# Runge–Kutta Neural Network for Identification of Dynamical Systems in High Accuracy

Yi-Jen Wang and Chin-Teng Lin, *Member, IEEE*

*Abstract*—**This paper proposes the Runge–Kutta neural networks (RKNN's) for identification of unknown dynamical systems described by ordinary differential equations (i.e., ordinary differential equation or ODE systems) in high accuracy. These networks are constructed according to the Runge–Kutta approximation method. The main attraction of the RKNN's is that they precisely estimate the changing rates of system states (i.e., the right-hand side of the ODE $\dot{\mathbf{x}} = f(\mathbf{x})$) directly in their subnetworks based on the space-domain interpolation within one sampling interval such that they can do long-term prediction of system state trajectories. We show theoretically the superior generalization and long-term prediction capability of the RKNN's over the normal neural networks. Two types of learning algorithms are investigated for the RKNN's, gradient- and nonlinear recursive least-squares-based algorithms. Convergence analysis of the learning algorithms is done theoretically. Computer simulations demonstrate the proved properties of the RKNN's.**

*Index Terms*—**Contraction mapping, gradient descent, nonlinear recursive least square, radial-basis function, Runge–Kutta method, Vander Pol's equation.**

## I. INTRODUCTION

NEURAL networks have been used widely for identification of dynamical systems [1], [2]; including feedforward networks [3], [4] and recurrent networks [5]–[9]. For both kinds, the identified systems are usually considered as discrete-time or first-order discretized systems described by ordinary difference equations. There are several problems in using such kinds of models for long-term prediction of the states of unknown dynamical systems which can be described by ordinary differential equations (ODE's). First, the accuracy of long-term prediction is usually not good, especially for parallel-model prediction where the past predicted values instead of real system outputs are referred by the networks for future prediction. This is majorly because the network learns the system states, instead of the changing rates of system states (i.e., the right-hand side of ODE), through the input–output training pairs, and thus cannot catch the long-term behavior of the identified systems well. Second, larger approximation errors are introduced in first-order discretizing the continuous relationships of practical systems. Third, the proper order of the neural identifier for identifying an ODE system is not easy to know. Furthermore, the existing neural identifier can only predict the system behavior well at fixed time interval (using fixed regular sampling rate). This is not the nature of an ODE system. Although a high-order discretization is more accurate than the first-order discretization, the resulting ordinary difference equations of the former are usually complex and intractable.

In this paper, we present a class of feedforward neural networks called Runge–Kutta neural networks (RKNN's) for precisely modeling an ODE system in the form of $\dot{\mathbf{x}} = f(\mathbf{x})$ with an unknown $f$, where the state vector $\mathbf{x}$ is assumed to be measured noise-free. The neural approximation of $f$ is used in the well-known Runge–Kutta integration formulas [10] to obtain an approximation of $\mathbf{x}$. With the designed network structure and proposed learning schemes, the RKNN's perform high-order discretization of unknown ODE systems *implicitly* (i.e., internally in the network) without the aforementioned complexity and intractability problems. The main attraction of the RKNN's is that they can precisely estimate the changing rates of system states (i.e., the right-hand side of ODE) directly in their subnetworks based on the space-domain interpolation within one sampling interval such that they can do long-term prediction of system state trajectories and are good at parallel-model prediction. Also, since the RKNN models the right-hand side of ODE in its subnetworks directly, some known continuous relationships (physical laws) of the identified system can be incorporated into the RKNN directly to speed up its learning. Such kind of *a priori* knowledge is not easy to be used directly in normal neural identifiers. Another important feature of the RKNN is that it can predict the system behavior at any time instant, not limited by fixed time step (fixed sampling time) as the case in normal neural modeling.

An $n$-order RKNN consists of $n$ identical subnetworks (each with identical network structure and weights) connected in the way realizing an $n$-order Runge–Kutta algorithm. The subnetwork is a normal neural network such as multilayer perceptron network or radial basis function network. Each subnetwork models the right-hand side of ODE directly, and thus the RKNN can approximate an ODE system in high-order accuracy. We verify theoretically the superior generalization and long-term prediction ability of the RKNN's over the normal neural networks by providing some quantitative measures of the errors involved in the RKNN modeling. Associated with the RKNN's are two classes of learning algorithms derived by the gradient-descent method and recursive least-square (RLS) method, respectively. Especially, a class of RLS algorithms, called nonlinear recursive least-square (NRLS) learning algorithms, are derived to increase the learning rate and prediction accuracy of the RKNN's. The NRLS generalizes the original RLS to nonlinear cases such that it can tune the parameters

Manuscript received August 21, 1996; revised April 29, 1997 and November 27, 1997. This work was supported by the R.O.C. National Science Council under Grant NSC87-2213-E-009-136.

The authors are with the Department of Electrical and Control Engineering, National Chiao-Tung University, Hsinchu, Taiwan, R.O.C.

Publisher Item Identifier S 1045-9227(98)01812-8.

in the hidden layers of the RKNN's (such as the centers and variances of the radial basis function networks). Several kinds of NRLS algorithms with different orders of prediction accuracy are developed. The convergence property of the NRLS algorithms applied to the RKNN's is also studied theoretically.

The rest of this paper is organized as follows. Section II presents the structure of the proposed RKNN. In this section, we also analyze the approximation and prediction accuracy of the RKNN quantitatively in the training and generalization phases, respectively. Section III derives the gradient learning algorithms and the nonlinear recursive least-square learning algorithms with different orders of prediction accuracy for the RKNN's. The convergence properties of these algorithms are also analyzed theoretically in this section. Finally, the long-term prediction capability of the proposed RKNN's is demonstrated on two application examples through extensive computer simulations.

## II. Runge–Kutta Neural Network (RKNN)

### A. Problem Statement

Consider a nonlinear system described by the following ODE:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t)) \tag{1}$$

with initial condition

$$\mathbf{x}(0) = \mathbf{x}_0$$

where the state vector $\mathbf{x}(t) \in \Re^m$ and time $t \in [0, T]$. Our objective is to develop a neural network that can model an ODE system precisely whose right-hand-side function $f$ is unknown such that it can do long-term prediction of the state trajectory $\mathbf{x}(t)$ of the system described in (1). More notably, the proposed network is expected to be a *parallel-model* predictor; i.e., it uses only the initial system state, $\mathbf{x}(0)$, to yield long-term output, $\mathbf{y}(t)$, which is highly accurate prediction of the state $\mathbf{x}(t)$ over $t \in [0, T]$ by feeding the past outputs of the identifier back to itself recursively.

For further development, the following assumptions are made.

1) State trajectory $\mathbf{x}(t)$ belongs to a bounded open domain of $\Re^m$ denoted by $D$; i.e., $\mathbf{x}(t) \in D, t \in [0, T]$.
2) The function $f(\mathbf{x})$ is a continuous function and satisfies the Lipschitz condition [11]; i.e., there is a constant $k$ such that $\|f(\mathbf{x}) - f(\mathbf{y})\| \leq k\|\mathbf{x} - \mathbf{y}\|$ for any $\mathbf{x}, \mathbf{y} \in D$, where $\|\cdot\|$ is any norm in $\Re^m$. Except special indication, the symbol $\|\cdot\|$ represents 2-norm in this paper.
3) In the training phase, we can obtain some training state trajectories $\mathbf{x}(t; \mathbf{x}_0)$, i.e., solution of (1), with different initial states $\mathbf{x}_0 \in D$, where we assume the state vector $\mathbf{x}(t)$ is measurable with no-noise at each time step $h$.
4) The continuous system $f(\cdot)$ in (1) is time-invariant.

To predict the state trajectory of the unknown system described by (1), a conventional and popular neural-network-based approach [1], [3] is to construct a neural network, say $\hat{N}_f(\cdot)$, that directly learns the system state trajectory, i.e., $\mathbf{x}(ih; \mathbf{x}_0) \cong \hat{N}_f(\mathbf{x}((i-1)h); \mathbf{x}_0)$. We denote such a network as a direct-mapping neural network (DMNN). This could be interpreted as a first-order discretization of the ODE system in (1). Using the DMNN to do multistep prediction (i.e., long-term prediction) of the system state trajectories has some drawbacks. First, it cannot directly obtain the function behavior of $f(\cdot)$. Second, it is usually not easy to obtain high accuracy for the multistep prediction of state trajectories. Third, it cannot perform the prediction with variable step size; i.e., the time step $h$ must be a constant value used in the training phase. The last point may cause inconvenience for using the neural identifier in practical applications sometimes.

In the next section, we shall propose a class of neural networks, called RKNN's, to attack the drawbacks of the DMNN's. We shall also develop learning algorithms for the RKNN such that the function $f(\cdot)$ in (1) can be directly approximated by the neural network, $N_f(\cdot)$, and then the state trajectory $\mathbf{x}(t)$ can be predicted by the solution of $\dot{\mathbf{y}}(t) = N_f(\mathbf{y}(t))$ with the same initial point, $\mathbf{y}(0) = \mathbf{x}_0$, where $\mathbf{y}(t) \in \Re^m$.

### B. Structure of the RKNN

Before propose the RKNN, we first see a universal approximation property of neural networks; for any system described by (1) satisfying Assumptions 1)–4), there exists a neural network $N_f(\cdot)$ such that the identification model described by $\dot{\mathbf{y}}(t) = N_f(\mathbf{y}(t))$ can do long-term prediction of the state trajectory $\mathbf{x}(t)$ to any degree of accuracy. By the universal approximation theory of neural networks [12], [13] and the theory of ODE's [14], [15], or according to the result in [16], we have the following lemma.

*Lemma 1:* Given any solution trajectory $\mathbf{x}(t; \mathbf{x}_0)$ of the system described in (1) with $\mathbf{x}_0 \in D, t \in [0, T]$, and a function $f(\cdot)$ satisfying Assumption 2, for any $\varepsilon > 0$ there exits a neural network $N_f(\cdot)$ such that the trajectory $\mathbf{y}(t; \mathbf{x}_0)$ corresponding to the system $\dot{\mathbf{y}} = N_f(\mathbf{y}(t)), \mathbf{y}(0) = \mathbf{x}_0$, satisfies $\|\mathbf{y}(t; \mathbf{x}_0) - \mathbf{x}(t; \mathbf{x}_0)\| < \varepsilon$, for all $0 \leq t \leq T$.

The above Lemma shows the existence of a neural network that meets the required property, but it dose not indicate how such a network can be obtained. We shall then try to construct a neural network, RKNN, that fits Lemma 1. The RKNN is motivated by the Runge–Kutta algorithm [10]. We construct an $n$-order RKNN to realize the computation flow of an $n$-order Runge–Kutta algorithm. For example, the structure of a fourth-order RKNN is shown in Fig. 1, where the neural network $N_f(\cdot)$ plays the role of the unknown function, $f(\cdot)$, in Lemma 1. The input–output relationship of the fourth-order RKNN is described by

$$\mathbf{y}(i+1) = \mathbf{y}(i) + \frac{1}{6}h(\mathbf{k}_0 + 2\mathbf{k}_1 + 2\mathbf{k}_2 + \mathbf{k}_3) \tag{2}$$

where

$$\begin{aligned}
\mathbf{k}_0 &= N_f(\mathbf{y}(i); \omega) \\
\mathbf{k}_1 &= N_f\left(\mathbf{y}(i) + \frac{1}{2}h\mathbf{k}_0; \omega\right) \\
\mathbf{k}_2 &= N_f\left(\mathbf{y}(i) + \frac{1}{2}h\mathbf{k}_1; \omega\right) \\
\mathbf{k}_3 &= N_f(\mathbf{y}(i) + h\mathbf{k}_2; \omega)
\end{aligned}$$

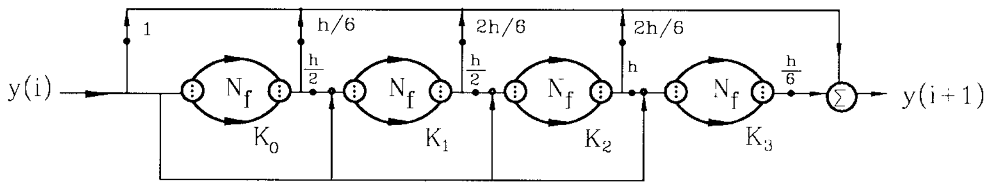

Fig. 1. Structure of the fourth-order RKNN constructed by the $N_f(\cdot)$ neural networks.

where the neural network $N_f(\mathbf{x};\omega)$, with input $\mathbf{x}$ and weights $\omega$, can be the multilayer perceptron network or the radial basis function network. It is noted that the four $N_f(\mathbf{x};\omega)$'s in Fig. 1 are identical, meaning that they have the same structure and use the same corresponding weights. Since the $n$ subnetworks of an $n$-order RKNN are identical, only one subnetwork needs to be realized in either software or hardware implementation. Hence, the real network size of an $n$-order RKNN is the same as that of its constituent subnetwork.

With the supervised learning algorithms developed in the next section, we can tune the weights, $\omega$, of $N_f(\mathbf{x};\omega)$ by training the corresponding RKNN on the training trajectories obtained from the identified system described in (1). When the total prediction error is sufficiently small or the weight vector $\omega$ converges to $\omega^*$, we can obtain a RKNN, $N_f(\mathbf{x};\omega^*)$, which approximates the continuous function $f(\mathbf{x})$ of the system in (1) accurately. It can be expected that the RKNN's have higher prediction accuracy than the DMNN's according to the property of Runge–Kutta algorithm [17], [18]. In the followings, we shall show the degree of accuracy theoretically.

We now show quantitatively that the RKNN has higher prediction accuracy and better generalization capability than the conventional DMNN that uses a single $N_f$ in Fig. 1. In the following analysis, we assume that $f(\mathbf{x})$ is a $n$-times continuously differentiable function (i.e., $f(\mathbf{x}) \in C^n(D)$). To simplify the accuracy analysis of the RKNN in the training phase and generalization phase, we shall focus on the second-order RKNN and consider one-dimensional input $x(t)$. The following notation will be used frequently in the analysis:

$$g(h) = O(h^p) \qquad \text{as } h \longrightarrow 0$$

which means that there exists a positive constant $K$ such that $\|g(\mathbf{x})\| \le Kh^p$ as $h$ sufficiently close to zero.

*Lemma 2:* Consider one-step prediction of the system $\dot{x}(t) = f(x(t))$ at $x(i) \equiv x(ih)$ using a second-order RKNN

$$y(i+1) = x(i) + \frac{h}{2}[N_f(x(i)) + N_f(x(i) + hN_f(x(i)))].$$

Suppose $\|y(i+1) - x(i+1)\| \approx O(h^3)$ for all $x(i) \in D$. Then $N_f$ can approximate $f$ to the following accuracy:

$$\left\| f\left(x(i) + \frac{1}{2}hf(x(i))\right) - N_f\left(x(i) + \frac{1}{2}hN_f(x(i))\right)\right\| = O(h^2).$$

*Proof:* The output of the system in (1) with initial condition $x(i)$ is

$$\begin{aligned} x(i+1) &= x(i) + \dot{x}|_{x(i)} \cdot h + \frac{1}{2}\ddot{x}\Big|_{x(i)} \cdot h^2 + O(h^3) \\ &= x(i) + f(x(i)) \cdot h + \frac{h^2}{2}\frac{\partial f}{\partial x}\Big|_{x(i)} \cdot \dot{\mathbf{x}}(i) + O(h^3) \\ &= x(i) + f(x(i)) \cdot h + \frac{h^2}{2}\frac{\partial f}{\partial x}\Big|_{x(i)} \cdot f(x(i)) + O(h^3). \end{aligned} \tag{3}$$

The output of the second-order RKNN with input $x(i)$ is

$$\begin{aligned} y(i+1) &= x(i) + \frac{h}{2}[N_f(x(i)) + N_f(x(i) + hN_f(x(i)))] \\ &= x(i) + N_f(x(i))h + \frac{h^2}{2}\frac{\partial N_f}{\partial x}\Big|_{x(i)} N_f(x(i)) + O(h^3). \end{aligned} \tag{4}$$

The magnitude of the difference of the above two output values is

$$\begin{aligned} \|x(i+1) - y(i+1)\| &= h\left\| f(x(i)) - N_f(x(i)) + \frac{h}{2}\frac{\partial f}{\partial x}\Big|_{x(i)} \cdot f(x(i)) \right. \\ &\quad \left. - \frac{h}{2}\frac{\partial N_f}{\partial x}\Big|_{x(i)} N_f(x(i))\right\| + O(h^3). \end{aligned} \tag{5}$$

From the assumption that $\|y(i+1) - x(i+1)\| \approx O(h^3)$ for all $x(i) \in D$, we have

$$\left\| f(x(i)) + \frac{h}{2}\frac{\partial f}{\partial x} f(x(i)) - \left[N_f(x(i)) + \frac{h}{2}\frac{\partial N_f}{\partial x} N_f(x(i))\right]\right\| = O(h^2)$$

and thus

$$\left\| f\left(x(i) + \frac{h}{2}f(x(i))\right) - N_f\left(x(i) + \frac{h}{2}N_f(x(i))\right)\right\| = O(h^2).$$

This completes our proof. □

*Remark 1:* For the DMNN $y(i+1) = \hat{N}_f(x(i))$, suppose we obtain the network output accuracy to the third order (i.e., $\|y(i+1) - x(i+1)\| = O(h^3)$). This accuracy order is the same as that of $N_f(\cdot)$ in the RKNN assumed in Lemma 2. Since we usually estimate the continuous function $f(x(i))$ simply by a causal one-step prediction method, $\hat{f}(x(i)) = \frac{1}{h}(\hat{N}_f(x(i)) - x(i))$, we can only achieve the first-order approximation accuracy in predicting the continuous function $f(x(i))$ using the DMNN, i.e., $\|f(x(i)) - \hat{f}(x(i))\| = O(h)$.

*Remark 2:* Under the same assumptions made in Lemma 2, if $\frac{\partial f}{\partial x}$ and $\frac{\partial N_f}{\partial x}$ do not grow rapidly and $h$ is sufficiently small, we can have $\|f(x(i)) - N_f(x(i))\| = O(h^2)$. Hence the one-step prediction accuracy of the second-order RKNN is better than that of the DMNN.

*Lemma 3:* Suppose the DMNN and second-order RKNN reach perfect approximation in the training phase, i.e., $\hat{N}_f(x(i)) = x(i+1)$ and $N_f(x(i)) = f(x(i))$ for all $i$, where $(x(i), x(i+1)) \in D \times D$ are input–output training pairs obtained from the system described in (1). Then in the generalization phase, if we apply $y(i) = x(i) + \triangle$ to the inputs of the DMNN and second-order RKNN, the output prediction errors of these two networks to the first-order in $\triangle$ and $h$ are, respectively,

$$\varepsilon_{\text{DMNN}} = \left[1 + \frac{\partial f}{\partial x}h - \frac{\partial \hat{N}_f}{\partial x}\right] \cdot \triangle$$

and

$$\varepsilon_{\text{RKNN}} = \left[\frac{\partial f}{\partial x} - \frac{\partial N_f}{\partial x}\right] h \cdot \triangle.$$

*Proof:* The solution of the system described in (1) can be written as

$$x(i+1) = x(i) + f(x(i))h + \frac{1}{2}\frac{\partial f}{\partial x}\bigg|_{x(i)} \cdot f(x(i))h^2 + O(h^3). \tag{6}$$

In generalization phase, we have

$$\begin{aligned} y(i+1) &= x(i) + \triangle + f(y(i)) \cdot h \\ &\quad + \frac{1}{2}\frac{\partial f}{\partial x}\bigg|_{y(i)} \cdot f(y(i)) \cdot h^2 + O(h^3) \\ &= x(i) + \triangle + f(x(i)) \cdot h + \frac{\partial f}{\partial x}\bigg|_{x(i)} \cdot \triangle \cdot h \\ &\quad + \frac{1}{2}\frac{\partial f}{\partial x}\bigg|_{y(i)}\left(f(x(i)) + \frac{\partial f}{\partial x}\bigg|_{y(i)} \cdot \triangle\right)h^2 + O(h^3) \\ &= x(i) + \triangle + f(x(i)) \cdot h + \frac{\partial f}{\partial x}\bigg|_{x(i)} \cdot \triangle \cdot h \\ &\quad + \frac{1}{2}\frac{\partial f}{\partial x}\bigg|_{x(i)} f(x(i))h^2 + O(\triangle^n h^m) \\ &= x(i+1) + \triangle + \frac{\partial f}{\partial x}\bigg|_{x(i)} \cdot \triangle \cdot h + O(\triangle^n h^m) \end{aligned}$$

for $n \geq 1,\ m \geq 2$. The last equality comes from (6). By the assumption that $\hat{N}_f(x(i)) = x(i+1)$, the DMNN's output with input $x(i) + \triangle$ is

$$\begin{aligned} \hat{y}(i+1) &= \hat{N}_f(x(i) + \triangle) \\ &= \hat{N}_f(x(i)) + \frac{\partial \hat{N}_f}{\partial x}\bigg|_{x(i)} \cdot \triangle + O(\triangle^2). \end{aligned}$$

Hence the DMNN's prediction error to the first order in $\triangle$ and $h$ is

$$\begin{aligned} \varepsilon_{\text{DMNN}} &= y(i+1) - \hat{y}(i+1) \\ &= \left[1 + \frac{\partial f}{\partial x}\bigg|_{x(i)} h - \frac{\partial \hat{N}_f}{\partial x}\bigg|_{x(i)}\right]\triangle. \end{aligned}$$

By the assumption that $N_f(x(i)) = f(x(i))$, the output of the second-order RKNN is

$$\begin{aligned} \hat{y}(i+1) &= y(i) + \frac{h}{2}[N_f(y(i)) + N_f(y(i) + h \cdot N_f(y(i)))] \\ &= y(i) + h \cdot \left[\frac{1}{2}N_f(y(i)) + \frac{1}{2}N_f(y(i))\right. \\ &\quad \left. + \frac{1}{2}\frac{\partial N_f}{\partial x}\bigg|_{y(i)} \cdot N_f(y(i)) \cdot h + O(h^2)\right] \\ &= y(i) + h \cdot \left[N_f(y(i)) + \frac{1}{2}\frac{\partial N_f}{\partial x}\bigg|_{y(i)} \cdot N_f(y(i)) \cdot h\right] \\ &\quad + O(h^3) \\ &= y(i) + h \cdot [N_f(x(i) + \triangle)] + \frac{1}{2}h^2\frac{\partial N_f}{\partial x}\bigg|_{y(i)} \\ &\quad \cdot N_f(y(i)) + O(h^3) \\ &= x(i) + \triangle + h \cdot \left[f(x(i)) + \frac{\partial N_f}{\partial x}\bigg|_{x(i)} \cdot \triangle\right] \\ &\quad + \frac{1}{2}h^2 \cdot \frac{\partial f}{\partial x}\bigg|_{x(i)} f(x(i)) + O(\triangle^n h^m). \end{aligned}$$

Hence the second-order RKNN's prediction error to the first order in $\triangle$ and $h$ is

$$\begin{aligned} \varepsilon_{\text{RKNN}} &= y(i+1) - \hat{y}(t+1) \\ &= \left(\frac{\partial f}{\partial x}\bigg|_{x(i)} - \frac{\partial N_f}{\partial x}\bigg|_{x(i)}\right) \cdot \triangle \cdot h. \end{aligned} \tag{7}$$

This completes our proof. □

*Remark 3:* From Lemma 3 and in the situation that $\frac{\partial N_f}{\partial x}$ and $\frac{\partial \hat{N}_f}{\partial x}$ fit $\frac{\partial f}{\partial x}$ to the same order of accuracy, it is clear that the second-order RKNN has better generalization capability than the DMNN and we can expect that this is even true for the higher order RKNN's.

## III. Learning Algorithms for the RKNN

In this section, we shall develop learning algorithms for the RKNN. Consider an initial state $\mathbf{x}_0 \in D$ and a trajectory $\mathbf{x}(t; \mathbf{x}_0)$ which is the solution of system $\dot{\mathbf{x}}(t) = f(\mathbf{x}(t))$ corresponding to the initial state $\mathbf{x}_0$. At each time step $h$, we get the sampling data, $\mathbf{x}(i; \mathbf{x}_0) \equiv \mathbf{x}(ih; \mathbf{x}_0), i = 0, \cdots, \lfloor \frac{T}{h} \rfloor = L - 1$, where $h$ is sufficiently small. We collect $\mathbf{x}(i; \mathbf{x}_0)$ for several different initial states $\mathbf{x}_0 \in D$ (i.e., several different trajectories) as training data of the RKNN. By the learning algorithms developed in this section, the weights $\omega$ of $N_f(\mathbf{x};\omega)$ in the RKNN are tuned such that the outputs $\mathbf{y}(t;\mathbf{x}_0)$ of the identified system $\dot{\mathbf{y}}(t) = N_f(\mathbf{y}(t);\omega), \mathbf{y}(0) = \mathbf{x}_0$, can approximate the solution $\mathbf{x}(t;\mathbf{x}_0)$ of $\dot{\mathbf{x}}(t) = f(\mathbf{x}(t)), \mathbf{x}(0) = \mathbf{x}_0$, for all $t$ in some fixed interval $[0,T]$.

### *A. Gradient Learning Algorithms*

The generalized gradient descent (backpropagation) rule can be used to derive learning algorithms for the RKNN with each subnetwork $N_f$ being the multilayer perceptron, radial basis function network or other proper neural networks. In this section, we shall develop gradient learning algorithms for the fourth-order RKNN. The output of the fourth-order RKNN with input $\mathbf{y}(i)$ is described in (2). The derivation of gradient learning algorithms for other order RKNN's can be done in a similar way.

The derived gradient learning algorithm can propagate the error gradients backward in the RKNN to minimize the squared error, $E = \|\mathbf{x}(i+1) - \mathbf{y}(i+1)\|^2$, at time step $i$. A recursive update rule is expressed as follows. We consider $N_f(\mathbf{y}(i);\omega)$ as a function of $\mathbf{y}(i)$ and $\omega$, and denote $\Psi_{\mathbf{y}}(\cdot) = \frac{\partial N_f(\mathbf{y}(i);\omega)}{\partial \mathbf{y}(i)}$ and $\Psi_\omega(\cdot) = \frac{\partial N_f(\mathbf{y}(i);\omega)}{\partial \omega}$, where $\mathbf{y}(i)$ is the input state vector at time step $i$ and $\omega_j$ is the $j$th component of $\omega$. Differentiating (2) with respect to the weight $\omega_j$, we have

$$\frac{\partial \mathbf{y}(i+1)}{\partial \omega_j} = \frac{1}{6}h\left(\frac{\partial \mathbf{k}_0}{\partial \omega_j} + 2\frac{\partial \mathbf{k}_1}{\partial \omega_j} + 2\frac{\partial \mathbf{k}_2}{\partial \omega_j} + \frac{\partial \mathbf{k}_3}{\partial \omega_j}\right) \tag{8}$$

where

$$\begin{aligned}
\frac{\partial \mathbf{k}_0}{\partial \omega_j} &= \Psi_{\mathbf{y}}(\mathbf{y}(i);\omega)\frac{\partial \omega}{\partial \omega_j}\\
\frac{\partial \mathbf{k}_1}{\partial \omega_j} &= \Psi_{\mathbf{y}}\left(\mathbf{y}(i) + \frac{1}{2}h\mathbf{k}_0;\omega\right)\frac{\partial \mathbf{k}_0}{\partial \omega_j}\cdot\frac{1}{2}h\\
&\quad + \Psi_\omega\left(\mathbf{y}(i) + \frac{1}{2}h\mathbf{k}_0;\omega\right)\frac{\partial \omega}{\partial \omega_j}\\
\frac{\partial \mathbf{k}_2}{\partial \omega_j} &= \Psi_{\mathbf{y}}\left(\mathbf{y}(i) + \frac{1}{2}h\mathbf{k}_1;\omega\right)\frac{\partial \mathbf{k}_1}{\partial \omega_j}\cdot\frac{1}{2}h\\
&\quad + \Psi_\omega\left(\mathbf{y}(i) + \frac{1}{2}h\mathbf{k}_1;\omega\right)\frac{\partial \omega}{\partial \omega_j}\\
\frac{\partial \mathbf{k}_3}{\partial \omega_j} &= \Psi_{\mathbf{y}}(\mathbf{y}(i) + h\mathbf{k}_2;\omega)\frac{\partial \mathbf{k}_2}{\partial \omega_j}\cdot h\\
&\quad + \Psi_\omega\left(\mathbf{y}(i) + \frac{1}{2}h\mathbf{k}_2;\omega\right)\frac{\partial \omega}{\partial \omega_j}.
\end{aligned}$$

Hence

$$\begin{aligned}
\frac{\partial E}{\partial \omega_j} &= -2(\mathbf{x}(i+1) - \mathbf{y}(i+1))\cdot\frac{\partial \mathbf{y}(i+1)}{\partial \omega_j}\\
&= -2(\mathbf{x}(i+1) - \mathbf{y}(i+1))\cdot\frac{1}{6}h\left(\frac{\partial \mathbf{k}_0}{\partial \omega_j} + 2\frac{\partial \mathbf{k}_1}{\partial \omega_j}\right.\\
&\quad\left. + 2\frac{\partial \mathbf{k}_2}{\partial \omega_j} + \frac{\partial \mathbf{k}_3}{\partial \omega_j}\right).
\end{aligned} \tag{9}$$

We shall next derive $\Psi_{\mathbf{y}}$ and $\Psi_\omega$ in the case that $N_f(\mathbf{y};\omega)$ is a radial basis function network (called the *radial-basis-type* RKNN). With these derivatives, we can apply the formulas, (8) and (9), to find the error gradient directions in the weight space to update the weights. The same procedure can also be used to derive the gradient learning algorithm for the RKNN with the subnetwork $N_f(\mathbf{y};\omega)$ being a multilayer perceptron.

In a radial-basis-type RKNN, the constituent subnetwork is a radial basis function network

$$[N_f(\mathbf{y}(i))]_k = \sum_{l=1}^{N} W_{kl}\phi_l(\mathbf{y}(i);\mathbf{t}_l,\sigma_l), \quad k = 1,\cdots,m \tag{10}$$

where $\mathbf{y}(i) \in \Re^m$ is the input vector, $k$ indicates the $k$th output of $N_f(\cdot)$, $N$ is the total number of radial basis functions in $N_f(\cdot)$, $W_{kl}$ is the $(k,l)$th element of the weight matrix $W$ representing the connection weight from the $l$th radial-basis-function node to the $k$th output node, and $\phi_l(\cdot)$ is the radial basis function defined by

$$\phi_l(\mathbf{y}(i);\mathbf{t}_l,\sigma_l) = \exp\{-[(\mathbf{y}(i) - \mathbf{t}_l)^T\sigma_l(\mathbf{y}(i) - \mathbf{t}_l)]\}$$

where $\mathbf{t}_l \in \Re^m$ is the center of a radial basis function, and $\sigma_l$ is the inverse of variance matrix of the $l$th radial basis function.

Differentiating $N_f$ with respect to weights $\omega = [W_{kl}, \mathbf{t}_l, \sigma_l]$, we obtain $\Psi_\omega(\cdot)$ as follows:

$$\begin{aligned}
\frac{\partial[N_f(\mathbf{y}(i))]_k}{\partial W_{jl}} &= \begin{cases}\phi_l(\cdot) & k=j\\ 0 & k\neq j\end{cases}, && \text{(scaler)}\\
\frac{\partial[N_f(\mathbf{y}(i))]_k}{\partial \mathbf{t}_l} &= 2W_{kl}\phi_l'(\cdot)\sigma_l(\mathbf{y}(i) - \mathbf{t}_l) && \text{(vector)}\\
\frac{\partial[N_f(\mathbf{y}(i))]_k}{\partial \sigma_l} &= -W_{kl}\phi_l'(\cdot)(\mathbf{y}(i) - \mathbf{t}_l)(\mathbf{y}(i) - \mathbf{t}_l)^T. && \text{(matrix)}
\end{aligned} \tag{11}$$

Similarly, differentiating $N_f(\mathbf{y}(i))$ with respect to the state $\mathbf{y}(i)$, we obtain

$$\left.\frac{\partial N_f}{\partial \mathbf{y}(i)}\right|_{\mathbf{y}(i)} = \begin{bmatrix} -2\sum_{j=1}^{N} W_{1j}\phi_j(\cdot)\sigma_j(\mathbf{y}(i) - \mathbf{t}_j)^T\\ \vdots\\ \vdots\\ -2\sum_{j=1}^{N} W_{mj}\phi_j(\cdot)\sigma_j(\mathbf{y}(i) - \mathbf{t}_j)^T \end{bmatrix}. \quad \text{(Matrix)} \tag{12}$$

Substituting (11) and (12) to (8) and (9) recursively, we can obtain the gradient directions of $E$. Hence (8) can be used to update the weights $\omega$ (including $W_{kl}, t_l, \sigma_l$) of the radial basis function network $N_f(\cdot)$ in the RKNN.

### *B. Nonlinear Recursive least-square Learning Algorithms*

Because the gradient update rule in (9) for the RKNN's is not a batch update method and it is difficult to choose a proper learning constant, the total error's convergent rate of the gradient learning algorithms developed in the last section is quite small. In this section, we shall develop faster learning algorithms, the *nonlinear recursive least-square* (NRLS) algorithms, for the RKNN's. Like the nonlinear least-square method discussed in [19], the NRLS algorithms generalize the conventional recursive least-square (RLS) algorithms to nonlinear cases. The property comparison between the gradient algorithms (i.e., the steepest descent method) and RLS algorithms can be found in [20] and [21]. To minimize the square error function $E(\omega)$, the steepest descent method finds and follows the gradient vector to improve the approximation accuracy as iteration number increasing. On the other hand, the RLS method directly solves the roots of equation $\frac{\partial E(\omega)}{\partial \omega} = 0$ to locate the local minimum points directly. It then transforms the equation of $\frac{\partial E(\omega)}{\partial \omega} = 0$ into a regression model defined by $\phi^T(\cdot)\omega = 0$, where $\omega$ represents network weights and $\phi$ is the regressor. Finally we can use a recursive algorithm to solve the equation $\phi^T(\cdot)\omega = 0$.

In the following sections, we shall derive three different NRLS algorithms for the RKNN's for different accuracy levels and scopes of tunable parameters in $N_f(\cdot;\omega)$. Again, we focus on the radial-basis-type RKNN's in the following derivation.

*1) Zero-Order NRLS Learning Algorithm* When proper radial basis functions of $N_f(\cdot)$ in the RKNN have been chosen (i.e., the centers $\mathbf{t}_j$ and inverse variance matrices $\sigma_j(\cdot)$ have been decided and fixed), we only need to tune the weights $W$ on the links connecting the radial-basis-functions nodes $\phi_j(\cdot;\mathbf{t}_l,\delta_l)$ to the output layer of $N_f(\cdot)$. In this section, we shall develop the first NRLS algorithm of the RKNN, called the zero-order NRLS learning algorithm, for this learning task. Assume we are given training trajectories $\{\mathbf{x}(i;\mathbf{x}_0) \mid i = 0,\cdots,L-1, \mathbf{x}_0 \in D\}$ from the system $\dot{\mathbf{x}}(t) = f(\mathbf{x}(t))$ with $\mathbf{x}(0) = \mathbf{x}_0$. From (10), we define

$$E(W) \equiv \frac{1}{2}\sum_{i=1}^{L}(\mathbf{x}(i) - \mathbf{y}(i))^2 \tag{13}$$

where $\mathbf{y}(i)$ is the output of the RKNN. According to (10) and the RKNN structure described in (2), the output $\mathbf{y}(i)$ of the RKNN with input $\mathbf{x}(i-1)$ at time step $i-1$ can be written as

$$\begin{aligned}\mathbf{y}(i) = \mathbf{x}(i-1) + \frac{h}{6}\Bigg\{&\sum_{l=1}^{N} W_l\phi_l(\mathbf{x}(i-1)) \\ &+ 2\sum_{l=1}^{N} W_l\phi_l\left(\mathbf{x}(i-1) + \frac{h}{2}\mathbf{k}_0(i-1)\right) \\ &+ 2\sum_{l=1}^{N} W_l\phi_l\left(\mathbf{x}(i-1) + \frac{h}{2}\mathbf{k}_1(i-1)\right) \\ &+ \sum_{l=1}^{N} W_l\phi_l(\mathbf{x}(i-1) + h\mathbf{k}_2(i-1))\Bigg\}\end{aligned} \tag{14}$$

where $\phi_l$ is the $l$th radial basis function, $W_l$ is the connection weight between the $\phi_l$ node to the output node of $N_f(\cdot)$, and $\mathbf{k}_0, \mathbf{k}_1, \mathbf{k}_2$ are the outputs of the $N_f(\cdot)$ subnetworks in the RKNN defined by (2). In Appendix, we derive the regression form of $\mathbf{y}(i) - \mathbf{x}(i-1)$ in (14) such that we can solve the connection weights $W_l$'s using the nonlinear least-square method in [19].

With the regression form, $\phi(\mathbf{x}(i-1);W)$, of $\mathbf{y}(i) - \mathbf{x}(i-1)$ derived in Appendix and according to the RLS method, we find that minimization of (13) is equivalent to finding the solution of the following equations in the least-square sense:

$$\begin{bmatrix}\phi^T(\mathbf{x}(0);W)\\ \vdots \\ \vdots \\ \phi^T(\mathbf{x}(L-1);W)\end{bmatrix}\begin{bmatrix}W_1\\ \vdots\\ \vdots\\ W_N\end{bmatrix} = \begin{bmatrix}\mathbf{x}(1)-\mathbf{x}(0)\\ \vdots\\ \vdots\\ \mathbf{x}(L)-\mathbf{x}(L-1)\end{bmatrix}. \tag{15}$$

If we let

$$\begin{aligned}&\mathbf{d}^T \equiv [\mathbf{x}(1)-\mathbf{x}(0), \mathbf{x}(2)-\mathbf{x}(1), \cdots, \mathbf{x}(L)-\mathbf{x}(L-1)]\\ &\mathcal{L}(\mathbf{x}(0),\cdots,\mathbf{x}(L-1);W)\\ &\quad \equiv [\phi(\mathbf{x}(0);W),\cdots,\phi(\mathbf{x}(L-1);W)]^T\end{aligned}$$

then (15) can be expressed as $\mathcal{L}(\mathbf{x}(0),\cdots,\mathbf{x}(L-1);W)W = \mathbf{d}$. Notice that the problem of solving (15) is a nonlinear least-square problem, because the regression matrix $\mathcal{L}(\mathbf{x}(0),\cdots,\mathbf{x}(L-1);W)$ is a function of parameter $W$. Analogous to the derivation of the nonlinear least-square method in [19], we combine the fixed point method in [22] and the RLS algorithm to find the solution $W^*$ of (15) in the least-square sense. We denote this method as the zero-order NRLS algorithm for the RKNN's. This algorithm is listed as follows.

Step 1) Choose initial weights $W(0) = W_0$ and set $i = 0$.
Step 2) Substitute $W(i)$ into the regression matrix to get $\mathcal{L}(\mathbf{x}(0),\cdots,\mathbf{x}(L-1);W(i))$.
Step 3) Use RLS algorithm to solve $\mathcal{L}(\mathbf{x}(0),\cdots,\mathbf{x}(L-1);W(i))W = \mathbf{d}$ to get the solution $W = W^*$.
Step 4) Let $W(i+1) = W^*$.
Step 5) If the sequence $W(i)$ converges, then stop; otherwise, set $i = i+1$ and go to Step 2).

The sufficient conditions for the convergence of the nonlinear least-square method are given in [19]. We shall now study the convergence property of the above zero-order NRLS learning algorithm by showing that the radial-basis-type RKNN's own the required sufficient conditions in [19]. First, let us see some definitions and lemma that will be used in our proof [22].

*Definition 1:* Let $T$ be an operator mapping $\Re^m$ into $\Re^m$. Then $\mathbf{x} \in \Re^m$ is called a fixed point of $T$, if $\mathbf{x} = T\mathbf{x}$.

*Definition 2:* Let $B(\mathbf{z},r)$ be a neighborhood of $\mathbf{z} \in \Re^m$ defined by $B(\mathbf{z},r) = \{\mathbf{x} \mid \|\mathbf{x}-\mathbf{z}\| \le r\}$. Then an operator $T : \Re^m \longrightarrow \Re^m$ is said to be a contraction mapping in $B(\mathbf{z},r)$, if there exits a constant $0 \le \theta \le 1$ such that $\|T\mathbf{x}_1 - T\mathbf{x}_2\| \le \theta\|\mathbf{x}_1-\mathbf{x}_2\|$ for all $\mathbf{x}_1,\mathbf{x}_2 \in B(\mathbf{z},r)$, where $\theta$ is a contraction factor.

*Lemma 4:* Let $T$ be an operator from $\Re^m$ to $\Re^m$. Assume $T$ is a contraction mapping with the contraction factor $\theta < 1$ in $B(\mathbf{x}_0,r)$, where $r \ge r_0 = \frac{1}{1-\theta}\|\mathbf{x}_0 - T\mathbf{x}_0\|$. Then the sequence $\mathbf{x}_n = T\mathbf{x}_{n-1}$, $n = 1,2,\cdots$ converges to a fixed point $\mathbf{x}^*$.

The proof of the above lemma can be found in [22]. The fact in Lemma 4 is also given in [22, Th. 6.2.2].

We shall then study and prove the convergence property of the proposed zero-state NRLS learning algorithm applied to the radial-basis-type RKNN's. To simplify the analysis, we consider the second-order RKNN with single state variable, i.e., $\mathbf{x}(t) \in \Re^1$. According to the structure of the second-order RKNN, we have

$$\begin{aligned} &x(i+1) \\ &= x(i) + \frac{h}{2}\{N_f(x(i);W) + N_f(x(i) \\ &\quad + hN_f(x(i);W);W)\} \\ &= x(i) + \frac{h}{2}\left\{\phi^T(x(i))W + \phi^T(x(i))W \right. \\ &\qquad \left. + h\frac{\partial N_f}{\partial x}\bigg|_{(x(i),W)} \phi^T(x(i))W + O(h^2)\right\}. \end{aligned}$$

If we neglect the $O(h^2)$ term and substitute the above equation to (15), then we have

$$h \cdot \begin{bmatrix} \phi^T(x_0) + hM(x_0,W)\phi^T(x_0) \\ \vdots \\ \vdots \\ \phi^T(x_{L-1}) + hM(x_{L-1},W)\phi^T(x_{L-1}) \end{bmatrix} W = \mathbf{y}_d \qquad (16)$$

where $\mathbf{y}_d^T = (x(1)-x(0), x(2)-x(1), \cdots, x(L)-x(L-1))$, and

$$\begin{aligned} M(x(i),W) &= \frac{1}{2}\frac{\partial N_f}{\partial x}\bigg|_{(x(i),W)} \\ &= \sum_{j=1}^{N} W_j \exp\{-\|x(i)-t_j\|^2_{\sigma_j}\} \\ &\quad \cdot \sigma_j \cdot (x(i)-t_j) \end{aligned}$$

where $\|x(i)-t_j\|_{\sigma_j} \equiv \|(x(i)-t_j)^T\sigma_j(x(i)-t_j)\|$.

If we let

$$\mathcal{L}(x(0),\cdots,x(L-1)) \equiv h \cdot \begin{bmatrix} \phi^T(x(0)) \\ \vdots \\ \vdots \\ \phi^T(x(L-1)) \end{bmatrix}$$

be an $L \times N$ matrix and

$$\begin{aligned} &D(W;x(0),\cdots,x(L-1)) \\ &\equiv h \cdot \begin{bmatrix} M(x(0),W) & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & M(x(L-1),W) \end{bmatrix} \end{aligned}$$

be an $L \times L$ matrix, then (16) becomes

$$\begin{aligned} &[I + D(W;x(0),\cdots,x(L-1))] \\ &\times \mathcal{L}(x(0),\cdots,x(L-1))W = \mathbf{y}_d. \end{aligned}$$

Let $\mathbf{x} = (x(0),\cdots,x(L-1))$. Then we have

$$\mathcal{L}(\mathbf{x})W = \begin{bmatrix} \frac{1}{1+hM(x(0),W)} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \frac{1}{1+hM(x(L-1),W)} \end{bmatrix} \mathbf{y}_d.$$

If $\mathcal{L}^T(\mathbf{x})\mathcal{L}(\mathbf{x})$ is nonsingular, the above equation has the least-square-sense solution

$$\begin{aligned} W &= [\mathcal{L}^T(\mathbf{x})\mathcal{L}(\mathbf{x})]^{-1} \\ &\times \begin{bmatrix} \frac{1}{1+hM(x(0),W)} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \frac{1}{1+hM(x(L-1),W)} \end{bmatrix} \mathbf{y}_d. \end{aligned} \qquad (17)$$

Next if we can show $(I + D(W;x(0),\cdots,x(L-1))^{-1}$ is a contraction mapping, then according to Lemma 4, we can find the solution $W^*$ by substituting $W$ to the above equation recursively. To show this, we first take a variation vector, $W^+$, in the neighborhood of $W^0$ and find the following difference:

$$\begin{aligned} &\frac{1}{1+M(x(i),W^+)h} - \frac{1}{1+M(x(i),W^0)h} \\ &= \sum_{j=1}^{N} c_j(x(i))(W_j^+ - W_j^0) \\ &\le [c_1(x(i)),\cdots,c_N(x(i))] \cdot \|W^+ - W^0\| \end{aligned} \qquad (18)$$

where where $\|\triangle_j\| = \|x(i) - t_j\|_{\sigma_j}$.

Then we have the following lemma.

*Lemma 5:* Let $\mathbf{c}(x(i);W^+,W^0) = [c_1(x(i)),\cdots, c_N(x(i))]$, where $c_k(x(i))$ is defined by (19), shown at the bottom of the page, and $x(i)$ are the training data. If $\|\mathbf{c}(x(i))\| < 1$ in the neighborhood of $W^0$ for all $i = 0,\cdots,L-1$, then $\{I + D(x(0),\cdots,x(L-1);W)\}^{-1}$ has a fixed point, $W^*$, in the neighborhood of $W^0$.

*Proof:* Let $\Im(W) \equiv [I + D(x(0),\cdots,x(L-1))]^{-1}$. Consider the following equation:

$$W = \Im(W). \qquad (20)$$

By (18), we know that for any $W^+$ close to $W^0$ and for the $i$th diagonal component of $\Im$, we have $[\Im(W^+)]_i - [\Im(W^0)]_i \le \|\mathbf{c}(x(j))\| \cdot \|W^+ - W^0\|$, where $i \in 0,\cdots,L-1$. If we take the maximum norm of each diagonal element of $\Im$, we have

$$\|\Im(W^+) - \Im(W^0)\|_\infty \le k\|W^+ - W^0\|$$

where $k = \max\{\mathbf{c}(0),\cdots,\mathbf{c}(x(L-1))\}$. Hence by Definition 2, (20) is a contraction mapping and by Lemma 4 or by Theorem 6.2.2 in [19], $\Im$ has a fixed point in the neighborhood of $W^0$. This completes the proof. □

If $\mathcal{L}^T\mathcal{L}$ is nonsingular, then because $\mathbf{c}(x(i))$ is a function of $h$, (17) is a contraction mapping for sufficiently small $h$. Again, by Lemma 4 it is sure that $W$ will converge to a fixed point $W^*$. Although the convergence property of Lemma 5 is proved for the second-order RKNN's, it can be expanded to the fourth-order RKNN's or even higher order RKNN's directly.

*2) First-Order NRLS Learning Algorithm* Due to the nonlinearity of the radial basis function network with respective to the center vector $\mathbf{t}_l$ and inverse variance matrix $\sigma_l$, it cannot put into the regression form such as (15) with respective to $\mathbf{t}_l$ and $\sigma_l$. Hence the zero-order NRLS algorithm derived in the

$$c_k(x(i)) = \frac{h \cdot e^{-\|\triangle_k\|^2} \cdot \sigma_k(x(i) - t_k)}{\left(1 + \sum_{j=1}^{N} W_j^+ e^{-\|\triangle_j\|^2}\sigma_j(x(i)-t_j)\cdot h\right)\left(1 + \sum_{j=1}^{N} W_j^0 e^{-\|\triangle_j\|^2}\sigma_j(x(i)-t_j)\cdot h\right)} \qquad (19)$$

above section cannot be applied directly to tune the center and variance of the radial basis function subnetwork in the RKNN. In this section, we shall derive another NRLS algorithm that can update output weights, centers, and variances of the radial basis functions simultaneously. This algorithm is called the first-order NRLS learning algorithm. The key point is to let the problem of minimizing $E(\omega)=\frac{1}{2}\sum_{i=1}^{L}(\mathbf{x}(i)-R_B(\mathbf{x}(i-1);\omega))^2$ be approximated by minimizing $\tilde{E}(\omega)$, which is the square error between the desired output $\mathbf{x}(i)$ and the output of linearly approximated neural network $R_B(\cdot;\omega)$ at the weight parameters $\omega(k)$ at time step $k$, where $R_B(\cdot)$ is a radial-basis-type RKNN. If we let $\tilde{E}(\omega)$ be the first-order approximation of $E(\omega)$ then we have

$$\tilde{E}(\omega)=\frac{1}{2}\sum_i\left\|\mathbf{x}(i)-R_B(\mathbf{x}(i-1);\omega(k)) - \left.\frac{\partial R_B(\mathbf{x}(i-1);\omega)}{\partial\omega}\right|_{\omega=\omega(k)}(\omega-\omega(k))\right\|^2 \tag{21}$$

where $\omega(k)$ denotes the parameters (including the output weights, center vectors, and variances) of the radial basis functions.

By using the similar techniques in Section III-B1, we can solve the following equation to minimize (21):

$$\begin{bmatrix}\left.\frac{\partial R_B^T}{\partial\omega}\right|_{\mathbf{x}(0),\omega(k)}\\ \vdots\\ \vdots\\ \left.\frac{\partial R_B^T}{\partial\omega}\right|_{\mathbf{x}(L-1),\omega(k)}\end{bmatrix}\triangle\omega=\begin{bmatrix}\mathbf{x}(1)-R_B(\mathbf{x}(0);\omega)\\ \vdots\\ \vdots\\ \mathbf{x}(L)-R_B(\mathbf{x}(L-1);\omega)\end{bmatrix} \tag{22}$$

in the least-square sense and update the parameters by

$$\omega(k+1)=\omega(k)+\triangle\omega^* \tag{23}$$

where $\triangle\omega^*$ is the least-square solution of (22).

To compute the term $\frac{\partial R_B(\cdot)}{\partial\omega}$ in (22), we need to use the recurrent algorithm described in (8) to obtain the regression form of (22). When $\omega(k)$ converges to a vector $\omega^*,\omega^*$ is an approximated solution which minimizes $\tilde{E}$, where $\tilde{E}$ is the approximated square error of (13).

*3) Second-Order NRLS Learning Algorithm* Similar to the concept in Section III-B2, let $\tilde{\tilde{E}}$ be the second-order approximation of $E$ described by (13). Its accuracy will be better than the accuracy of $\tilde{E}$ which is the first-order approximation of (13). With $\tilde{\tilde{E}}$, we can derive a second-order NRLS learning algorithm that can also tune the output weights, centers, and variances of the radial-basis-type RKNN's simultaneously.

The second-order approximation of $E$ is

$$\tilde{\tilde{E}}(\omega)=\frac{1}{2}\sum_i\left\|\mathbf{x}(i)-R_B(\mathbf{x}(i-1);\omega(k))-\left.\frac{\partial R_B}{\partial\omega}\right|_{\omega(k)}\cdot\triangle\omega-\frac{1}{2}(\triangle\omega)^T\left.\frac{\partial^2R_B}{\partial\omega^2}\right|_{\omega(k)}\triangle\omega\right\|^2. \tag{24}$$

The regression form of (24) at the $k$th time step is

$$\begin{bmatrix}\left(\frac{\partial R_B}{\partial\omega}+\triangle\omega\frac{\partial^2R_B}{\partial\omega^2}\right)\Big|_{\mathbf{x}(0),\omega(k)}\\ \vdots\\ \vdots\\ \left(\frac{\partial R_B}{\partial\omega}+\triangle\omega\frac{\partial^2R_B}{\partial\omega^2}\right)\Big|_{\mathbf{x}(L),\omega(k)}\end{bmatrix}\triangle\omega = \begin{bmatrix}\mathbf{x}(1)-R_B(\mathbf{x}(0);\omega(k))\\ \vdots\\ \vdots\\ \mathbf{x}(L)-R_B(\mathbf{x}(L-1);\omega(k))\end{bmatrix}. \tag{25}$$

The solution of (25) can be obtained by repeatedly using the fixed point method described in Section III-B1. Let $\triangle\omega^*(k)$ denote a solution of (25) in the least-square sense at the $k$th time step, and then update $\omega$ in the next time step $k+1$ by

$$\omega(k+1)=\omega(k)+\triangle\omega^*(k).$$

If $\{\omega(k)\}$ converges to $\omega^*$, then $\omega^*$ is the least-square solution of (13). Notice that $\omega$ in (24) can be the output weights, centers, or variances of a radial basis function subnetwork $N_f(\cdot)$.

The use of the second-order NRLS learning algorithm for training the RKNN has some computation problems. If we want to obtain the second-order derivative of $R_B(\cdot)$, we need to compute the coupled term $\frac{\partial R_B}{\partial\mathbf{x}\partial\omega}$ and the second derivative for $\omega$; i.e., $\frac{\partial^2R_B}{\partial\omega^2}$. This will increase the computation complexity and loading. In the simulations of the next section, we shall neglect the coupled terms. We shall also compare the approximation accuracy of the first and second-order NRLS learning algorithms by the simulations.

## IV. Simulations

In this section, we shall apply the RKNN's to model two ODE systems with unknown structures. From the simulations, we shall demonstrate some good properties of the RKNN including its high accuracy in long-term prediction, and good prediction power even for adaptively changing time step. In modeling ODE systems, *a priori* knowledge about the structure of the modeled systems, if available, can be incorporated into the RKNN to speed up the learning and simplify the network structure (e.g., small node number). Such knowledge can be possibly obtained from nature's physical law. The learning algorithms developed in Section III will be used to train the RKNN's in the following two examples. The $N_f$ subnetwork of the RKNN's in the following simulations will be the radial basis function network. The initial centers of the radial basis functions are chosen to be uniformly distributed in the input space, and the variances are decided by equally overlapping principal.

*Example 1:* Consider a second-order system, known as Vander Pol's equation

$$\ddot{y}(t)-(1-y^2(t))\dot{y}(t)+y(t)=0$$

with

$$y(0)=y_0,\quad \dot{y}(0)=v_0 \tag{26}$$

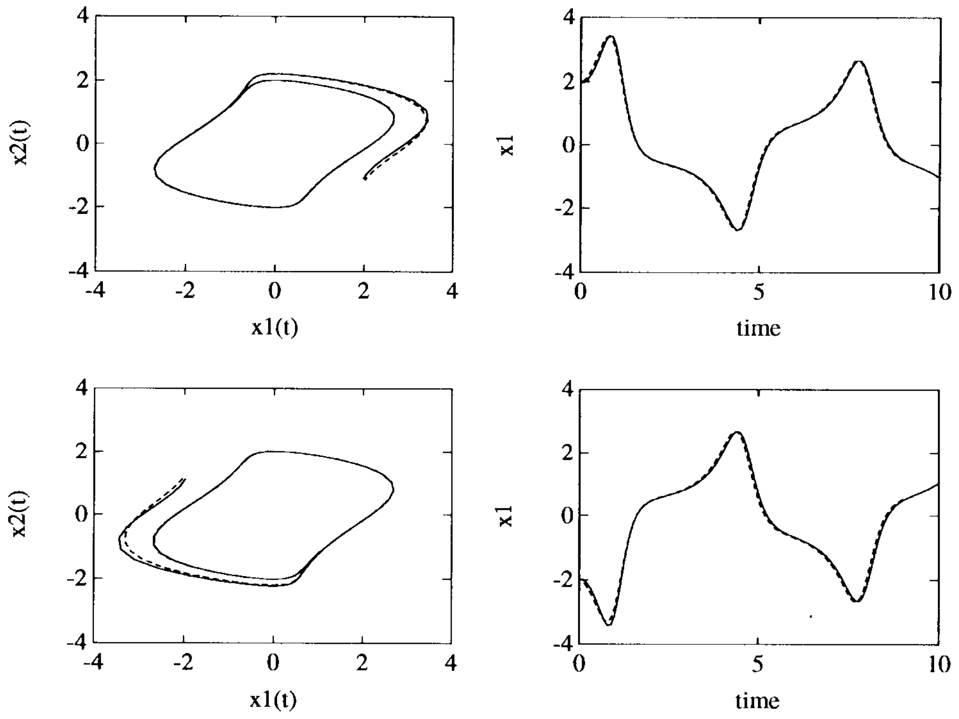

Fig. 2. Predicted trajectories of the RKNN $(\cdots)$ with time step $h = 0.09$ and desired trajectories $(-)$ in Example 1, where the initial conditions are $(2, -1)$ and $(-2, 1)$.

where $0 \leq t \leq T$. Assume that we do not know the structure (equation) of this system, but we can measure the position $y$ and velocity $\dot{y}$ noiselessly at each time step $ih$, where $h = 0.1$ second. Let $y(ih; y_0, v_0), \dot{y}(ih; y_0, v_0)$ denote a solution of (26) at time step $t = ih$, where $i = 1, \cdots, \lfloor \frac{T}{h} \rfloor$. Let $S$ be the set of collected trajectories, whose initial states are uniformly distributed in the region $D$ of $(y, \dot{y})$ space. Using $S$ as a training set to train the parameters of $N_f(\cdot)$, four of which constitute the RKNN, we can obtain a RKNN that approximates Vander Pol's system described by (26) for any initial condition belongs to $D$.

For representation clarity, let $y(t)$ be denoted by $x_2(t)$ and $\dot{y}(t)$ by $x_1(t)$, then $N_f(\cdot) = N_f(x_1, x_2)$. Here the $N_f(\cdot)$ network can be considered as a mapping from $(x_1, x_2)$ to $(N_f(x_1, x_2), x_1)$, and $N_f(\cdot)$ network will identify the variable $\ddot{y}(t)$, where

$$N_f(x_1, x_2) = \sum_{i=1}^{N} W_i \phi_i(x_1, x_2; t_{1i}, t_{2i}, \sigma_{1i}, \sigma_{2i})$$

is the weighted sum of radial basis functions $\phi_i$'s.

In the simulations, the training set $S$ contains 41 training trajectories whose initial states $(x_1, x_2)$ are uniformly distributed in a square region $D = [-3, 3] \times [-2, 2] \in \Re^2$. In the training phase, we first use the gradient learning algorithm to tune the parameters $W_i$, $t_{1i}$, $t_{2i}$, $\sigma_{1i}$, $\sigma_{2i}$ until the error between training trajectories and the RKNN output trajectories cannot be further reduced. We then fix $t_i$'s and $\sigma_i$'s, and use the zero-order NRLS algorithm to tune the weights $W_i$ to converge to the solution, $W^*$. The use of NRLS algorithm can further reduce the errors between the desired values and RKNN outputs over the whole training set $S$.

To compare the prediction accuracy of the RKNN and DMNN, we use a DMNN, $\hat{N}_f(x_1, x_2)$, to do the same modeling task. The used $\hat{N}_f(x_1, x_2)$ is also a radial basis function network, which has the same number of nodes as the $N_f(x_1, x_2)$ of the RKNN in the above simulations. After training the $\hat{N}_f(x_1, x_2)$ using the same procedure for training the RKNN (i.e., gradient learning followed by zero-order least-square learning), we test the long-term parallel-model prediction capability of the trained RKNN and DMNN in the generalization phase. In this phase, the two networks are test on a collection of different initial states distributed in $D$. Table I lists the average root-mean-square (rms) errors of long-term prediction over 100 time steps under the test initial states. The results indicate that the RKNN has much better long-term prediction capability than the DMNN. Fig. 2 shows the predicted trajectories of the RKNN using time step $h = 0.09$ in the generalization phase, instead of $h = 0.1$ used in the training phase. As shown in Fig. 3, the trained DMNN cannot correctly predict the trajectories with time step $h = 0.09$. This illustrates that the RKNN can perform the prediction well even with variable time-step size.

*Example 2:* In this example, we apply the RKNN to model the trajectory of a vertically falling body. Assume a radar had tracked and recorded two different scenarios trajectories, $S^1(t), S^2(t)$, of a falling body starting at two different initial conditions, where the trajectory $S^i(t)$ indicates the falling body's altitude position $x_1^i(t)$ and vertical velocity $x_2^i(t)$ at time $t$; i.e., $S^i(t) = (x_1^i(t), x_2^i(t))$ is a trajectory in the $\Re^2$ space. These variables are defined in Fig. 4. In the simulations, we assume that the trajectory data are governed by the

TABLE I
COMPARISON OF RMS PREDICTION ERRORS OF THE RKNN AND DMNN OVER 100 TIME STEPS FOR THE VANDER POL'S SYSTEM IN EXAMPLE 1

| Initial condition | RKNN ($N_f(\cdot)$) | | DMNN ($\hat{N}_f(\cdot)$) | |
|---|---|---|---|---|
| $(x_1, x_2)$ | Velocity error | Position error | Velocity error | Position error |
| ( 0.5, 0.5) | 0.0040 | 0.0030 | 0.0201 | 0.0144 |
| (-0.5, 0.5) | 0.0018 | 0.0015 | 0.0128 | 0.0089 |
| (-0.5, -0.5) | 0.0007 | 0.0007 | 0.0587 | 0.0429 |
| ( 0.5, -0.5) | 0.0016 | 0.0013 | 0.0025 | 0.0020 |
| ( 1.0, 1.0) | 0.0016 | 0.0011 | 0.0088 | 0.0060 |
| (-2.0, -2.0) | 0.0052 | 0.0041 | 0.0107 | 0.0081 |
| ( 2.0, 2.0) | 0.0027 | 0.0021 | 0.0089 | 0.0069 |
| ( 2.5, -1.0) | 0.0136 | 0.0093 | 0.0305 | 0.0255 |
| (-2.5, 1.0) | 0.0194 | 0.0135 | 0.0341 | 0.0223 |
| (-0.2, 0.3) | 0.0012 | 0.0007 | 0.0106 | 0.0097 |
| ( 0.2, -0.3) | 0.0035 | 0.0026 | 0.0124 | 0.0105 |
| (-0.2, -0.3) | 0.0032 | 0.0022 | 0.0554 | 0.0405 |
| ( 0.2, 0.3) | 0.0023 | 0.0015 | 0.0506 | 0.0370 |
| ( 0.1, 0.0) | 0.0055 | 0.0049 | 0.1149 | 0.1318 |
| (-0.1, 0.0) | 0.0049 | 0.0045 | 0.0629 | 0.0625 |
| ( 0.0, 0.1) | 0.0079 | 0.0055 | 0.0908 | 0.0739 |
| ( 0.0, -0.1) | 0.0077 | 0.0054 | 0.0795 | 0.0681 |
| (-0.1, 0.5) | 0.0012 | 0.0010 | 0.0045 | 0.0031 |
| ( 2.0, 2.0) | 0.0027 | 0.0021 | 0.0089 | 0.0069 |
| (-1.0, -0.5) | 0.0018 | 0.0014 | 0.0201 | 0.0162 |

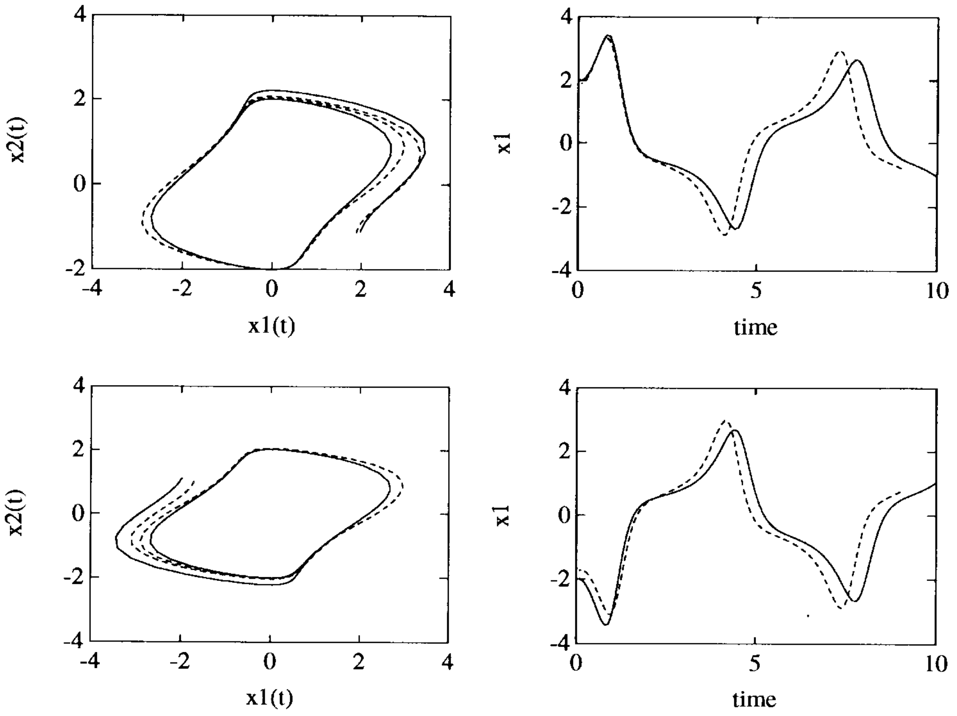

Fig. 3. Predicted trajectories of the DMNN $(\cdots)$ with time step $h = 0.09$ and desired trajectories $(-)$ in Example 1,where the initial conditions are $(2, -1)$ and $(-2, 1)$.

following system equations:

$$\begin{aligned}\dot{x}_1 &= -x_2(t),\\ \dot{x}_2 &= -c_1\exp(c_2x_1(t))x_2^2(t)\end{aligned} \tag{27}$$

where $c_1 = 3 \times 10^{-3}$ (drag coefficient), $c_2 = -5 \times 10^{-5}$ (air density), with two initial conditions $(x_1^1(0), x_2^1(0))$ as $(200 \times 10^3$ ft, $16 \times 10^3$ ft/s) and $(x_1^2(0), x_2^2(0)) = (300 \times 10^3$ ft, $20 \times 10^3$ ft/s).

Under the assumption that the knowledge about (27) is unknown, the problem is to design an RKNN, $\dot{\mathbf{x}}(t) = N_f(\mathbf{x})$, where $\mathbf{x} = (x_1, x_2)$, which can be used to approximate the system described in (27) and predict the long-term behaviors

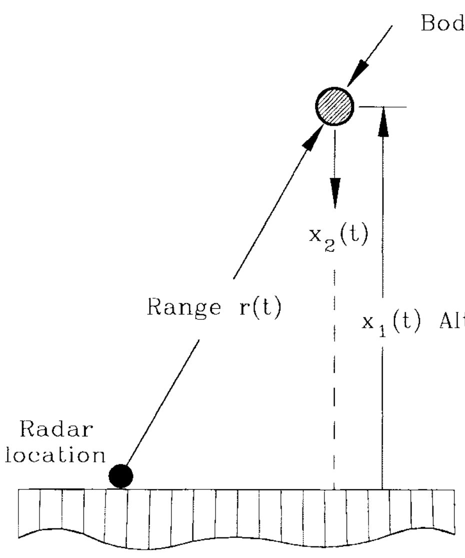

Fig. 4. Geometry of the falling body problem in Example 2.

of falling trajectories with different initial conditions. In this example, we shall demonstrate two RKNN's constructed by two neural networks, $N_f^1(\mathbf{x})$ and $N_f^2(\mathbf{x})$, respectively. The network $N_f^1(\mathbf{x})$ is constructed according to the physical law of falling body; that is, the drag force is proportion to the square of velocity. Its structure is shown in Fig. 5(a). The second network $N_f^2(\mathbf{x})$ is a normal radial basis function network shown in Fig. 5(b). We use the first-order and second-order NRLS learning algorithms developed in Section III to tune the weights and variances of radial basis functions simultaneously until the outputs of the RKNN converge to the training trajectories $S^1(t), S^2(t)$. To compare the prediction accuracy of the RKNN and DMNN, we use a DMNN $N_f^3(\mathbf{x})$ as shown in Fig. 5(c) to do the same learning task. The DMNN has the same structure as $N_f^2(\mathbf{x})$.

In the prediction tests, we test the three trained networks, two RKNN's and one normal radial basis function network, on three different initial conditions around the region of training trajectories $S^1(t), S^2(t)$. Using the same training data and same training method (the zero-order NRLS algorithm), the rms of prediction errors on each case is listed in Table II. Notice that, in Table II, the trajectories with initial points (200, 16) and (300, 20) are training trajectories, and that with initial point (300, 15) and that with initial points (200, 21) and (300, 22) are interpolative and extrapolative testing trajectories, respectively. The results show that $N_f^1(\mathbf{x})$ has better extrapolation and interpolation capability than $N_f^2(\mathbf{x})$ and $N_f^3(\mathbf{x})$, and $N_f^2(\mathbf{x})$ is more accurate than $N_f^3(X)$.

We next use the first-order and second-order NRLS learning algorithms to tune the weights and variances of the RKNN with $N_f^2(x)$ simultaneously. The results are shown in Table III, in which we compare the long-term prediction accuracy of different learning algorithms for the RKNN with $N_f^2(x)$, including gradient learning algorithm, zero-order NRLS, first-order NRLS, and second-order NRLS learning algorithms. As compared to Table II, we find that the higher-order NRLS schemes improve the prediction power of the RKNN greatly, especially in the divergence case with initial

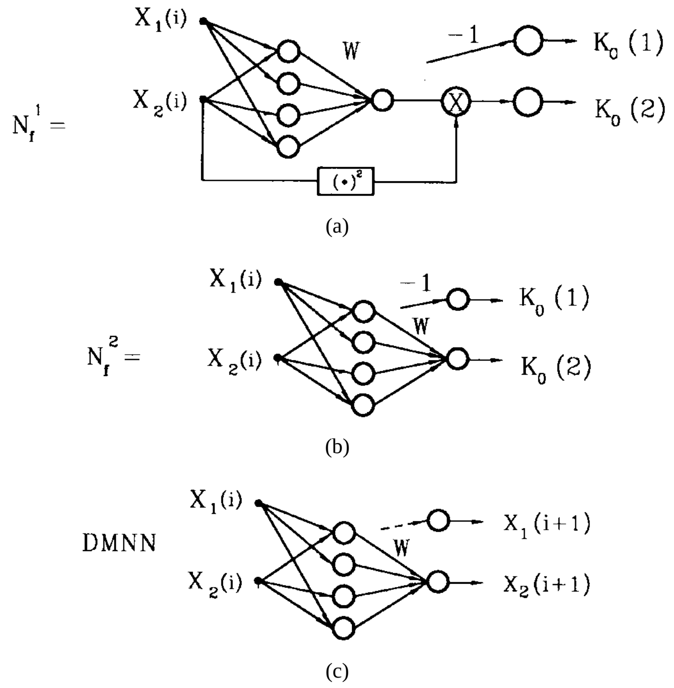

Fig. 5. Structures of the RKNN's and DMNN used to predict the behavior of a falling body in Example 2. (a) The network constructed by the physical law of falling body. (b) The normal radial basis function network. (c) The direct-mapping neural network.

TABLE II
COMPARISON OF RMS PREDICTION ERRORS OF THE RKNN'S AND DMNN USING THE SAME ZERO-ORDER NRLS LEARNING ALGORITHM IN EXAMPLE 2 (UNIT $\times 10^3$ FT)

| Initial point | RKNN | RKNN | DMNN |
|---|---|---|---|
| $(x_1, x_2)$ | $N_f^1(x)$ | $N_f^2(x)$ | $N_f^3(x)$ |
| ( 200, 21 ) | ( 7.75, 1.26 ) | ( 17.3, 2.48 ) | ( 8.95, 2.34 ) |
| ( 200, 16 ) | ( 0.28, 0.09 ) | ( 0.33, 0.21 ) | ( 2.36, 0.50 ) |
| ( 300, 15 ) | ( 2.19, 0.31 ) | ( 2.54, 0.42 ) | ( 909, 50.0 ) |
| ( 300, 20 ) | ( 0.25, 0.10 ) | ( 0.54, 0.21 ) | ( 1.04, 0.60 ) |
| ( 300, 22 ) | ( 39.4, 4.70 ) | ( 560, 47.2 ) | ( 47.0, 9.30 ) |

condition $(x_1^1(0), x_2^1(0)) = (300 \times 10^3$ ft, $22 \times 10^3$ ft/s). In this case, the RKNN constructed by $N_f^2(x)$ and trained by the zero-order NRLS algorithm cannot predict the trajectory successfully. Figs. 6 and 7 show the long-term predicted trajectories of the RKNN with $N_f^2(x)$ trained by the first and second-order NRLS learning algorithms, respectively. Fig. 8 shows the prediction capability of the RKNN with $N_f^1(x)$ trained by the zero-order NRLS algorithm. It is observed that due to the *a priori* knowledge incorporated in the RKNN with $N_f^1(x)$, the RKNN can predict well even using the simple gradient learning algorithm followed by the zero-order NRLS learning algorithm.

## V. CONCLUSION

In this paper, we constructed an RKNN for identification of ODE systems with unknown right-hand-side functions. We also derived two classes of learning algorithms for training the RKNNs; gradient learning algorithms and NRLS learning algorithms. The NRLS is a generalization of the recursive least-square algorithm to nonlinear cases such that it can tune the parameters in the hidden layers of the RKNN's (such as the

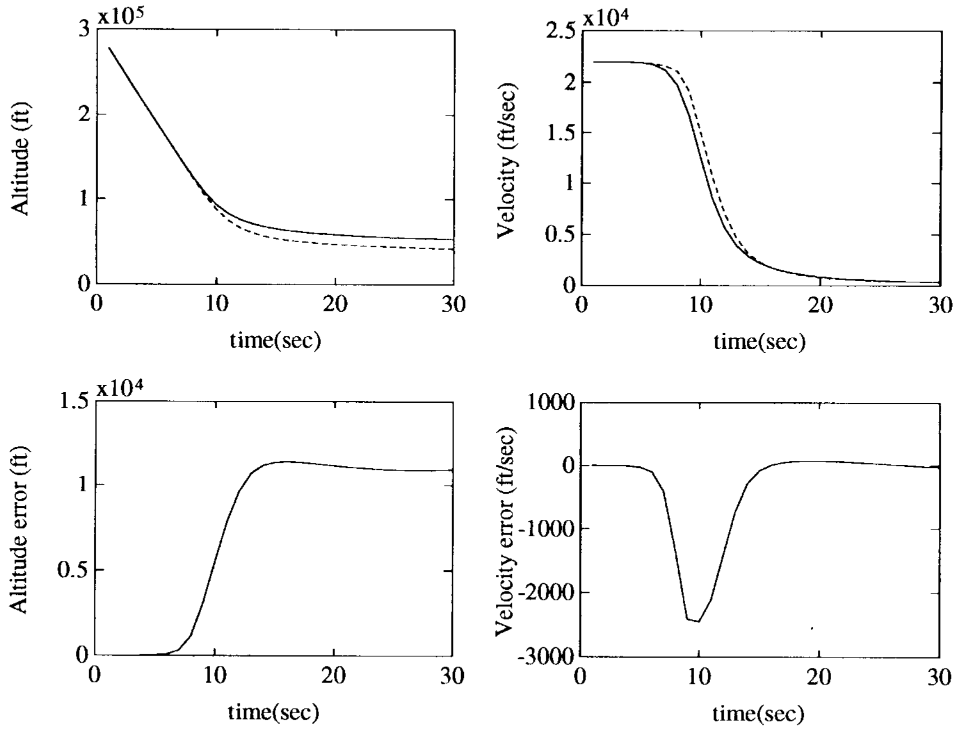

Fig. 6. Predicted trajectories of the RKNN with $N_f^2(\mathbf{x})$ trained by the first-order NRLS algorithm in Example 2.

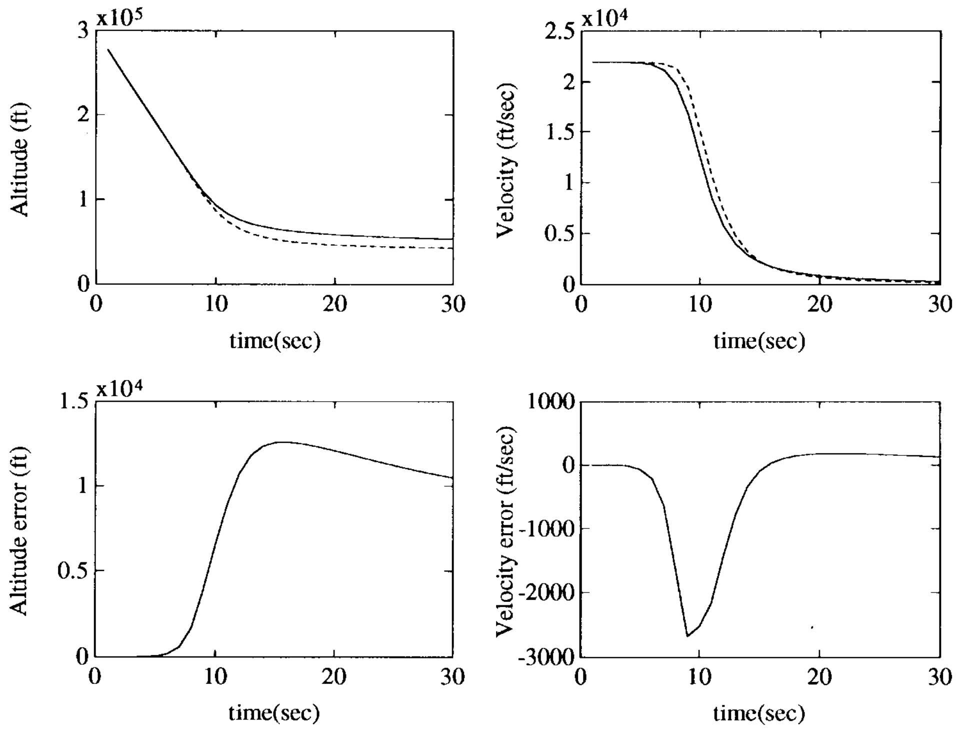

Fig. 7. Predicted trajectories of the RKNN with $N_f^2(\mathbf{x})$ trained by the second-order NRLS algorithm in Example 2.

centers and variances of the radial-basis-type RKNN) fastly. The convergence property of the proposed NRLS algorithms applied to the RKNN's was studied theoretically. The RKNN's have several good properties. 1) Since the RKNN estimates the derivative (changing rate) of system states (i.e., the right-hand side of ODE's) directly in their subnetworks based on space-domain interpolation instead of time-domain interpolation, it can do long-term prediction of the identified system behavior well and is good at parallel-model prediction. 2) With the designed structure and proposed learning schemes, the RKNN's perform high-order discretization of ODE systems with unknown right-hand-side functions *implicitly* (i.e., internally in the network) while keeping the simplicity and tractability of the first-order discretization scheme. 3) The RKNN is shown

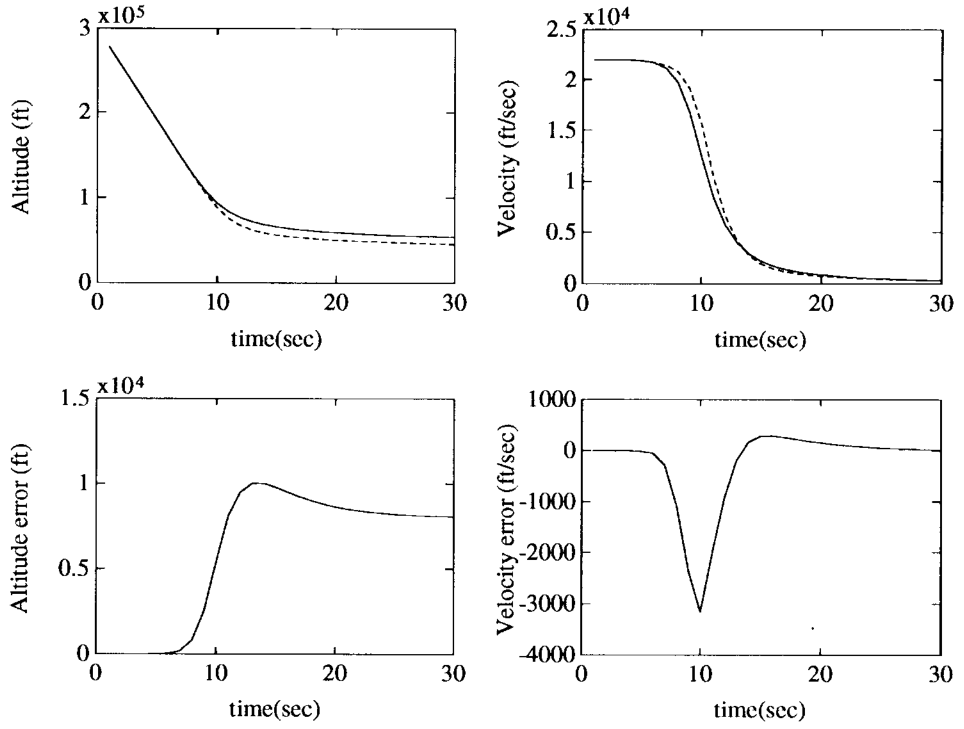

Fig. 8. Predicted trajectories of the RKNN with $N_f^1(\mathbf{x})$ trained by the zero-order NRLS algorithm in Example 2.

TABLE III
COMPARISON OF RMS PREDICTION ERRORS OF THE RKNN WITH $N_f^2(\mathbf{x})$ TRAINED BY DIFFERENT LEARNING ALGORITHMS IN EXAMPLE 2 (UNIT $\times 10^3$ FT)

| Initial point $(x_1, x_2)$ | Gradient+Zero-order NRLS | First-order NRLS | Second-order NRLS |
|---|---|---|---|
| ( 200, 21 ) | ( 17.3, 2.48 ) | ( 10.8, 1.54 ) | ( 12.8, 1.82 ) |
| ( 200, 16 ) | ( 0.33, 0.21 ) | ( 0.15, 0.05 ) | ( 0.06, 0.02 ) |
| ( 300, 15 ) | ( 2.54, 0.42 ) | ( 3.30, 0.44 ) | ( 4.00, 0.59 ) |
| ( 300, 20 ) | ( 0.54, 0.21 ) | ( 0.27, 0.05 ) | ( 0.17, 0.04 ) |
| ( 300, 22 ) | ( 560, 47.2 ) | ( 49.0, 4.57 ) | ( 52.0, 4.90 ) |

theoretically and experimentally to be superior to normal neural networks in generalization and long-term prediction capability for the same network size and training procedure. 4) The RKNN needs no tapped delay line or internal memory for identifying memoryless systems, and thus without the problems of deciding the length or size of the tapped delay line or internal memory existent in normal neural identifier. 5) Since the RKNN models the right-hand side of ODE in its subnetworks directly, some known continuous relationship (physical laws) of the identified system can be incorporated into the RKNN to cut down the network size, speed up its learning, and enhance its long-term prediction capability. 6) The RKNN can predict the system behavior at any time instant, not limited by fixed time step (fixed sampling time) as the case in normal neural modeling. Although the algorithm derivation and theory proof focused on the fourth-order or second-order RKNN's in this paper, they can be generalized to any $n$-order RKNN's directly. In future works, we shall focus on the real parameter aggregates instead of the unstructured parameter set $\omega$. From this point of view and because of some of the parameters have matrix form, it would be interesting to use Kronecker products (tensor products) and matrix calculus in the developments in order to get a more compact notation [23].

## APPENDIX

This appendix derives the regression form of $\mathbf{y}(i)-\mathbf{x}(i-1)$ in (14). Equation (14) can be written as

$$\begin{aligned}
&\mathbf{y}(i)-\mathbf{x}(i-1)\\
&= \frac{h}{6}\Bigg\{\sum_{l=1}^{N} W_l\phi_l(\mathbf{x}(i-1)) + 2\sum_{l=1}^{N} W_l\phi_l\Big(\mathbf{x}(i-1)\\
&\quad + \frac{h}{2}\mathbf{k}_0(i-1)\Big) + 2\sum_{l=1}^{N} W_l\phi_l\Big(\mathbf{x}(i-1)+\frac{h}{2}\mathbf{k}_1(i-1)\Big)\\
&\quad + \sum_{l=1}^{N} W_l\phi_l(\mathbf{x}(i-1)+h\mathbf{k}_2(i-1))\Bigg\}\\
&= W_1\Bigg\{\frac{h}{6}\phi_1(\mathbf{x}(i-1)) + \frac{2h}{6}\phi_1\Big(\mathbf{x}(i-1)+\frac{h}{2}\mathbf{k}_0(i-1)\Big)\\
&\quad + \frac{2h}{6}\phi_1\Big(\mathbf{x}(i-1)+\frac{h}{2}\mathbf{k}_1(i-1)\Big)\\
&\quad + \frac{h}{6}\phi_1(\mathbf{x}(i-1)+h\mathbf{k}_2(i-1))\Bigg\}
\end{aligned}$$

$$
\begin{aligned}
&+ W_2\left\{\frac{h}{6}\phi_2(\mathbf{x}(i-1)) + \frac{2h}{6}\phi_2\left(\mathbf{x}(i-1)+\frac{h}{2}\mathbf{k}_0(i-1)\right)\right.\\
&+ \frac{2h}{6}\phi_2\left(\mathbf{x}(i-1)+\frac{h}{2}\mathbf{k}_1(i-1)\right)\\
&\left.+ \frac{h}{6}\phi_2(\mathbf{x}(i-1)+h\mathbf{k}_2(i-1))\right\} + \cdots\\
&+ W_N\left\{\frac{h}{6}\phi_N(\mathbf{x}(i-1)) + \frac{2h}{6}\phi_2(\mathbf{x}(i-1)\right.\\
&+ \frac{h}{2}\mathbf{k}_0(i-1)) + \frac{2h}{6}\phi_N\left(\mathbf{x}(i-1)+\frac{h}{2}\mathbf{k}_1(i-1)\right)\\
&\left.+ \frac{h}{6}\phi_N(\mathbf{x}(i-1)+h\mathbf{k}_2(i-1))\right\}.
\end{aligned}
$$

Hence we obtain the nonlinear regression form, $\boldsymbol{\phi}(x(i-1);W)$, of $\mathbf{y}(i)-\mathbf{x}(i-1)$ expanded by radial basis functions $\phi_j$ as

$$
\begin{aligned}
&\boldsymbol{\phi}^T(\mathbf{x}(i-1);W)\\
&= \left\{\frac{h}{6}\left[\phi_1(\mathbf{x}(i-1)) + 2\phi_1\left(\mathbf{x}(i-1)\right.\right.\right.\\
&\quad \left.+\frac{h}{2}\mathbf{k}_0(i-1)\right) + 2\phi_1\left(\mathbf{x}(i-1)+\frac{h}{2}\mathbf{k}_1(i-1)\right)\\
&\quad \left.+\phi_1(\mathbf{x}(i-1)+h\mathbf{k}_2(i-1))\right], \cdots, +\frac{h}{6}\left[\phi_N(\mathbf{x}(i-1))\right.\\
&\quad +2\phi_N\left(\mathbf{x}(i-1)+\frac{h}{2}\mathbf{k}_0(i-1)\right) + 2\phi_N\left(\mathbf{x}(i-1)\right.\\
&\quad \left.\left.\left.+\frac{h}{2}\mathbf{k}_1(i-1)\right) + \phi_N(\mathbf{x}(i-1)+h\mathbf{k}_2(i-1))\right]\right\}
\end{aligned}
$$

where

$$
\begin{aligned}
\mathbf{k}_0(i-1) &= \sum_{l=1}^{N} W_l\phi_l(\mathbf{x}(i-1))\\
\mathbf{k}_1(i-1) &= \sum_{l=1}^{N} W_l\phi_l\left(\mathbf{x}(i-1)+\frac{h}{2}\mathbf{k}_0(i-1)\right)\\
\mathbf{k}_2(i-1) &= \sum_{l=1}^{N} W_l\phi_l\left(\mathbf{x}(i-1)+\frac{h}{2}\mathbf{k}_1(i-1)\right)\\
\mathbf{k}_3(i-1) &= \sum_{l=1}^{N} W_l\phi_l(\mathbf{x}(i-1)+h\mathbf{k}_2(i-1)).
\end{aligned}
$$

## ACKNOWLEDGMENT

The authors would like to thank the reviewers for their helpful suggestions in improving the quality of the final manuscript.

## REFERENCES

[1] W. T. Miller, R. S. Sutton, and P. J. Werbos, *Neural Networks for Control*. Cambridge, MA: MIT Press, 1990.
[2] K. J. Hunt, D. Sbarbaro, R. Zbikowski, and P. J. Gawthrop, "Neural networks for control systems: A survey," *Automatica*, vol. 28, no. 6, pp. 1083–1112, Nov. 1992.
[3] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical system using neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 4–27, 1990.
[4] Y. Ichikawa and T. Sawa, "Neural network application for direct feedback controllers," *IEEE Trans. Neural Networks*, vol. 3, pp. 224–231, 1992.
[5] P. S. Sastry, G. Santharam, and K. P. Unnikrishnan, "Memory neural networks for identification and control of dynamical systems," *IEEE Trans. Neural Networks*, vol. 5, pp. 306–319, 1994.
[6] R. J. Williams, "Adaptive state representation and estimation using recurrent connectionist networks," in *Neural Networks for Control*, W. T. Miller, R. S. Sutton, and P. J. Werbos, Eds. Cambridge, MA: MIT Press, 1990.
[7] A. G. Parlos, K. T. Chong, and A. F. Atiya, "Application of recurrent multilayer perceptron in modeling complex process dynamics," *IEEE Trans. Neural Networks*, vol. 5, pp. 255–266, 1994.
[8] P. J. Werbos, "Backpropagation through time: What it does and how to do it," in *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.
[9] S. W. Piche, "Steepest descent algorithms for neural network controllers and filters," *IEEE Trans. Neural Networks*, vol. 5, pp. 198–212, 1994.
[10] J. D. Lambert, *Computation Methods in O.D.E.* New York: Wiley, 1973, ch. 4.
[11] F. Caller and C. Desoer, *Linear System Theory*. New York: Spring-Verlag, 1992.
[12] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 3, pp. 551–560, 1989.
[13] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Computa.*, vol. 3, pp. 246–257, 1991.
[14] I. G. Petrovski, *Ordinary Differential Equations*. Englewood Cliffs, NJ: Prentice–Hall, 1966.
[15] E. D. Sontag, *Mathematical Control Theory: Deterministic Finite-Dimensional Systems*. New York: Spring-Verlag, 1990.
[16] E. B. Kosmatopoulos, M. M. Polycarpou, M. A. Christodoulou, and P. A. Ioannou, "High-order neural-network structures for identification of dynamical systems," *IEEE Trans. Neural Networks*, vol. 6, pp. 422–431, 1995.
[17] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, 2nd ed. New York: Springer-Verlag, 1993.
[18] V. Lakshmikantham and D. Trigiante, *Theory of Difference Equations: Numerical Methods and Applications*. Boston, MA: Academic, 1988.
[19] R. Fletcher, *Practical Methods of Optimization*, 2nd ed. Chichester, U.K.: Wiley, 1987.
[20] S. Haykin, *Neural Networks*. New York: Macmillan, 1994, ch. 7.
[21] G. C. Goodwin and K. S. Sin, *Adaptive Filtering Prediction and Control*. Englewood Cliffs, NJ: Prentice-Hall, 1984, ch. 3.
[22] P. Linz, *Theoretical Numerical Analysis*. New York: Wiley, 1979, pp. 126–136.
[23] A. Weinmann, *Uncertain Models and Robust Control*. New York: Springer-Verlag, 1991, ch. 4 and 5.

**Yi-Jen Wang** was born in Taipei, Taiwan, R.O.C., in 1958. He received the B.S. degree from Fu-Jen Catholic University, Taiwan, in 1981, and the M.S. degree from the National Tsing-Hua University, Taiwan, in 1983, both in applied mathematics. He is currently working toward the Ph.D. degree at the Department of Electrical and Control Engineering, National Chiao-Tung University, Taiwan.

From 1983 to 1993 he was an Assistant Researcher in the Chung-Shan Institute of Science and Technology (CSIST), Taiwan. His current research interests include neural networks, optimal control, and digital signal processing.

**Chin-Teng Lin** (S'88–M'91) received the B.S. degree in control engineering from the National Chiao-Tung University, Taiwan, R.O.C., in 1986 and the M.S.E.E. and Ph.D. degrees in electrical engineering from Purdue University, West Lafayette, IN, in 1989 and 1992, respectively.

Since August 1992, he has been with the College of Electrical Engineering and Computer Science, National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., where he is currently a Professor of Electrical and Control Engineering. His current research interests include fuzzy systems, neural networks, intelligent control, human–machine interface, and video and audio processing. He is the coauthor of *Neural Fuzzy Systems—A Neuro-Fuzzy Synergism to Intelligent Systems* (Englewood Cliffs, NJ: Prentice-Hall, 1996) and the author of *Neural Fuzzy Control Systems with Structure and Parameter Learning* (Singapore: World Scientific, 1994).

Dr. Lin is a member of Tau Beta Pi and Eta Kappa Nu. He is also a member of the IEEE Computer Society, the IEEE Robotics and Automation Society, and the IEEE Systems, Man, and Cybernetics Society.