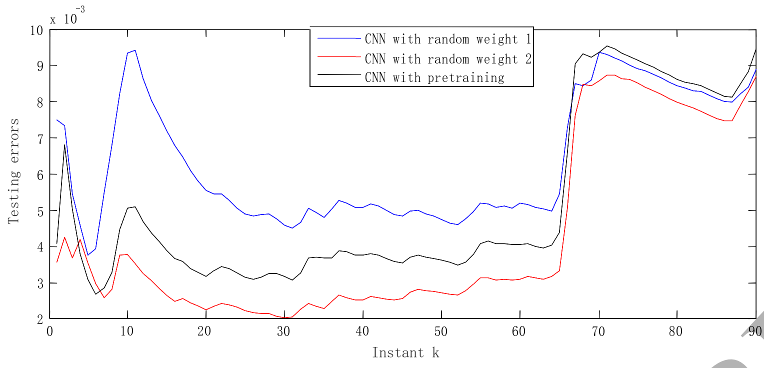

Figure 7: Testing errors of different CNNs (Gas furnace)

5.3 Wiener-Hammerstein system

The Wiener-Hammerstein system is the series connection of three parts: a linear system, a static nonlinearity and another linear system. The data of the Wiener-Hammerstein benchmark are generated from the electrical circuit consisting of three cascade blocks [32]. There is no direct measurement of static nonlinearity, since it lies between two unknown linear dynamic systems. The benchmark data set consists of $188,000$ input/output pairs. The data set is divided in two parts [1]: $100,000$ samples are for the training, $88,000$ samples are for the testing. We define the recursive input vector for the model as $x(k) = [u(k), \cdots, u(k-15)]$. The comparison with MLP is shown in Figure 8, the influence of the pre-training for CNN with random weights is shown in Figure 9. For the Wiener-Hammerstein benchmark data, the normal MLP can not model it, when $x(k) = [u(k), \cdots, u(k-15)]$. The CNN with random weights can model it very well.

We also evaluated how random weights affect modeling errors, see Table 3. We can see that the pre-training can improve modeling accuracy a bit. The hidden weight distribution $[0,1]$ is adequate for the Wiener-Hammerstein v problem.

Table 3. The testing squared errors of W-H $(\times 10^{-3})$

Filter distributions	Random	Pre-training
$[-5,-3]$	25.9	20.3
$[-3,0]$	21.7	22.4
$[0,1]$	6.6	6.2
$[1,3]$	8.3	7.1
$[-5,5]$	13.1	14.2

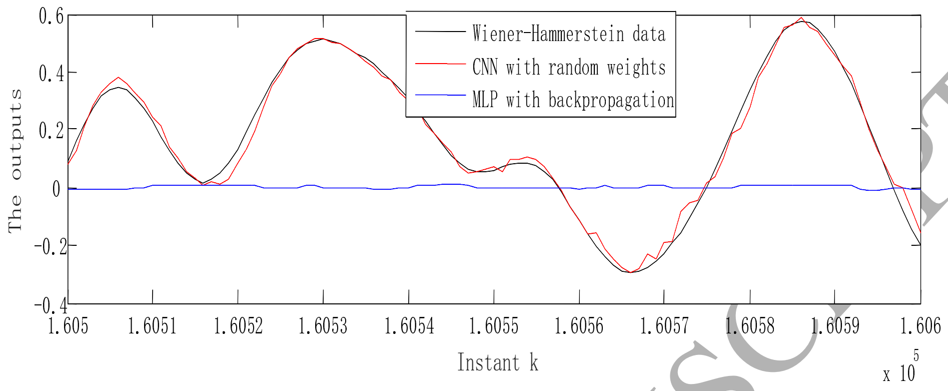

Figure 8: Testing results of Wiener-Hammerstein modelling. CNNs with random weights are much better than MLP.

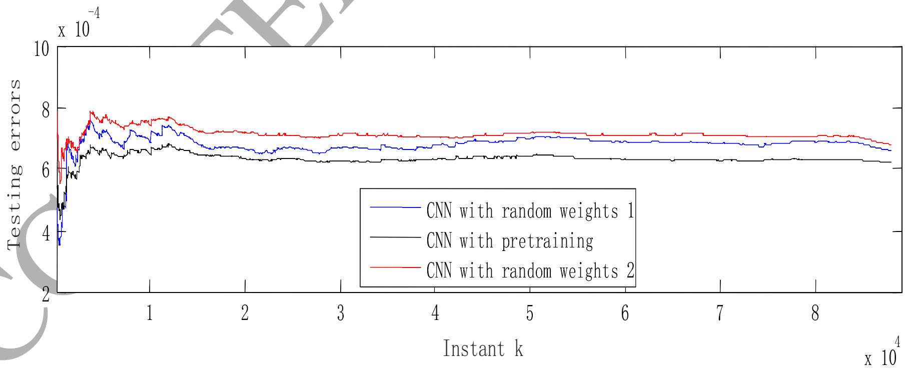

Figure 9: Squared testing errors of Wiener Hammerstein modelling. CNNs with random weights have similar results with the pre-training.

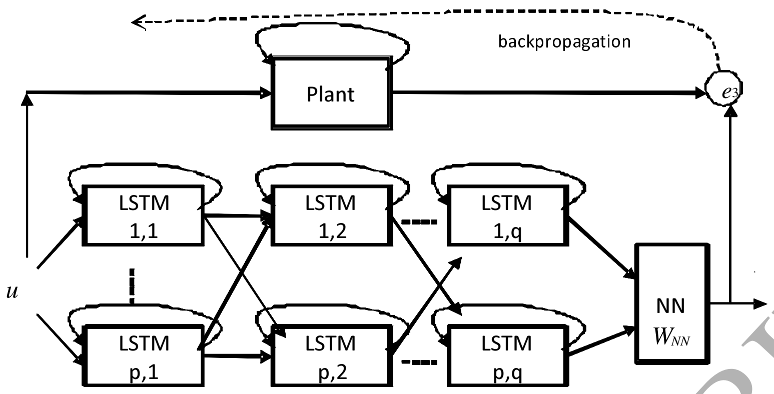

Figure 10: LSTM for nonlinear system modeling

5.4 Comparisons

We compare our algorithm with other popular black-box methods, such as the support vector machine (SVM) [5], multilayer perceptrons with gradient learning algorithm (MLP) [27], and the long-short term memory network (LSTM) [15]. The MLP is the same as [27]. Table 4 provides comparisons of these three examples with MLP and SVM. The SVM uses the radial basis function kernel. For the simulation model (4), CNNs with random weights have great advantages over MLP and SVM.

Table 4. Modelling errors of CNN and MLP ($\times 10^{-3}$)

	Nonlinear system	Gas furnace	W-H
Case 1	1.4	9.5	6.6
Case 2	3.6	8.7	6.8
Case 3	4.6	8.9	6.2
MLP	158	153	19.6
SVM	21	32	27

Now we discuss LSTM for our benchmark problems. We have successfully applied this model to nonlinear system modeling in [10], see Figure 10. The modelling results with different p and q are shown in Table 5. We can see that LSTM has results similar to those of CNN with random weights. However, LSTM has more training time, because it needs backpropagation through time (BPTT). The modelling results of the Wiener-Hammerstein system with LSTM are shown in Figure 11.

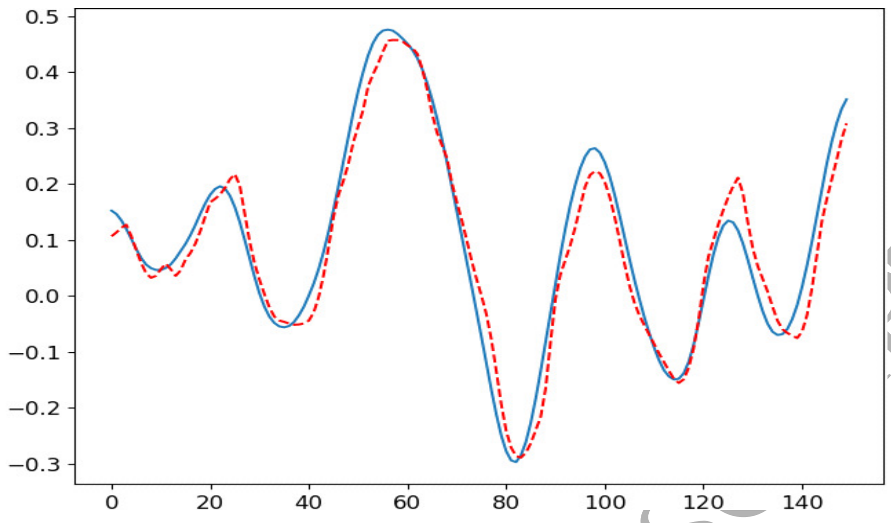

Figure 11: Testing result of Wiener-Hammerstein modelling. LSTM has similar results as NNs with random weights.

Table 5. Different structure of LSTM ($\times 10^{-3}$)

LSTM	Nonlinear system	Gas furnace	W-H
$p=3, q=1$	3.17	8.17	7.91
$p=50, q=1$	3.15	7.15	8.48
$p=4, q=4$	2.53	9.53	8.03
$p=4, q=20$	2.53	8.53	7.31
$p=8, q=40$	6.95	8.95	7.02

Then we compare different CNN structures, i.e., how do CNN filters affect the precision of modeling with random weights? Table 5 shows the modeling errors with different hidden nodes m (the number of filters). For these three examples, 20 filters in each convolution layer are sufficient.

Table 6. Modeling errors with different filters ($\times 10^{-3}$)

Number of filters	**3**	**10**	**15**	**20**	**30**
Non linear System	13	14	9.2	7.3	7.8
Gas Furnace	10.2	8.7	8.4	7.2	7.5
Wiener Hammerstein	9.5	6.2	4.7	3.1	3.6

Table 7 shows the modeling errors with different hidden layers n. For these three examples, $5-10$ hidden layers are sufficient.

Table 7. Modeling errors with different hidden layers ($\times 10^{-3}$)

Number of layers	**3**	**5**	**10**	**20**
Gas Furnace	8.4	7.4	4.7	4.9
Non linear System	9.2	7.5	13	18
Wiener-Hamnerstein	17	4.7	6.1	7.3

The CNN structure affects the system identification when the number of filters and the number of CNN structure are small. If they are large enough, for example, the number of filters is more than 20 in Table 6, and the number of CNN structures is more than 5, the modeling accuracies are not as affected by them.

6 Conclusion

The main contribution of the paper is that we analyze the impact of random weights in the identification of nonlinear systems. To do this, we use convolutional neural networks and frequency domain analysis. From the theoretical analysis and simulations, we can see that different random distributions for CNN filters have good modeling performances. Although there are optimal weights, we need the pre-training and the fine training to get closer to them. The pre-training of the hidden weights (filters) needs more calculation time. Three benchmark problems show the impact of random weights in the identification of nonlinear systems. Further works will develop the online version of this algorithm.

References

[1] J. C. A. Barata, M. S. Hussein, The Moore-Penrose pseudoinverse: A tutorial review of the theory, Brazilian Journal of Physics, 42(1), 146-165, 2012.

[2] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, Greedy layer-wise training of deep networks, in: Proceedings of the Advances in Neural Information Processing Systems (NIPS'06), 153-160, 2007.

[3] Y. Bengio, O. Delalleau, Justifying and generalizing contrastive divergence, Neural Computation, 21(6), 1601-1621, 2009.

[4] M. Binkowski, G. Marti, P. Donnat, Autoregressive convolutional neural networks for asynchronous time series, arXiv:1703.04122, 2017

[5] K. De Brabanter, P. Dreesen, P. Karsmakers, K. Pelckmans, J. De Brabanter, J. Suykens and B. De Moor, Fixed-size LS-SVM applied to the Wiener-Hammerstein benchmark. in: Proceedings of the 15th IFAC Symposium on System Identification, 826-831, 2009.

[6] E.Brigham, The fast Fourier Transform and Its Appllications, Prentice Hall, Englewood Cliffs, NJ, 1988.

[7] N. Cheney, M. Schrimpf, G. Kreiman, On the robustness of convolutional neural networks to internal architecture and weight perturbations, arXiv:1703.08245, 2017.

[8] E. de la Rosa and W.Yu, Randomized algorithms for nonlinear system identification with deep learning modification, Information Sciences, 364, 197-212, 2016.

[9] S. Genc, Parametric system identification using deep convolutional neural networks, in: Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN17), 2112-2119, 2017

[10] J. Gonzalez, W. Yu, Non-linear system modeling using LSTM neural networks, in: Proceedings of the 2nd IFAC Conference on Modelling Identification and Control of Nonlinear Systems (MICNON 2018), 485-490, 2018.

[11] R.Gray, Toeplitz and circulant matrices: A review. Foundations and Trends in Communications and Information Theory, 2(3):155-239, 2006.

[12] N. Hatami, Y. Gavet and J. Debayle, Classification of time-series images using deep convolutional neural networks, arXiv:1710.00886v2, 2017

[13] K. He, X. Zhang, S. Ren, J. Sun, Spatial pyramid pooling in deep convolutional networks for visual recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence, 37(9), 1904-1916, 2015.

[14] G. Hinton, S. Osindero, Y. Teh, A fast learning algorithm for deep belief nets, Neural Computation, 18 (7), 2006.

[15] S. Hochreiter and J. Schmidhuber, Long short-term memory, Neural computation, 9(8), 1735-1780, 1997

[16] R. Horn, C.J ohnson, Matrix Analysis, 2nd Ed., Cambridge University Press, 2012

[17] B. Igelnik and Y-H.Pao, Stochastic choice of basis functions in adaptive function approximation and the functional-link met, IEEE Transactions on Neural Networks, 6 (2),1320-1329, 1995.

[18] G. Box, G. Jenkins, G. Reinsel. Time Series Analysis: Forecasting and Control, 4th Ed, Wiley, 2008.

[19] Y. Kang, S. Chen, X. Wang, and Y. Cao, Deep convolutional identifier for dynamic modeling and adaptive control of unmanned helicopter, IEEE Trans. on Neural Networks and Learning System, DOI: 10.1109/TNNLS.2018.2844173, 2018

[20] A. Krizhevsky, I. Sutskever, G. Hinton, Imagenet classification with deep convolutional neural networks, Advances in neural information processing systems, 1097-1105, 2015.

[21] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, L.D. Jackel, Backpropagation applied to handwritten zip code recognition, Neural Computation, 1(4), 541-551, 1989.

[22] Y. LeCun, L. Bottou, Y. Bengio, P. Haffne, Gradient based learning applied to document recognition, Proceeding of the IEEE, 1998.

[23] H. Lee, R. Groose, R. Ranganath, and A. Ng, Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations, in: Proceedings of the 26th International Conference on Machine Learning (ICML09), 609-616, 2009.

[24] M. Li and D. Wang, Insights into randomized algorithms for neural networks: Practical issues and common pitfalls, Information Sciences, 382, 170-178,2016.

[25] M. Lopez and W. Yu, Nonlinear system modeling using convolutional neural networks, in: Proceedings of the14th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE17), 2017.

[26] M. Matusugu, M. Katsuhiko, M. Yusuke, K. Yuji, Subject independent facial expression recognition with robust face detection using a convolutional neural network, Neural Networks, 16(5), 555-559, 2003.

[27] K.Narendra, K.Parthasarathy, Identification and control of dynamical systems using neural networks, IEEE Transactions on Neural Networks, 1(1), 4-27, 1990.

[28] A. Saxe, P. Koh, Z. Chen, M. Bhand, B. Suresh, A. Ng, On random weights and unsupervised feature learning, in: Proceedings of the 28th International Conference on Machine Learning, 1089–1096, 2011.

[29] S. Scardapane and D. Wang, Randomness in neural networks: An overview, WIREs Data Mining and Knowledge Discovery, doi: 10.1002/widm.1200, 2017.

[30] J.Schlemper, J.Caballero, J.Hajnal, A.Price, D.Rueckert, A deep cascade of convolutional neural networks for dynamic mr image reconstruction, IEEE transactions on medical imaging, 37, 491-503, 2018.

[31] W. F. Schmidt, M. A. Kraaijveld, R. P. W. Duin, Feedforward neural networks with random weights, in: Proceedings of the 11th IAPR International Conference on Pattern Recognition, 1-4, 1992.

[32] J. Schoukens, J. Suykens, L. Ljung, Wiener-Hammerstein benchmark, in: Proceedings of the 5th IFAC Symposium on System Identification, 2009.

[33] P. Sermanet, Y. LeCun, Traffic sign recognition with multi-scale convolutional networks, in: Proceedings of the 2011 International Joint Conference on Neural Networks (IJCNN), 2011.

[34] N.Srivastava, G.Hinton, A.Krizhevsky, I.Sutskever, R.Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting. Journal of machine learning research, 15 (1), 1929-1958, 2014.

[35] M. Sugeno, T. Yasukawa, A fuzzy logic based approach to qualitative modeling, IEEE Trans. on Fuzzy Systems, 1 (1), 7-31, 1993.

[36] J. Tapson and A. van Schaik, Learning the pseudoinverse solution to network weights, Neural Networks, 45, 94-100, 2013.

[37] D. Wang and M. Li, Stochastic configuration networks: Fundamentals and algorithms. IEEE Transactions on Cybernetics, 47(10), 3466-3479, 2017

[38] L. Wang and R. Langari, Complex systems modeling via fuzzy logic, IEEE Trans. on Syst., Man, and Cybernetics, 26 (1), 100-106, 1996.

[39] M.D. Zeiler, Hierarchical Convolutional Deep Learning in Computer Vision, Ph.D. Thesis, 2013.