Chapter 8

Applications of General Regression Neural Networks in Dynamic Systems

Ahmad Jobran Al-Mahasneh, Sreenatha Anavatti, Matthew Garratt and Mahardhika Pratama

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.80258

Abstract

Nowadays, computational intelligence (CI) receives much attention in academic and industry due to a plethora of possible applications. CI includes fuzzy logic (FL), evolutionary algorithms (EA), expert systems (ES) and artificial neural networks (ANN). Many CI components have applications in modeling and control of dynamic systems. FL mimics the human reasoning by converting linguistic variables into a set of rules. EA are metaheuristic population-based algorithms which use evolutionary operations such as mutation, crossover, and selection to find an optimal solution for a given problem. ES are programmed based on an expert knowledge to make informed decisions in complex tasks. ANN models how the neurons are connected in animal nervous systems. ANN have learning abilities and they are trained using data to make intelligent decisions. Since ANN have universal approximation abilities, they can be used to solve regression, classification, and forecasting problems. ANNs are made of interconnected layers where every layer is made of neurons and these neurons have connections with other neurons. These layers consist of an input layer, hidden layer/layers, and an output layer.

Keywords: applications, general regression, neural networks, dynamic systems

1. Introduction

Nowadays, computational intelligence (CI) receives much attention in academic and industry due to a plethora of possible applications. CI includes fuzzy logic (FL), evolutionary algorithms (EA), expert systems (ES), and artificial neural networks (ANN). Many CI components have applications in modeling and control of dynamic systems. FL mimics the human reasoning by converting linguistic variables into a set of rules. EA are metaheuristic population-based

algorithms which use evolutionary operations such as mutation, crossover, and selection to find an optimal solution for a given problem. ES are programmed based on an expert knowledge to make informed decisions in complex tasks. ANN model how the neurons are connected in animal nervous systems. ANN have learning abilities and they are trained using data to make intelligent decisions. Since ANN have universal approximation abilities [1], they can be used to solve regression, classification, and forecasting problems. ANNs are made of interconnected layers where every layer is made of neurons, and these neurons have connections with other neurons. These layers consist of an input layer, hidden layer/layers, and an output layer. ANN have two major types as shown in **Figure 1**: feed-forward neural network (FFNN) and recurrent neural network (RNN). In FFNN, the data can only flow from the input to hidden layer, while in RNN, the data can flow in any direction. The output of a single-hidden-layer FFNN can be written as

$$Y = (W_{HO}\, h(\mathbf{x}\, W_{IH} + \mathbf{b_I})) + \mathbf{b_O} \tag{1}$$

where Y is the network output, W_{HO} is the hidden-output layers weights matrix, h is the hidden layer activation function, $\mathbf{x}$ is the input vector, W_{IH} is the input-hidden layers weights matrix, $\mathbf{b_I}$ is the input layer bias vector, and $\mathbf{b_O}$ is the hidden layer bias vector.

The output of a single-hidden-layer RNN with a recurrent hidden layer can be written as

$$Y = (W_{HO}\, h(\mathbf{x}\, W_{IH} + \mathbf{h_{t-1}}\, W_{HH} + \mathbf{b_I})) + \mathbf{b_O} \tag{2}$$

The training of neural networks involves modifying the neural network parameters to reduce a given error function. Gradient descent (GD) [2, 3] is the most common ANN training method:

$$\theta_{new} = \theta_{old} - \lambda \frac{\partial E}{\partial \theta} \tag{3}$$

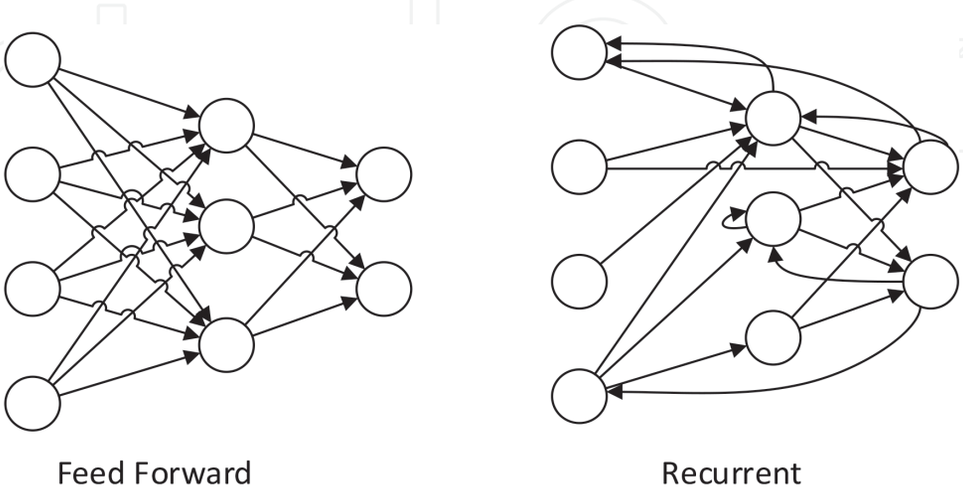

Figure 1. Feed-forward and recurrent networks.

where θ are the network parameters, λ is the learning rate, and E is the error function:

$$E = \frac{1}{N}\sum_{I=1}^{N}(y-t)^2 \tag{4}$$

where N is the number of samples, y is the network output, and t is the network target.

2. General regression neural network (GRNN)

The general regression neural network (GRNN) is a single-pass neural network which uses a Gaussian activation function in the hidden layer [4]. GRNN consists of input, hidden, summation, and division layers.

The regression of the random variable y on the observed values X of random variable x can be found using

$$E[y|X] = \frac{\int_{-\infty}^{\infty} yf(X,y)dy}{\int_{-\infty}^{\infty} f(X,y)dy} \tag{5}$$

where $f(X,y)$ is a known joint continuous probability density function.

When $f(X,y)$ is unknown, it should be estimated from a set of observations of x and y. $f(X,y)$ can be estimated using the nonparametric consistent estimator suggested by Parzen as follows:

$$\hat{f}(X,Y) = \frac{1}{2\pi^{(p+1)/2}\,\sigma^{(p+1)}}\frac{1}{n}\sum_{i=1}^{n} e^{-\frac{\left(X-X^i\right)^T\left(X-X^i\right)}{2\sigma^2}} e^{-\frac{\left(Y-Y^i\right)^2}{2\sigma^2}} \tag{6}$$

where n is the number of observations, p is the dimension of the vector variable, and x and σ are the smoothing factors.

Substituting (6) into (5) leads to

$$\hat{Y}(X) = \frac{\sum_{i=1}^{n} e^{\frac{-\left(X-X^i\right)^T\left(X-X^i\right)}{2\sigma^2}} \int_{-\infty}^{\infty} ye^{-\frac{\left(Y-Y^i\right)^2}{2\sigma^2}}dy}{\sum_{i=1}^{n} e^{\frac{-\left(X-X^i\right)^T\left(X-X^i\right)}{2\sigma^2}} \int_{-\infty}^{\infty} e^{-\frac{\left(Y-Y^i\right)^2}{2\sigma^2}}dy} \tag{7}$$

After solving the integration, the following will result:

$$\hat{Y}(X) = \frac{\sum_{i=1}^{n} ye^{\frac{-\left(X-X^i\right)^T\left(X-X^i\right)}{2\sigma^2}}}{\sum_{i=1}^{n} e^{\frac{-\left(X-X^i\right)^T\left(X-X^i\right)}{2\sigma^2}}} \tag{8}$$

2.1. Previous studies

GRNN was used in different applications related to modeling, system identification, prediction, and control of dynamic systems including: feedback linearization controller [5], HVAC

process identification and control [6], modeling and monitoring of batch processes [7], cooling load prediction for buildings [8], fault diagnosis of a building's air handling unit [9], intelligent control [10], optimal control for variable-speed wind generation systems [11], annual power load forecasting model [12], vehicle sideslip angle estimation [13], fault diagnosis for methane sensors [14], fault detection of excavator's hydraulic system [15], detection of time-varying inter-turn short circuit in a squirrel cage induction machine [16], system identification of nonlinear rotorcraft heave mode [17], and modeling of traveling wave ultrasonic motors [18].

Some significant modifications of GRNN include using fuzzy c-means clustering to cluster the input data of GRNN [19], modified GRNN which uses different types of Parzen estimators to estimate the density function of the regression [20], density-driven GRNN combining GRNN, density-dependent kernels and regularization for function approximation [21], GRNN to model time-varying systems [22], adapting GRNN for modeling of dynamic plants [23] using different adaptation approaches including modifying the training targets, and adding a new pattern and dynamic initialization of σ.

2.2. GRNN training algorithm

GRNN training is rather simple. The input weights are the training inputs transposed, and the output weights are the training targets. Since GRNN is an associative memory, after training, the number of the hidden neurons is equal to the number of the training samples. However, this training procedure is not efficient if there are many training samples, so one of the suggested solutions is using a data dimensionality reduction technique such as clustering or principal component analysis (PCA). One of the novel solutions to data dimensionality reduction is using an error-based algorithm to grow GRNN [24] as explained in **Algorithm 1**. The algorithm will check whether an input is required to be included in the training, based on prediction error before training GRNN with that input. If the prediction error without including that input is more than the certain level, then GRNN should be trained with it.

Algorithm 1 GRNN growing algorithm

1: Train GRNN with 10% of the training data

2: **for** $i \le i_{final}$ **do**

3: Find the output of GRNN y_i of input$_i$

4: Find the MSE: $\frac{1}{2}(y_i - Target_i)^2$

5: **if** MSE>Threshold **then**

6: Train GRNN with the input$_i$ Do not train GRNN with the input$_i$

7: **end if**

8: **end if**

Dataset	Training error after/before k-means MSE	Testing error after/before k-means MSE	Size reduction %
Abalone	0.0177/0.002	0.0141/0.006	99.76
Building energy	0.047/3.44e-05	0.0165/0.023	99.76
Chemical sensor	0.241/0.016	0.328/0.034	97.99
Cholesterol	0.050/4.605e-05	0.030/0.009	92

Table 1. Using GRNN with k-means clustering.

2.2.1. Reducing data dimensionality using clustering

Clustering techniques can be used to reduce the data dimensionality before feeding it to the GRNN. k-means clustering is one of the popular clustering techniques. The k-means clustering algorithm is explained in **Algorithm 2**. Also, results of comparing GRNN performance before and after applying k-means algorithm are shown in **Table 1**. Although the training and testing errors will increase, there are large reductions in the network size.

The aim of the algorithm is to minimize the distance objective function:

$$J = \sum_{i=1}^{N} \sum_{j=1}^{M} \left\| x_i - c_j \right\|^2 \tag{9}$$

Algorithm 2 : K-means clustering algorithm

1: Randomly select the number of clusters k and their centers c_i

2: **for** $i \le i_{final}$ **do**

3: Calculate the distance between the data point and the clusters centers c_i.

4: Assign every data sample to its closest cluster.

5: Calculate the clusters centers c_i again.

6: **end for**

2.2.2. Reducing data dimensionality using PCA

PCA can be used to reduce a large dataset into a smaller dataset which still carries most of the important information from the large dataset. In a mathematical sense, PCA converts a number of correlated variables into a number of uncorrelated variables. PCA algorithm is explained in **Algorithm 3**.

2.3. GRNN output algorithm

After GRNN is trained, the output of GRNN can be calculated using

Algorithm 3 : PCA algorithm

1: Find the dot product matrix $X^T X = \sum_{i=1}^{N} (X_i - \mu)^T (X_i - \mu)$

2: Find the eigenvalues of $X^T X = XV\Lambda^T$

3: Find the eigenvectors $U = XV\Lambda^{-1/2}$

4: Find d number of principal components $U_d = [u_1, u_2, ..u_d]$

5: Compute $Y = U_d^T X$

6: Find the covariance matrix of Y as $YY^T = U^T XX^T U = \Lambda$

7: Find $W = U\Lambda^{-1/2}$

Dataset	Training error after/before PCA MSE	Testing error after/before PCA MSE	Size reduction %
Abalone	0.197/0.002	0.188/0.006	99.8
Building energy	0.061/3.44e-05	0.049/0.023	99.6
Chemical sensor	0.241/0.016	0.328/0.034	98.3
Cholesterol	0.026/4.605e-05	0.028/0.009	92

Table 2. Using GRNN with PCA.

$$D = (X - W_i)^T (X - W_i) \tag{10}$$

$$\hat{Y} = \frac{\sum_{i=1}^{N} W_o e^{(D/2\sigma^2)}}{\sum_{i=1}^{N} e^{(D/2\sigma^2)}} \tag{11}$$

where D is the Euclidean distance between the input X and the input weights W_i, W_o is the output weight, and σ is the smoothing factor of the radial basis function.

GRNN output calculation is explained in **Algorithm 4**.

Algorithm 4 GRNN output

1: **procedure** $\hat{Y}$ X

2: Find the Eucldian distance D using (10)

3: Find the numerator of (11)

4: Find the denominator (11)

5: Divide the numerator over denominator

6: **return** $\hat{Y}$

7: **end procedure**

Other distance measures can be also used such as Manhattan (city block), so (10) will become

$$D = X - W_i \tag{12}$$

3. Estimation of GRNN smoothing parameter (σ)

Since σ is the only free parameter in GRNN and suitable values of it will improve GRNN accuracy, it should be estimated. Since there is no optimal analytical solution for finding σ, numerical approaches can be used to estimate it. The holdout method is one of the suggested methods. In this method, samples are randomly removed from the training dataset; then using the GRNN with a fixed σ, the output is calculated using the removed samples; then the error is calculated between the network outputs and the sample targets. This procedure is repeated for different σ values. The smoothing parameter (σ) with the lowest sum of errors is selected as the best σ. The holdout algorithm is explained in **Algorithm 5**.

Algorithm 5 Holdout method to find GRNN σ

1: Initialize GRNN σ vector $[\sigma_{low} : \Delta\sigma : \sigma_{up}]$
2: **for** $i \le i_{final}$ **do**
3: **for** $j \le j_{final}$ **do**
4: Randomly select datasample (data (j)) and remove it from the training data
5: Train GRNN with the reduced dataset
6: Find GRNN output $y(j)$ using the reduced sample (data (j))
7: Find GRNN $MSE(j)$ between $y(j)$ and the reduced sample data target
8: **end for**
9: Find the summation of the MSE
10: Find σ with the lowest summation of the MSE
11: **end for**

Other search and optimization methods might be also used to find σ. For instance, genetic algorithms (GA) and differential evolution (DE) are suitable options. **Algorithm 6** explains how to find σ using DE or GA. Also, the results of using DE and GA are depicted in **Figure 2**.

Algorithm 6 DE/GA find GRNN σ

1: Divide the data into training dataset and testing dataset
2: Train GRNN with the training data
3: **for** $i \le maxiterations$ **do**
4: Generate a random population of σ s
5: Evaluate the cost of each *sigma*
6: Perform mutation, crossover and selection
7: Find the best σ with the lowest MSE
8: **end for**

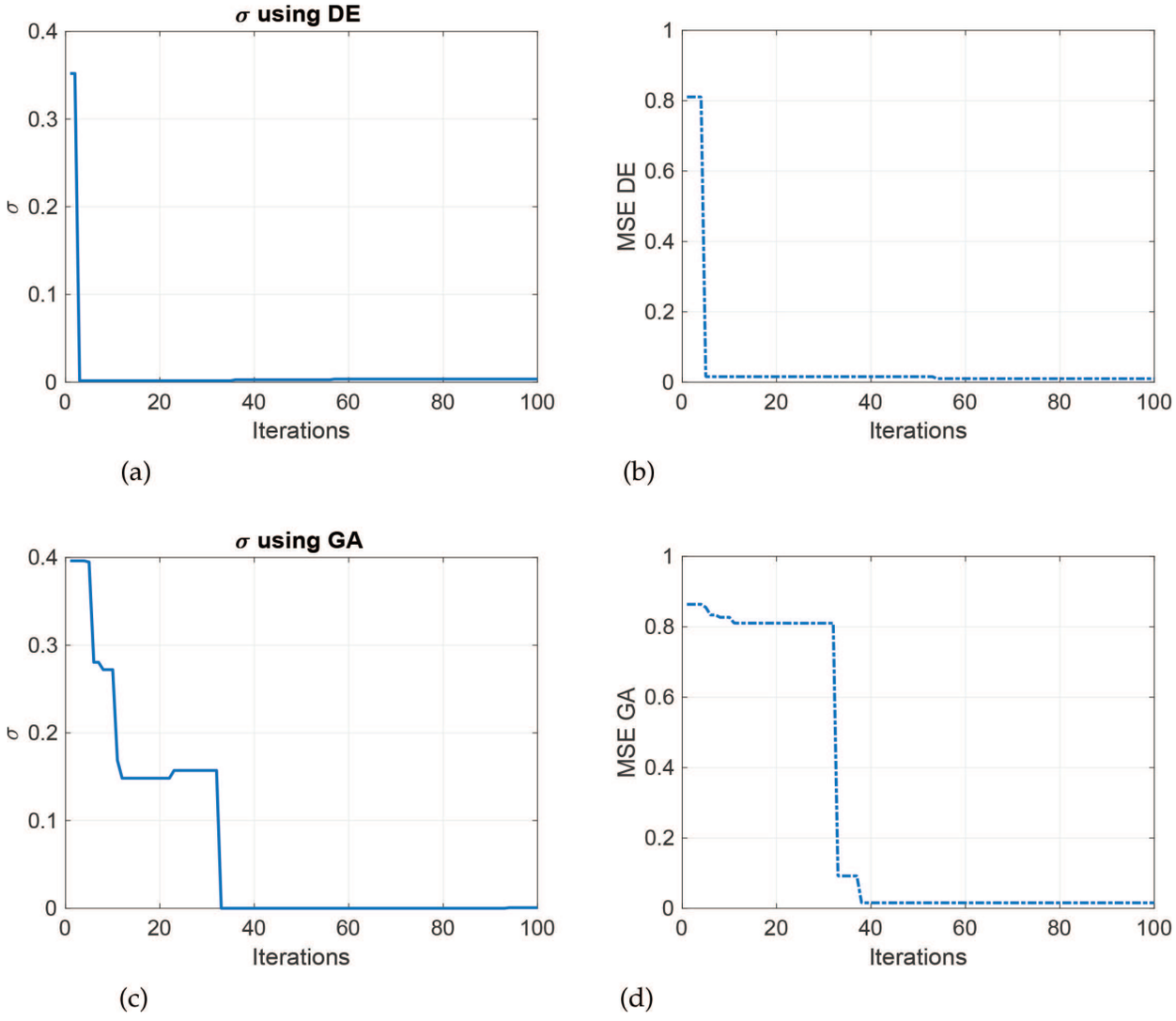

Figure 2. DE and GA used to estimate GRNN σ. (a) Estimation of σ using DE (b) MSE evolution when using DE to estimate s (c) Estimation of σ using GA (d) MSE evolution when using GA to estimate σ.

Both of GA and DE can find a good approximation of σ within 100 iterations only; however, DE converges faster since it is a vectorized algorithm.

4. GRNN vs. back-propagation neural networks (BPNN)

There are many differences between GRNN and BPNN. Firstly, GRNN is single-pass learning algorithm, while BPNN needs two passes: forward and backward pass. This means that GRNN consumes significantly less training time. Secondly, the only free parameter in GRNN is the smoothing parameter σ, while in BPNN more parameters are required such as weights, biases, and learning rates. This also indicates that GRNN quick learning abilities and its suitability for online systems or for system where minimal computations are required. Also, another difference is that since GRNN is an autoassociative memory network, it will store all the distinct input/output samples while BPNN has a limited predefined size. This size growth

Type	Dataset	Training time (sec)	Training error (MSE)	Testing error (MSE)
GRNN	Abalone	0.621	0.342	0.384
BPNN	Abalone	1.323	0.436	0.395
GRNN	Building energy	0.630	0.0731	0.628
BPNN	Building energy	1.880	0.1152	0.631
GRNN	Chemical sensor	0.701	0.888	1.316
BPNN	Chemical sensor	1.473	0.228	1.584
GRNN	Cholesterol	0.801	0.037	0.172
BPNN	Cholesterol	2.099	0.061	0.215

Table 3. GRNN vs. BPNN training and testing performance.

issue is resolved by either using clustering or PCA (read Sections 2.21 and 2.2.2). Finally, GRNN is based on the general regression theory, while BPNN is based on gradient-descent iterative optimization method.

To show the advantages of GRNN over BPNN, a comparison is held using standard regression datasets built inside MATLAB software [25]. For all the datasets, they are divided 70% for training and 30% for testing. After training the network with the 70% training data, the output of the neural network is found using the remaining testing data. The most notable advantage of GRNN over BPNN is the shorter training time which confirms its selection for dynamic systems modeling and control. Also, GRNN has less testing error which means it has better generalization abilities than BPNN. The comparison results are summarized in **Table 3**.

5. GRNN in identification of dynamic systems

System identification is the process of building a model of unknown/partially known dynamic system based on observed input/output data. Gray-box and black-box identification are two common approaches of system identification. In the gray-box approach, a nominal model of a dynamic system is known, but its exact parameters are unknown, so an identifier is used to find these parameters. In the black-box approach, the identification is based only on the data. Examples of black-box identification include fuzzy logic (FL) and neural networks (NN). GRNN can be used to identify dynamic systems quickly and accurately. There are two methods to use GRNN for system identification: the batch mode (off-line training) and sequential mode (online training). In the batch mode, all the observed data is available before the system identification, so GRNN can be trained with a big chunk of the data, while in the sequential mode only a few data samples are available for identification.

5.1. GRNN identification in batch training mode

In the batch mode, the observed data should be divided into training, validation, and testing. GRNN will be fed with all the training data to identify the system. Then in the validation stage, the network should be tested with different data, usually randomly selected, and the error is

recorded for every validation test. Then the validation process is repeated several times. Usually 10 times is standard. And then the average validation error is found based on all the validation tests. This validation procedure is called k-fold cross validation a standard technique in machine learning (ML) applications. To test the generalization ability of an identified model, a new dataset is used called testing dataset. Based on the model performance in the testing stage, one can decide whether the model is suitable or not.

5.1.1. Batch training GRNN to identify hexacopter attitude dynamics

In this example, GRNN is used to identify the attitude (pitch/roll/yaw) of a hexacopter drone based on real flight test data in the free flight mode. The data consist of three inputs: rolling, pitching, and yawing control values and three outputs: rolling, pitching, and yawing rates. The dataset contains 6691 data samples with a sample rate of 0.01 seconds. A total of 4683 samples are used to train GRNN in the batch mode, and the remaining data samples (2008) are used for testing. The results of hexacopter attitude identification are shown in **Figure 3(a–c)**. The results are accurate with very low error. MSE in training stage is 0.001139 and 0.00258 in the testing stage. Also, the training time was only 0.720 seconds.

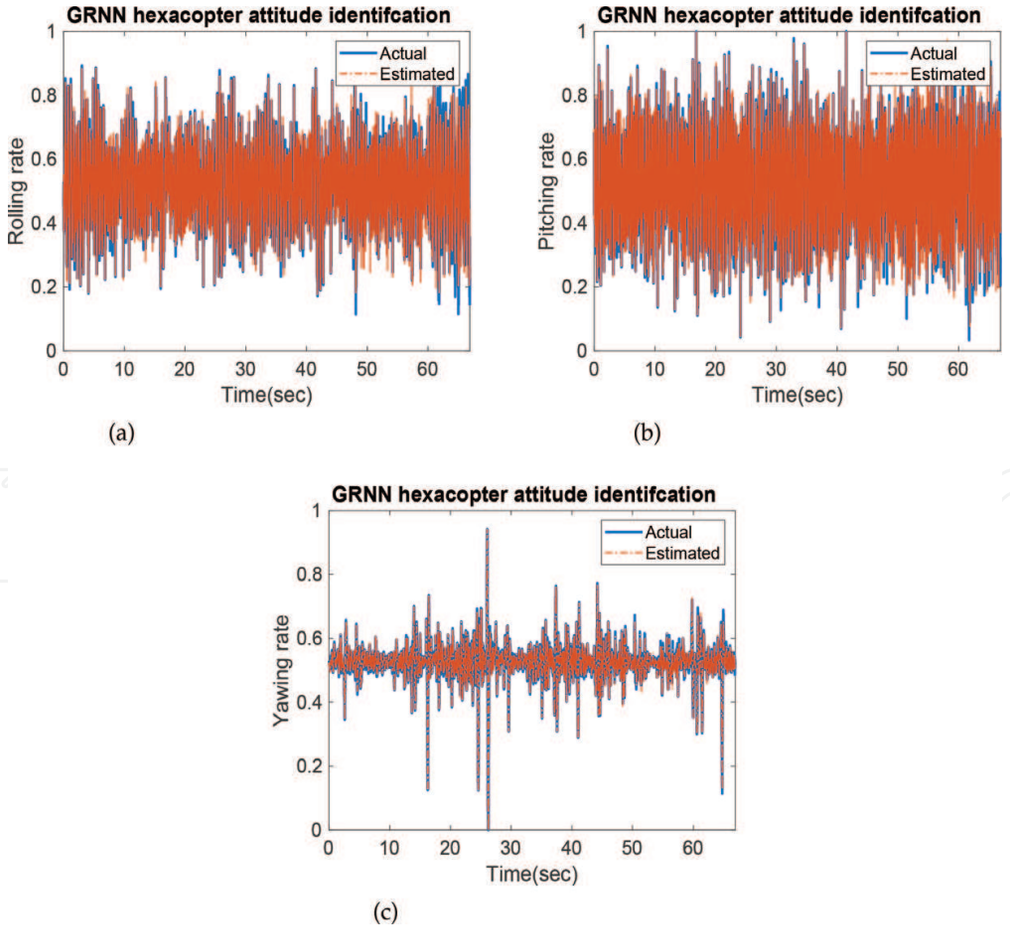

Figure 3. Attitude identification of hexacopter in batch training: (a) rolling rate identification, (b) pitching rate identification, and (c) yawing rate identification.

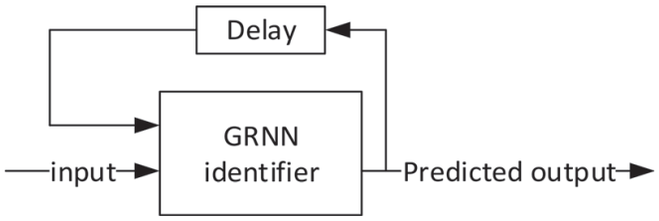

Figure 4. Sequential training GRNN.

5.2. GRNN identification in sequential training mode

In sequential training, the data flow once at a time which makes using the batch training procedures impossible. So GRNN should be able to find the system model from only the current and past measurements. So it is a prediction problem. Since GRNN converges to a regression surface even with a few data samples and since it is accurate and quick, it can be used in the online dynamic systems identification.

5.2.1. Sequential training GRNN to identify hexacopter attitude dynamics

To use GRNN in sequential mode, it is preferred to use the delayed output of the plant as an input in addition to the current input as shown in **Figure 4**. The same data which was used for batch mode is used in the sequential training. The inputs to GRNN are the control values of

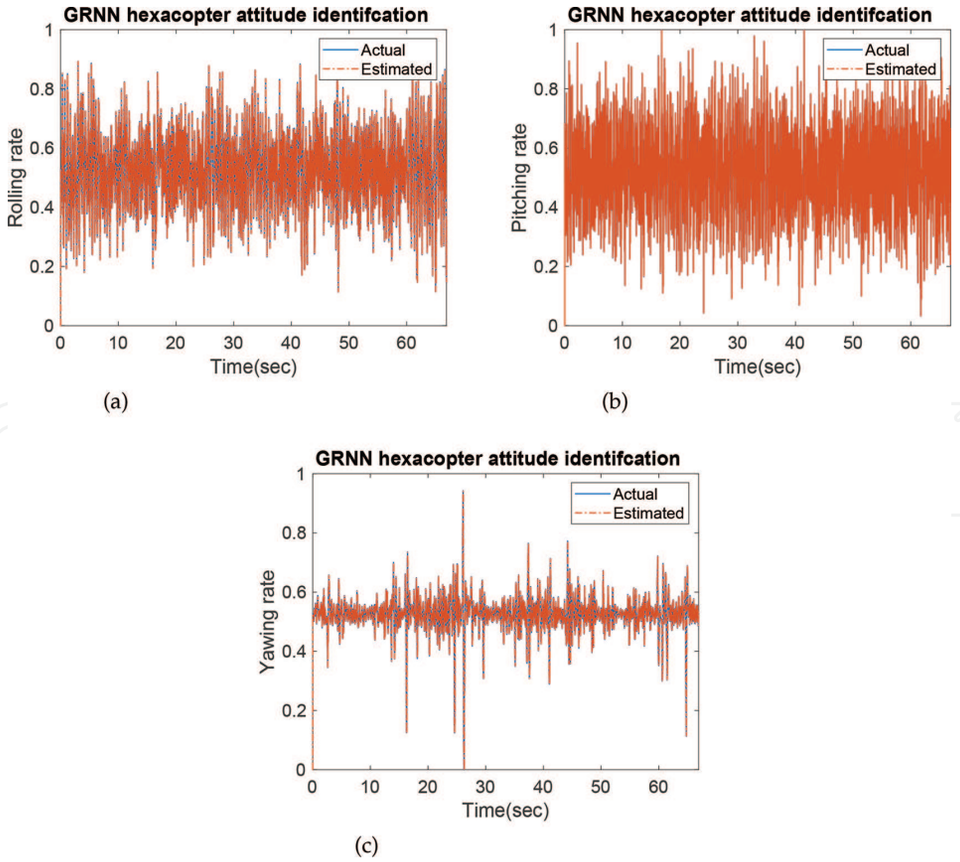

Figure 5. Attitude identification of hexacopter in sequential training: (a) rolling rate identification, (b) pitching rate identification, and (c) yawing rate identification.

rolling, pitching, and yawing and the delayed rolling, pitching, and yawing rates. The results of using GRNN in the sequential training mode are shown in **Figure 5(a–c)**. The results of sequential training are more accurate than the results in batch training.

6. GRNN in control of dynamic systems

The aim of adding a closed-loop controller to the dynamic systems is either to reach the desired performance or stabilize the unstable system. GRNN can be used in controlling dynamic systems as a predictive or feedback controller. GRNN in control systems can be used as either supervised or unsupervised. When GRNN is trained as a predictive then the controller input and output data are known, so this is a supervised problem. On the other hand, if GRNN is utilized as a feedback controller (see **Figure 6**) without being pretrained, only the controller input data is known so GRNN have to find the suitable control signal u.

6.1. GRNN as predictive controller

To utilize GRNN as a predictive controller, it should be trained with input-output data from another controller. For example, training a GRNN with a proportional integral derivative (PID) controller input/output data as shown in **Figure 7**. Then the trained GRNN can be used as a controller.

6.1.1. Example 1: GRNN as predictive controller

If we have a discrete time system Liu [26] described as

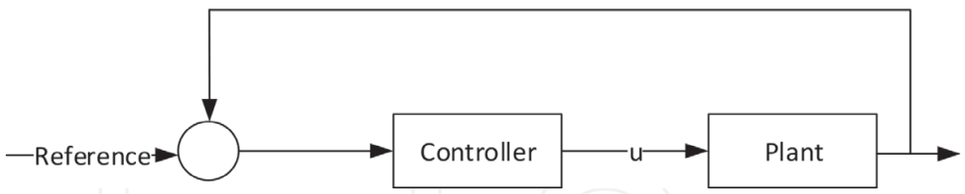

Figure 6. Unsupervised learning problem in control.

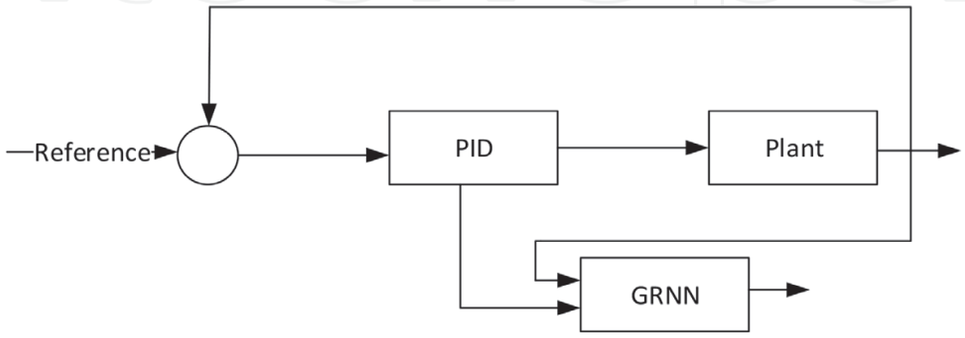

Figure 7. Training GRNN as predictive controller.

$$y(k+1) = 0.8 * sin\,(y(k)) + 15 * u(k) \tag{13}$$

The desired reference is $y_d(k) = 2^* sin\,(0.1\pi t)$.

The perfect control law can be written as

$$u(k) = \frac{y_d(k+1)}{15} - \frac{0.8 * sin\,(y(k))}{15} \tag{14}$$

To train GRNN as a predictive controller, the system described in (13) and (14) is simulated for 50 seconds. Then the controller output u and the plant output y were stored. GRNN is trained with the plant output as input and the controller output as output. For any time step the plant output is fed to GRNN, and the controller output u is estimated. The estimated controller output by GRNN and the perfect controller output are almost identical as shown in **Figure 8**. Also, the tracking performance after using GRNN as a predictive controller is very accurate as shown in **Figure 9**.

6.2. GRNN as an adaptive estimator controller

Since GRNN has robust approximation abilities, it can be used to approximate the dynamics of a given system to find the control law especially if the system is partially known or unknown.

Assume there is a nonlinear dynamic system written as

$$\dot{x} = f(x,t) + bu + d \tag{15}$$

where $\dot{x}$ is the derivative of the states, $f(x,t)$ is a known function of the states, b is the input gain, and d is the external disturbance.

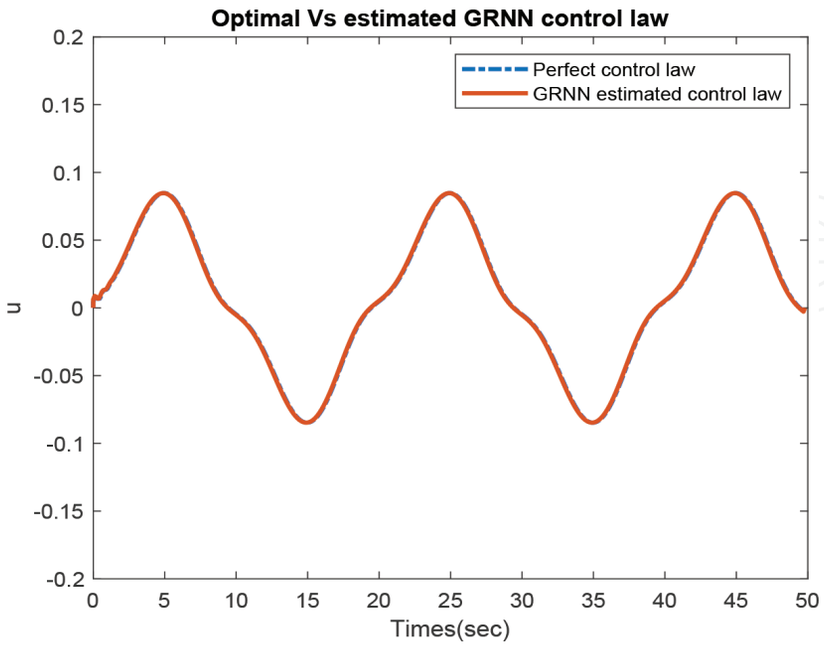

Figure 8. Perfect vs. estimated GRNN controller output.

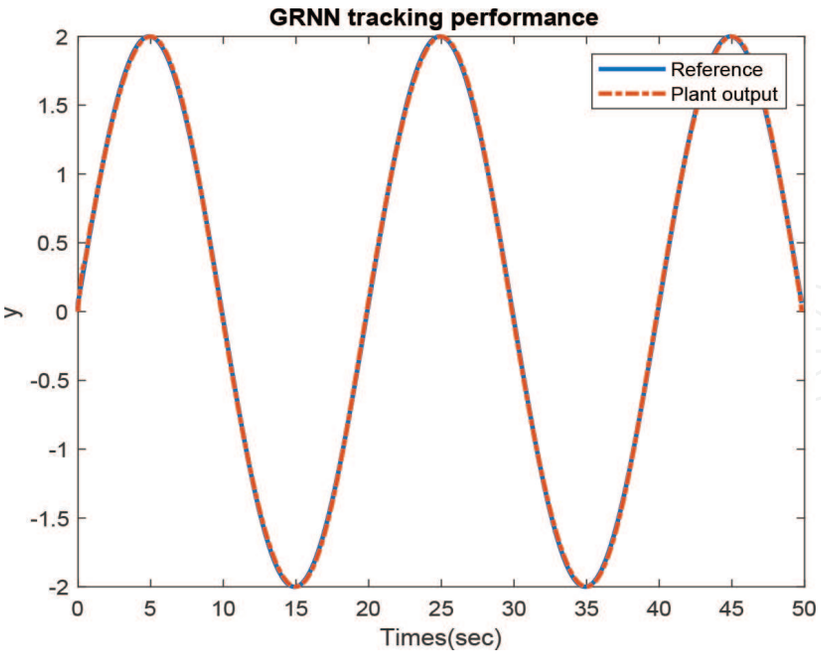

Figure 9. GRNN tracking performance.

The perfect control law can be written as

$$u = \frac{1}{b}(\dot{x} - f(x,t) - d) \tag{16}$$

If $f(x,t)$ is unknown, then the control law in (16) cannot be found; hence, the alternative is using GRNN to estimate the unknown function $f(x,t)$. To derive the update law of GRNN weights, let us define the objective function as MSE error function as follows:

$$E = \frac{1}{2}(\hat{y} - y)^2 \tag{17}$$

where $\hat{y}$ is the estimation of GRNN and y is the optimal value of $f(x,t)$. To derive the update law of the GRNN weights, the error should be minimized with respect to GRNN weights W:

$$\frac{\partial E}{\partial W} = \left(\hat{W}H - y\right) * H \tag{18}$$

where $\hat{W}$ is the GRNN current hidden-output layers weights and H is the hidden layer output, so the update law of GRNN weights will be

$$W_{i+1} = W_i + H\left(\hat{W}H - y\right) \tag{19}$$

6.3. Example 2: using GRNN to approximate the unknown dynamics

Let us consider the same discrete as in example 1:

http://dx.doi.org/10.5772/intechopen.80258

$$y(k+1) = f(k) + 15*u(k) \tag{20}$$

The desired reference is $y_d(k) = 2^* \, sin\,(0.1\pi t)$

where $f(k)$ is unknown nonlinear function.

The perfect control law can be written as

$$u(k) = \frac{-f(k)}{15} + \frac{y_d(k)}{15} \tag{21}$$

GRNN is used to estimate the unknown function $f(k)$. With applying the update law in (19), $f(k)$ is estimated with an acceptable accuracy as shown in **Figure 10**. MSE between the ideal and the estimated $f(k)$ is 0.0033. The accurate controller tracking performance is also shown **Figure 11**.

6.4. GRNN as an adaptive optimal controller

GRNN has learning abilities which means it is suitable to be an adaptive intelligent controller. Rather than approximating the unknown function in the control law (16), one can use GRNN to approximate the whole controller output as shown in **Figure 12**. The same update law as in (19) can be used to update GRNN weights to approximate the controller output u.

6.4.1. Example 3: using GRNN as an adaptive controller

Let us consider the same discrete system as in (13):

$$y(k+1) = 0.8*\, sin\,(y(k)) + 15*u(k)$$

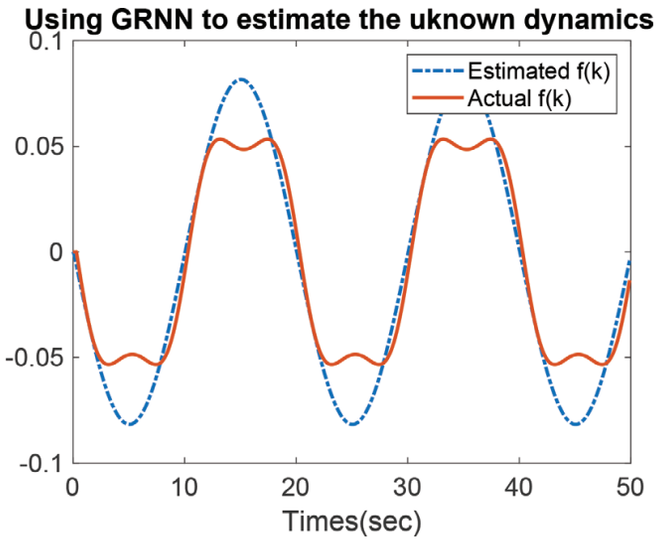

Figure 10. Using GRNN to estimate the unknown dynamics.

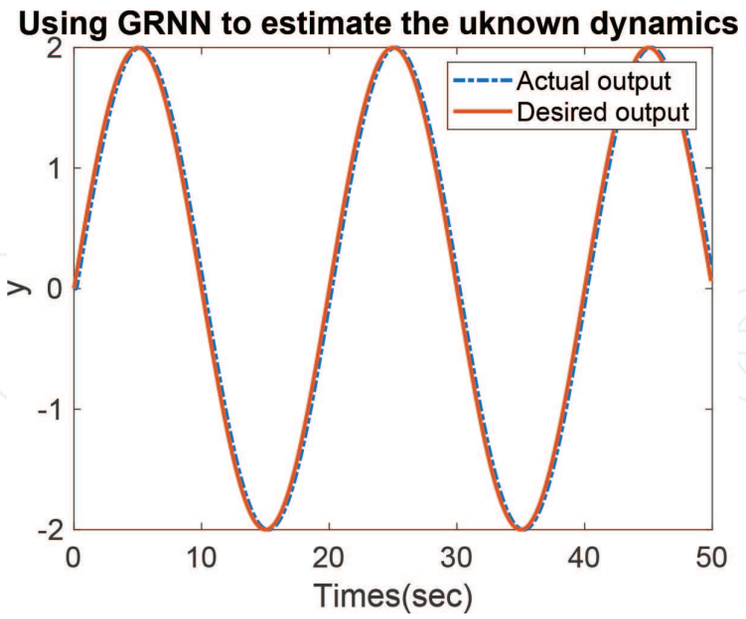

Figure 11. GRNN tracking performance for example 2.

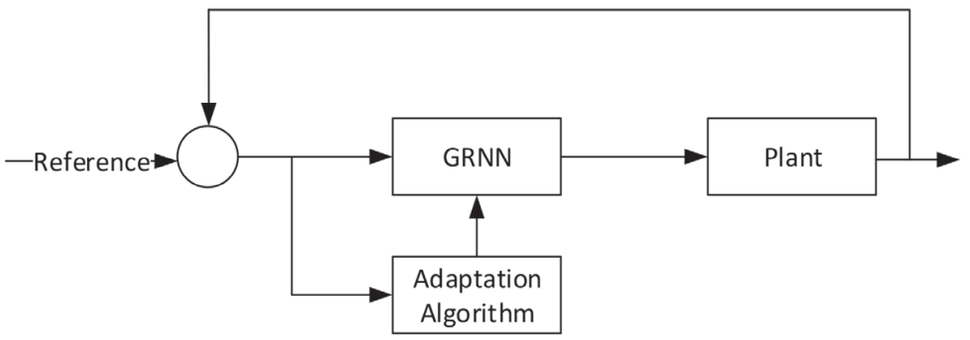

Figure 12. Training GRNN as an adaptive controller.

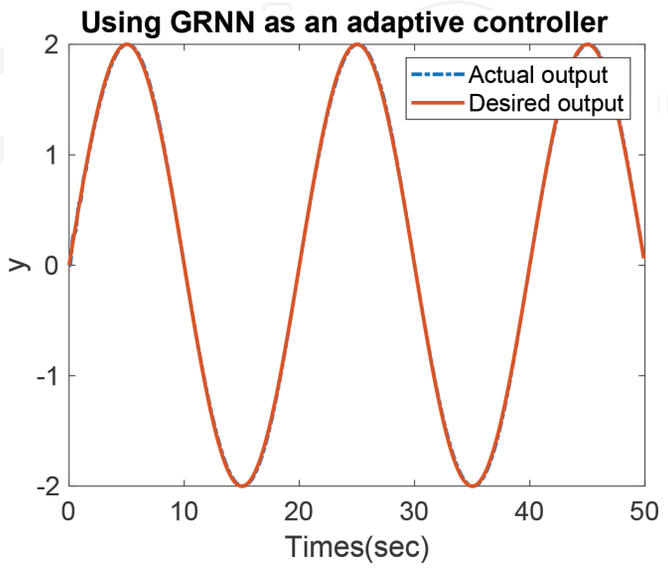

Figure 13. GRNN tracking performance for example 3.

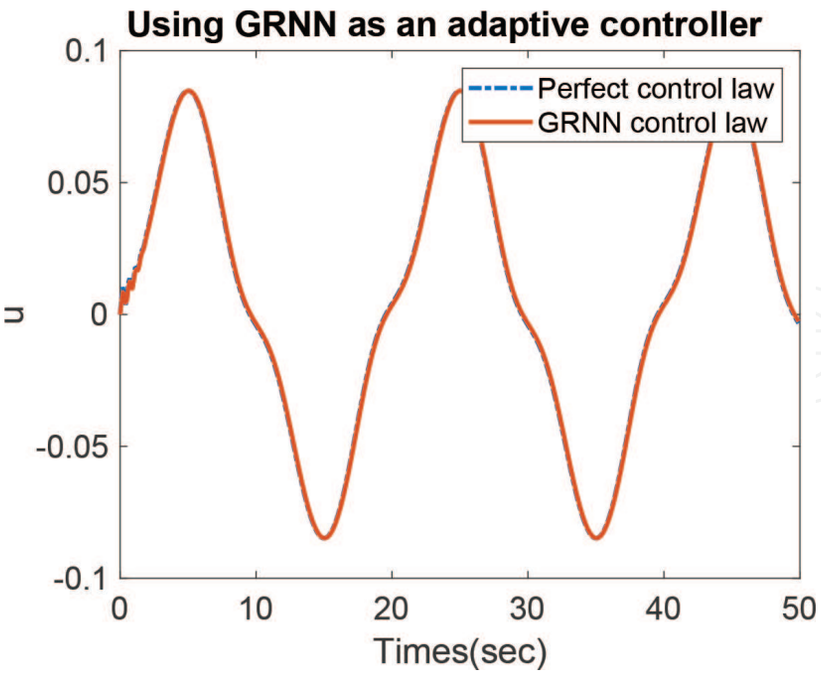

Figure 14. GRNN Estimated control law for example 3.

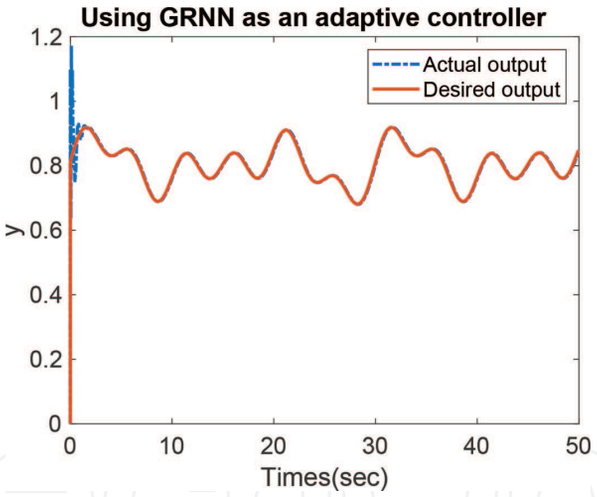

Figure 15. GRNN as an adaptive controller in Example 4.

with the same desired reference $y_d(k) = 2 * sin\,(0.1\pi t)$, but in this case GRNN is used to estimate the full controller output u as shown in **Figure 14** and the tracking performance is shown in **Figure 13**.

6.4.2. Example 4: using GRNN as an adaptive controller

Let us use GRNN to control a more complex discrete plant [27] described as

$$y(k+1) = 0.2\,cos\,(0.8(y(k)+y(k-1))) + 0.4\,sin\,(0.8(y(k-1)+y(k)+2u(k)+u(k-1))) + 0.1(9+y(k)+y(k-1)) + \frac{2(u(k)+u(k-1))}{(1+\,cos\,(y(k))} \quad (22)$$

The desired reference in this case is

$$y_d(k) = 0.8 + 0.05(\,sin\,(\pi k/50) + \,sin\,(\pi k/100) + \,sin\,(\,sin\,(\pi k/150)))$$

The tracking performance of adaptive GRNN is shown in **Figure 15**.

7. MATLAB examples

In this section, GRNN MATLAB code examples are provided.

7.1. Basic GRNN Commands in MATLAB

In this example, GRNN is trained to find the square of a given number.

To design a GRNN in MATLAB:

Firstly, create the inputs and the targets and specify the spread parameter:

```
clc
clear all
x=[1 3 5 7 9 11];
y=[1 9 25 49 81 121];
spread=0.1;
```

Secondly, create GRNN:

```
net1=newgrnn(x,y,spread);
```

To view GRNN after creating it:

```
view(net1)
```

The results are shown in **Figure 16**.

To find GRNN output based on a given input:

```
y2=net1(4);
```

The result is 17.

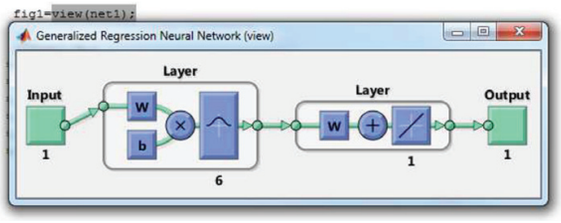

Figure 16. View GRNN in MATLAB.

7.2. The holdout method to find σ

```
clc
clear all
close all
x=[1 3 5 7 9 11];
y=[1 9 25 49 81 121];
data=[x;y];
%keep the original dataset
data_or=data;

% %spread vecotr . it can be more than zero and less or equal one
sp=0.0001:0.01:1;
emin=100;
tic
for j=1:length(sp)

for i=1:10 %10 folds

%s e l e c t a random data sample from training data set
randxi=randi(6);

randx(:,i)=data(:,randxi);
%remove the selected training sample from the training data set
data(:,randxi)=[];
%train GRNN with the reduced training dataset
net1=newgrnn(data(1,:),data(2,:),sp(j));
%find GRNN output of the removed sample
y1(i)=net1(randx(1,i));
% find the meaan suqre error between GRNN output and the removed
sample
% target
me(i)=mse(y1(i),randx(2,i));
%rest the training data as it was before remving the training sample
data=data_or;
%
end
% find the summation of the errors for the current sigma
m_sum(j)=sum(me);
end

%find sigma which have the least errors summation
[idx,ii]=min(m_sum);
best_sigma=sp(ii);
time_holdout=toc;
disp('best sigma is..');
disp(best_sigma);
```

Author details

Ahmad Jobran Al-Mahasneh[1], Sreenatha Anavatti[1]*, Matthew Garratt[1] and Mahardhika Pratama[2]

*Address all correspondence to: s.anavatti@adfa.edu.au

1 School of Engineering and Information Technology, The University of New South Wales Canberra, ACT, Australia

2 School of Computer Science and Engineering, Nanyang Technological University, Singapore

References

[1] Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators. Neural Networks. 1989;**2**(5):359-366

[2] Hagan MT, Menhaj MB. Training feedforward networks with the Marquardt algorithm. IEEE Transactions on Neural Networks. 1994;**5**(6):989-993

[3] Hecht-Nielsen R. Theory of the backpropagation neural network. In: Neural Networks for Perception. Netherlands: Elsevier; 1992. pp. 65-93

[4] Specht DF. A general regression neural network. IEEE Transactions on Neural Networks. 1991;**2**(6):568-576

[5] Schaffner C, Schroder D. An application of general regression neural network to nonlinear adaptive control. In: Fifth European Conference on Power Electronics and Applications, IET; 1993. pp. 219-224

[6] Ahmed O, Mitchell JW, Klein SA. Application of general regression neural network in hvac process identification and control, Technical report. Atlanta, GA (United States): American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc; 1996

[7] Kulkarni SG, Chaudhary AK, Nandi S, Tambe SS, Kulkarni BD. Modeling and monitoring of batch processes using principal component analysis assisted generalized regression neural networks. Biochemical Engineering Journal. 2004;**18**(3):193-210

[8] Ben-Nakhi AE, Mahmoud MA. Cooling load prediction for buildings using general regression neural networks. Energy Conversion and Management. 2004;**45**(13–14):2127-2141

[9] Lee WY, House JM, Kyong NH. Subsystem level fault diagnosis of a building's air-handling unit using general regression neural networks. Applied Energy. 2004;**77**(2): 153-170

[10] Sahroni A. Design of intelligent control system based on general regression neural network algorithm. GSTF Journal on Computing (JoC). 2018;**2**(4):103-110

[11] Hong CM, Cheng FS, Chen CH. Optimal control for variable-speed wind generation systems using general regression neural network. International Journal of Electrical Power and Energy Systems. 2014;**60**:14-23

[12] Li HZ, Guo S, Li CJ, Sun JQ. A hybrid annual power load forecasting model based on generalized regression neural network with fruit fly optimization algorithm. Knowledge-Based Systems. 2013;**37**:378-387

[13] Wei W, Shaoyi B, Lanchun Z, Kai Z, Yongzhi W, Weixing H. Vehicle sideslip angle estimation based on general regression neural network. Mathematical Problems in Engineering. 2016;**2016**:1-7

[14] Huang K, Liu Z, Huang D. Fault diagnosis for methane sensors using generalized regression neural network. International Journal of Online Engineering (iJOE). 2016;**12**(03):42-47

[15] He XY, He SH. Fault detection of excavators hydraulic system using dynamic general regression neural network. Applied Mechanics and Materials, Trans Tech Publications. 2011;**48**:511-514

[16] Pietrowski W. Detection of time-varying inter-turn short-circuit in a squirrel cage induction machine by means of generalized regression neural network. COMPEL-The International Journal for Computation and Mathematics in Electrical and Electronic Engineering. 2017;**36**(1):289-297

[17] Stenhouse T. Integration of helicopter autonomy payload for non-linear system identication of rotorcraft heave mode. The UNSW Canberra at ADFA Journal of Undergraduate Engineering Research. 2018;**9**(2):1-15

[18] Chen TC, Yu CH. Generalized regression neural-network-based modeling approach for traveling-wave ultrasonic motors. Electric Power Components and Systems. 2009;**37**(6): 645-657

[19] Husain H, Khalid M, Yusof R. Automatic clustering of generalized regression neural network by similarity index based fuzzy c-means clustering. In: 2004 IEEE Region 10 Conference, TENCON 2004, IEEE; 2004. pp. 302-305

[20] Tomandl D, Schober A. A modified general regression neural network with new, efficient training algorithms as a robust black box-tool for data analysis. Neural Networks. 2001; **14**(8):1023-1034

[21] Goulermas JY, Liatsis P, Zeng XJ, Cook P. Density-driven generalized regression neural networks for function approximation. IEEE Transactions on Neural Networks. 2007;**18**(6): 1683-1696

[22] Rutkowski L. Generalized regression neural networks in time-varying environment. IEEE Transactions on Neural Networks. 2004;**15**(3):576-596

[23] Seng T, Khalid M, Yusof R. Adaptive general regression neural network for modelling of dynamic plants. Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering. 1999;**213**(4):275-287

[24] Al-Mahasneh AJ, Anavatti SG, Garratt MA. Altitude identification and intelligent control of a flapping wing micro aerial vehicle using modified generalized regression neural networks. In: 2017 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE; 2017. pp. 2302-2307

[25] Mathworks 2018. Neural Network Toolbox Sample Data Sets for Shallow Networks. [Accessed: June 11, 2018]

[26] Liu J. Radial basis function neural network control based on gradient descent algorithm. In: Radial Basis Function Neural Network Control for Mechanical Systems. Germany: Springer; 2013. pp. 55-69

[27] Adetona O, Garcia E, Keel LH. A new method for the control of discrete nonlinear dynamic systems using neural networks. IEEE Transactions on Neural Networks. 2000; **11**(1):102-112