Training Recurrent Neural Networks with the Levenberg–Marquardt Algorithm for Optimal Control of a Grid-Connected Converter

Xingang Fu, *Student Member, IEEE,* Shuhui Li, *Senior Member, IEEE,*
Michael Fairbank, *Member, IEEE,* Donald C. Wunsch, *Fellow, IEEE,* and Eduardo Alonso

Abstract -- This paper investigates how to train a recurrent neural network (RNN) using the Levenberg-Marquardt (LM) algorithm, as well as how to implement optimal control of a grid-connected converter (GCC) using a RNN. To successfully and efficiently train a RNN using the LM algorithm, a new Forward Accumulation Through Time (FATT) algorithm is proposed to calculate the Jacobian matrix required by the LM algorithm. This paper explores how to incorporate FATT into the LM algorithm. The results show that the combination of the LM and FATT (LM-FATT) algorithms trains RNNs better than the conventional Backpropagation Through Time (BPTT) algorithm. The paper presents an analytical study on the optimal control of GCCs, including theoretically ideal optimal and suboptimal controllers. To overcome the inapplicability of the optimal GCC controller under practical conditions, a new RNN controller with an improved input structure is proposed to approximate the ideal optimal controller. The performance of an ideal optimal controller and a well-trained RNN controller was compared in close to real-life power converter switching environments, demonstrating that the proposed RNN controller can achieve close to ideal optimal control performance, even under low sampling rate conditions. The excellent performance of the proposed RNN controller under challenging and distorted system conditions further indicates the feasibility of using a RNN to approximate optimal control in practical applications.

***Index Terms* – optimal control, recurrent neural network, Levenberg-Marquardt, Forward Accumulation Through Time, Jacobian matrix, Backpropagation Through Time, dynamic programming, d-q vector control, grid-connected converter**

I. Introduction

In modern electric power systems, power electronic converters play an increasingly important role in the integration of smart grids, renewable energy resources and energy storage devices (Fig. 1). A grid-connected converter (GCC) is a key component that physically connects wind turbines, solar panels, or batteries to the grid [1], [2], and [3]. A critical issue for energy generation from renewable sources and for smart grid integration is the control of the GCC (green boxes in Fig. 1). Traditionally, this type of converter is controlled using a standard decoupled d-q vector control approach [4]. However, recent studies have noted the limitations of the standard vector controller [4]. Practically, these limitations could result in low power quality, inefficient power generation and transmission, and a possible loss of electricity, all of which cause loss of dollars for both electric utility companies and electric energy customers.

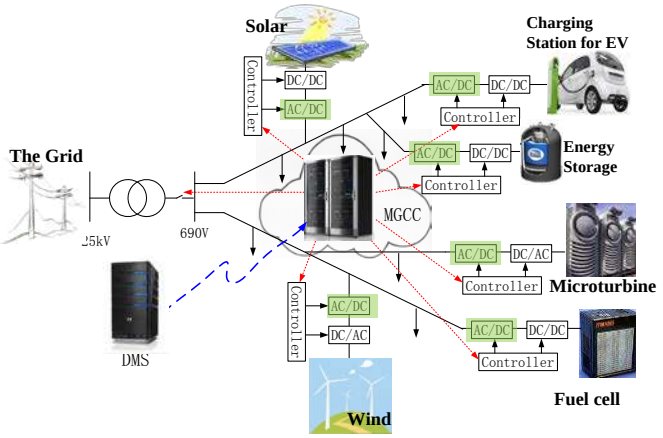


Fig.1 A microgrid with GCC-interfaced distributed energy sources

Recent research [5] has shown that recurrent neural networks (RNNs) can be trained and used to control grid-connected converters. In [5], the RNN implemented a dynamic programming (DP) algorithm and was trained using Backpropagation Through Time (BPTT). BPTT was combined with Resilient Propagation (RPROP) to accelerate the training. Compared to conventional standard vector control methods, the neural network vector controller produced an extremely fast response time, low overshoot, and, in general, the best performance [6]. In [7], it was shown that the neural network vector control technique can be extended to other applications, such as brushless dc motor drives.

For both applications, conventional control techniques, such as PID and predictive control, were integrated into the DP-based neural network design [5]-[7]. This unifying approach produced some important advantages, including zero steady-state error, great control under physical system constraints, and the ability to exhibit adaptive control

This work was supported in part by the U.S. National Science Foundation under Grant EECS 1102038/1102159, the Mary K. Finley Missouri Endowment, and the Missouri S&T Center for Infrastructure Engineering Studies and Intelligent Systems Center.
Xingang Fu and Shuhui Li are with the Department of Electrical & Computer Engineering, The University of Alabama, Tuscaloosa, AL 35487, USA (email: xfu@crimson.ua.edu, sli@eng.ua.edu).
Michael Fairbank and Eduardo Alonso are with the School of Mathematics, Computer Science and Engineering, City University London, EC1V 0HB, UK (email: michael.fairbank@virgin.net, e.alonso@city.ac.uk).
Donald C. Wunsch, the Mary K. Finley Missouri Distinguished Professor, is with the Department of Electrical & Computer Engineering, Missouri University of Science and Technology, Rolla, MO 65409-0040, USA (email: dwunsch@mst.edu).

behavior, even though the RNN controller was trained entirely offline. However, for such an integrative neural network control structure, training the RNN controller was very difficult using BPTT combined with RPROP due to issues such as slow convergence and oscillation problems that usually cause training to diverge. This paper also addresses the practical limitations that may prevent the creation of an optimal neural network controller based on DP. Both issues have caused great challenges in applying the neural network controller to a real-life system, which served as the motivation for the research presented here.

In [8], Real Time Recurrent Learning (RTRL) was proposed to train a RNN. However, the high computational cost of the RTRL causes it to be appropriate only for the online training of a small RNN [8]-[10]. Alternatively, Extended Kalman Filters (EKF) have proven useful in training RNN controllers for linear and nonlinear dynamical systems [11]-[13]. Nevertheless, EKFs are also computationally expensive because each estimation requires numerous matrix calculations. In addition, the eventual success and quality of EKF training depends highly on professional experience, including an appropriate selection of the network architecture, learning rates, and network inputs [10]. Levenberg-Marquardt (LM) ([14]-[16]) is used widely to train feed-forward networks. Although some research has shown the potential of training RNNs using LM [17]-[19], it has not been used broadly for this purpose. Furthermore, none of these studies have described how the Jacobian matrix was defined and calculated for a RNN. In the study presented in this paper, we investigated how the Jacobian matrix can be evaluated for a RNN by unrolling it forward through time.

In summary, the purpose of the study was to implement optimal GCC control under practical constraints, and to investigate how to utilize LM to improve RNN training. Accordingly, the defining features and contributions of the paper include: 1) an analytical study of the ideal optimal and suboptimal GCC controllers, 2) a Forward Accumulation Through Time (FATT) algorithm to calculate the Jacobian matrix efficiently for RNN training, 3) an approach to integrate FATT with LM to accelerate RNN training, and 4) a new RNN vector controller with improved input structure for a GCC to increase RNN adaptability to broad vector control applications.

The remainder of the paper is organized as follows. First, Section II introduces a GCC vector control model and analyzes ideal optimal GCC control characteristics. Section III illustrates the proposed RNN controller structure; it also explains how to extend LM to train a RNN and how to calculate the Jacobian matrix required by LM for efficient RNN training. Section IV compares the training performance of the proposed FATT-LM training algorithm with that of the BPTT training algorithm. Section V compares the performance of the ideal optimal controller and the neural network controller and evaluates the performance of the proposed RNN controller under challenging GCC operating conditions. Finally, the paper concludes with a summary of the main points.

II. Optimal Control of Grid-Connected Converter

A. Grid-Connected Converter Model

Fig. 2 shows the schematic of a GCC, which has a dc-link capacitor on the left and a three-phase voltage source representing the voltage at the Point of Common Coupling (PCC) of the ac system on the right. In the d-q reference frame, the voltage balance across the grid filter is given in Eq. (1), where ω_s is the angular frequency of the grid voltage, and L and R represent the inductance and resistance of the grid filter.

$$\begin{bmatrix} v_d \\ v_q \end{bmatrix} = R\begin{bmatrix} i_d \\ i_q \end{bmatrix} + L\frac{d}{dt}\begin{bmatrix} i_d \\ i_q \end{bmatrix} + \omega_s L\begin{bmatrix} -i_q \\ i_d \end{bmatrix} + \begin{bmatrix} v_{d1} \\ v_{q1} \end{bmatrix} \tag{1}$$

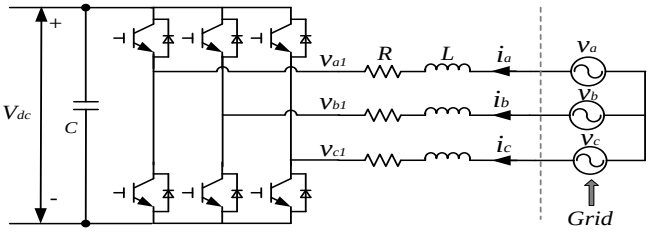


Fig. 2 Grid-connected converter schematic

From Eq. (1), the state-space model of the integrated GCC and grid system can be obtained using Eq. (2), where the system states are i_d and i_q, grid PCC voltages v_d and v_q are normally constant, and converter output voltages v_{d1} and v_{q1} are the control voltages that are to be specified by the output of the controller. For digital control implementation using neural networks, the continuous state-space model of the system in Eq. (2) must be converted to the discrete state-space model represented by Eq. (3), where T_s stands for the sampling period, and k is an integer time step. We used $T_s = 0.001s$ in all experiments. To simplify the expressions, the discrete system model in Eq. (3) is rewritten in Eq. (4), where $\overrightarrow{u_{dq}}(k)$ is represented by Eq. (5).

$$\frac{d}{dt}\begin{bmatrix} i_d \\ i_q \end{bmatrix} = \begin{bmatrix} R/L & -\omega_s \\ \omega_s & R/L \end{bmatrix}\begin{bmatrix} i_d \\ i_q \end{bmatrix} + \frac{1}{L}\begin{bmatrix} v_{d1} \\ v_{q1} \end{bmatrix} - \frac{1}{L}\begin{bmatrix} v_d \\ v_q \end{bmatrix} \tag{2}$$

$$\begin{bmatrix} i_d(kT_s + T_s) \\ i_q(kT_s + T_s) \end{bmatrix} = \mathbf{A}\begin{bmatrix} i_d(kT_s) \\ i_q(kT_s) \end{bmatrix} + \mathbf{B}\begin{bmatrix} v_{d1}(kT_s) - v_d \\ v_{q1}(kT_s) - v_q \end{bmatrix} \tag{3}$$

$$\overrightarrow{i_{dq}}(k+1) = \mathbf{A}\overrightarrow{i_{dq}}(k) + \mathbf{B}\overrightarrow{u_{dq}}(k) \tag{4}$$

$$\overrightarrow{u_{dq}}(k) = \overrightarrow{v_{dq1}}(k) - \overrightarrow{v_{dq}} \tag{5}$$

B. GCC Vector Control

Typically, a GCC has a nested-loop vector control structure consisting of a faster inner current loop and a slower outer loop, as shown in Fig. 3 [4]. In this figure, the d-axis loop is used for active power or dc-link voltage control, and the q-axis loop is used for reactive power or grid voltage support control. The active and reactive power control is converted into decoupled d-q current control, which implements the final control function by applying a voltage signal to the converter [20].

The control signal applied directly to the converter is a three-phase sinusoidal voltage. The general strategy for transforming d-q control signals into three-phase sinusoidal signals is also illustrated in Fig. 3, in which v_{d1}^* and v_{q1}^* are the d- and q-axis output voltages generated by the controller. The two d- and q-axis voltages are converted to the three-phase sinusoidal voltage signals, v_{a1}^*, v_{b1}^* and v_{c1}^*, through Park transformation [21] to control the voltage-source converter. The ratio of the GCC output voltage v_{d1} and v_{q1}, to the output voltage of the current-loop controller v_{d1}^* and v_{q1}^*, is a gain of k_{PWM}, which equals $V_{dc}/2$ if the amplitude of the triangle voltage waveform in the PWM scheme is $1\,V$ [22].

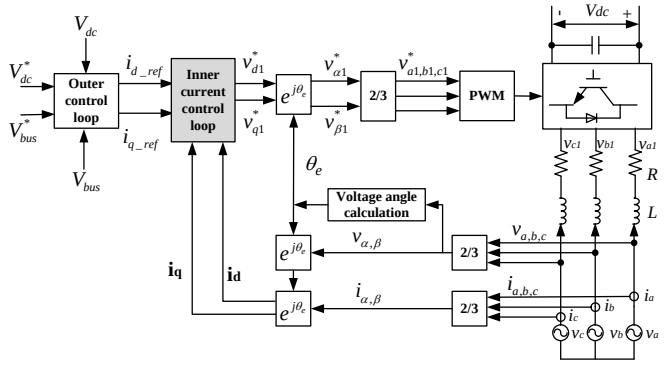


Fig. 3 Standard vector control structure

C. Dynamic Programming in GCC Vector Control

Dynamic programming (DP) employs Bellman's optimality principle [23] for solving optimization and optimal control problems. The typical structure of the discrete-time DP includes a discrete-time system model and a performance index or cost associated with the system [24].

The DP cost function associated with the vector-controlled system is defined as

$$C\left(\overrightarrow{i_{dq}}(j)\right)=\sum_{k=j}^{\infty}\gamma^{k-j}U(\overrightarrow{e_{dq}}(k)),\ j \quad 0,0 \quad \gamma \quad 1 \tag{6}$$

where γ is a discount factor, and U is defined as

$$\begin{aligned} U(\overrightarrow{e_{dq}}(k)) &= \left[e_d^2(k)+e_q^2(k)\right]^{\alpha} \\ &= \left\{\left[i_d(k)-i_{d\_ref}(k)\right]^2+\left[i_q(k)-i_{q\_ref}(k)\right]^2\right\}^{\alpha},\ \alpha>0 \end{aligned} \tag{7}$$

in which α is a constant. The function $C(\cdot)$, which depends on the initial time j and the initial state $\overrightarrow{i_{dq}}(j)$, is referred to as the cost-to-go of state $\overrightarrow{i_{dq}}(j)$ of the DP problem. The objective is to choose a vector control sequence $\overrightarrow{u_{dq}}(k)$ that minimizes the function $C(\cdot)$ in Eq. (6).

D. Ideal Optimal and Suboptimal Vector Control Models

The GCC dynamic model in Eq. (4) is linear, so the ideal optimal control problem can be represented as

$$\min\left(C\right)=0 \Leftrightarrow U(\overrightarrow{e_{dq}}(k))\equiv 0 \Leftrightarrow \overrightarrow{i_{dq}}(k)-\overrightarrow{i_{dq\_ref}}(k)\equiv 0 \tag{8}$$

Then, according to Eq. (4), the optimal control problem can be solved directly by

$$\overrightarrow{u_{dq}}(k)=\mathbf{B}^{-1}\left[\overrightarrow{i_{dq\_ref}}(k+1)-\mathbf{A}\overrightarrow{i_{dq}}(k)\right] \tag{9}$$

where $k=j, j+1,\ldots\infty$. Based on Eq. (5), the control voltage can be obtained by

$$\overrightarrow{v_{dq1}}(k)=\mathbf{B}^{-1}\left[\overrightarrow{i_{dq\_ref}}(k+1)-\mathbf{A}\overrightarrow{i_{dq}}(k)\right]+\overrightarrow{v_{dq}} \tag{10}$$

Furthermore, consider a special case in the steady state in which $\overrightarrow{i_{dq\_ref}}(k+1)=\overrightarrow{i_{dq}}(k)$. Eq. (9) can be simplified as

$$\overrightarrow{u_{dq}}(k)=\mathbf{B}^{-1}\left(\mathbf{I}-\mathbf{A}\right)\overrightarrow{i_{dq}}(k) \tag{11}$$

where $\mathbf{B}^{-1}\left(\mathbf{I}-\mathbf{A}\right)$ is a stabilization matrix proposed in [7] and [25].

The ideal optimal controller in Eq. (10) can regulate the system to catch up with the reference currents in one time step, which is the fastest response time. However, the ideal optimal controller may generate a control voltage signal $\overrightarrow{v_{dq1}}$ beyond the converter's pulse width modulation (PWM) constraint in order to meet the immediate current tracking requirement. To avoid a large control voltage signal, an extension of the one-step ideal optimal controller was also studied and will be discussed later; it is also a special analytical solution to the DP problem. The controller uses a constant control signal $\overrightarrow{u_{dq}}(k)\equiv\overrightarrow{u_{dq}^*}$ to catch up with the reference within L time steps. Then, from Eq. (4), $\overrightarrow{i_{dq}}(k+L)$ can be solved recursively, as shown by

$$\begin{aligned} \overrightarrow{i_{dq}}(k+L) &= \mathbf{A}^L\overrightarrow{i_{dq}}(k)+(\mathbf{A}^{L-1}+\cdots+\mathbf{A}+\mathbf{I})\mathbf{B}\cdot\overrightarrow{u_{dq}^*} \\ &= \mathbf{A}^L\overrightarrow{i_{dq}}(k)+\frac{\mathbf{I}-\mathbf{A}^L}{\mathbf{I}-\mathbf{A}}\mathbf{B}\cdot\overrightarrow{u_{dq}^*} \end{aligned} \tag{12}$$

Hence, the required $\overrightarrow{u_{dq}^*}$ can be found as follows:

$$\overrightarrow{u_{dq}^*}=\left[\frac{\mathbf{I}-\mathbf{A}^L}{\mathbf{I}-\mathbf{A}}\mathbf{B}\right]^{-1}\left[\overrightarrow{i_{dq\_ref}}(k+L)-\mathbf{A}^L\overrightarrow{i_{dq}}(k)\right] \tag{13}$$

Based on Eq. (5), the control voltage can be obtained as follows:

$$\overrightarrow{v_{dq1}}(k)=\left[\frac{\mathbf{I}-\mathbf{A}^L}{\mathbf{I}-\mathbf{A}}\mathbf{B}\right]^{-1}\left[\overrightarrow{i_{dq\_ref}}(k \quad L) \quad \mathbf{A}^L\overrightarrow{i_{dq}}(k)\right] \quad \overrightarrow{v_{dq}} \tag{14}$$

Eq. (15) proves that Eq. (9) is the limit form of Eq. (13):

$$\lim_{L\to 1}\overrightarrow{u_{dq}^*}=\left[\frac{\mathbf{I}-\mathbf{A}^1}{\mathbf{I}-\mathbf{A}}\mathbf{B}\right]^{-1}\left[\overrightarrow{i_{dq\_ref}}(k \quad L) \quad \mathbf{A}^1\overrightarrow{i_{dq}}(k)\right] \quad \overrightarrow{u_{dq}}(k) \tag{15}$$

Furthermore, Eq. (10) is the limit form of Eq. (14). Therefore, Eq. (14) can be considered a suboptimal controller ($L>1$) in the sense that the controller's response speed is determined by time step L. After $\overrightarrow{i_{dq}}$ catches up with $\overrightarrow{i_{dq\_ref}}$, that is, when it reaches a steady state, Eq. (11) is applied to control the system.

Fig. 4 illustrates an example of the ideal optimal controller, as specified in Eq. (10), for the GCC system. The figure reveals that the ideal optimal controller exhibits perfect tracking, the fastest response without any delay, no overshoot,

and no steady-state error. Fig. 5 demonstrates the suboptimal controller, Eq. (14), for GCC control, with L equal to 1, 3, 5, 7, and 9, respectively. As shown in Fig. 5, the controller can follow the reference current over exactly L time steps.

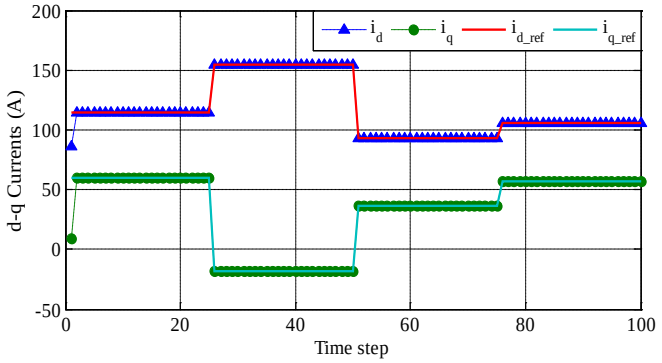


Fig. 4 Ideal optimal controller for GCC

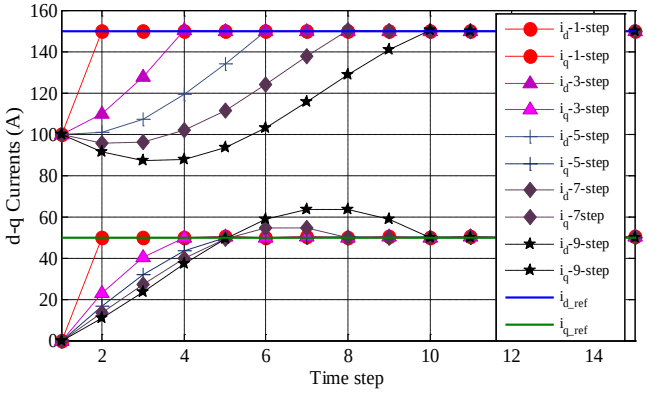


Fig. 5 GCC suboptimal controller in L steps

III. Neural Network Vector Controller and Proposed FATT-LM Training Algorithm

A. GCC Neural Network Vector Controller

The ideal optimal controller, Eq. (10), and suboptimal controller, Eq. (14), were deduced under the assumption that the exact GCC system parameters were known. In practice, the system parameters may deviate significantly from its nominal values. Particularly, the inductance of the grid filter could be affected by the temperature and grid voltage frequency. Changes in the system parameters will affect the performance of both the ideal optimal and suboptimal controllers. Thus, these controllers are not robust in practice. Therefore, a RNN vector controller is employed to approximate the ideal optimal controller.

Fig. 7 depicts the overall RNN vector-control structure of the GCC current-loop, which combines the vector control technique with the DP-based neural network design. The neural network component shown in Fig. 7 is a fully connected multi-layer perceptron [26] with 2 hidden layers having 6 nodes each, and 2 output nodes, with hyperbolic tangent functions at all nodes, as detailed in Fig. 6.

To avoid neural network input saturation, the inputs are regulated to the range [-1, 1] using the hyperbolic tangent function, as shown in Fig. 6. The first 4 input nodes are $\tanh(\overrightarrow{e_{dq}} / Gain)$ and $\tanh(\overrightarrow{s_{dq}} / Gain2)$, where

$$\overrightarrow{e_{dq}}(k) = \overrightarrow{i_{dq}}(k) - \overrightarrow{i_{dq\_ref}}(k) \quad (16)$$

is referred to as the "error input term," and

$$\overrightarrow{s_{dq}}(k) = \int_0^{kT_s} \overrightarrow{e_{dq}}(t)dt \quad (17)$$

is referred to as the "integral term." Unlike [6], this paper also proposes an RNN controller that achieves an improved input structure by using two error terms and two integral terms as the network inputs, i.e., removing the d-q current inputs indicated by the blue dashed lines in Fig. 7. This network input scheme is particularly important when some network states cannot be measured, such as the rotor currents in the control of a squirrel-cage induction motor. This improvement also reduces the number of weights between the input layer and the first hidden layer and requires less calculation effort in the real control loop.

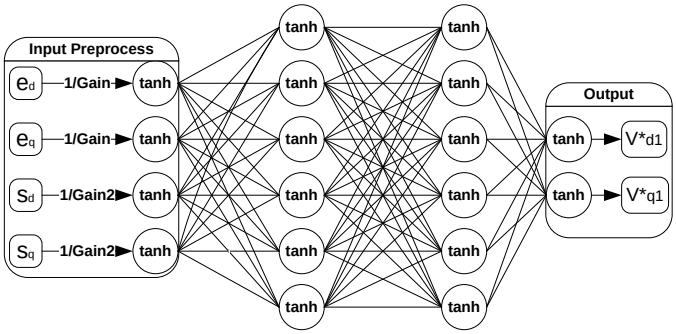


Fig. 6 RNN controller structure

B. Define RNN controller function

Based on the proposed RNN controller structure, the RNN can be denoted as $R(\overrightarrow{e_{dq}}(k), \overrightarrow{s_{dq}}(k), \vec{w})$, which is a function of $\overrightarrow{e_{dq}}(k)$, $\overrightarrow{s_{dq}}(k)$ and $\vec{w}$. If the RNN takes $\overrightarrow{i_{dq}}(k)$ as inputs, the function $R(\cdot)$ can also be denoted as $R(\overrightarrow{i_{dq}}(k), \overrightarrow{e_{dq}}(k), \overrightarrow{s_{dq}}(k), \vec{w})$. Thus, the control action $\overrightarrow{u_{dq}}(k)$ is expressed by

$$\overrightarrow{u_{dq}}(k) = \overrightarrow{v_{dq1}}(k) - \overrightarrow{v_{dq}} = k_{PWM} R(\overrightarrow{e_{dq}}(k), \overrightarrow{s_{dq}}(k), \vec{w}) \quad \overrightarrow{v_{dq}} \quad (18)$$

where k_{PWM} is the PWM gain, as explained in Section II.B.

The converter output voltages v_{d1} and v_{q1} are proportional to the control voltage of the RNN output, as explained in Section II.B. Although Fig. 6 shows a feed-forward network configuration, the controller is considered a recurrent network because the feedback signal generated by the system in Eq. (5) acts as a recurrent network connection from the output of the system shown in Fig. 7 back to the input.

C. Backpropagation Through Time (BPTT)

Before defining the main algorithm of this paper, i.e., LM+FATT, we first review the BPTT method used in [6] for this GCC problem for the purpose of comparison, as discussed in Section IV.

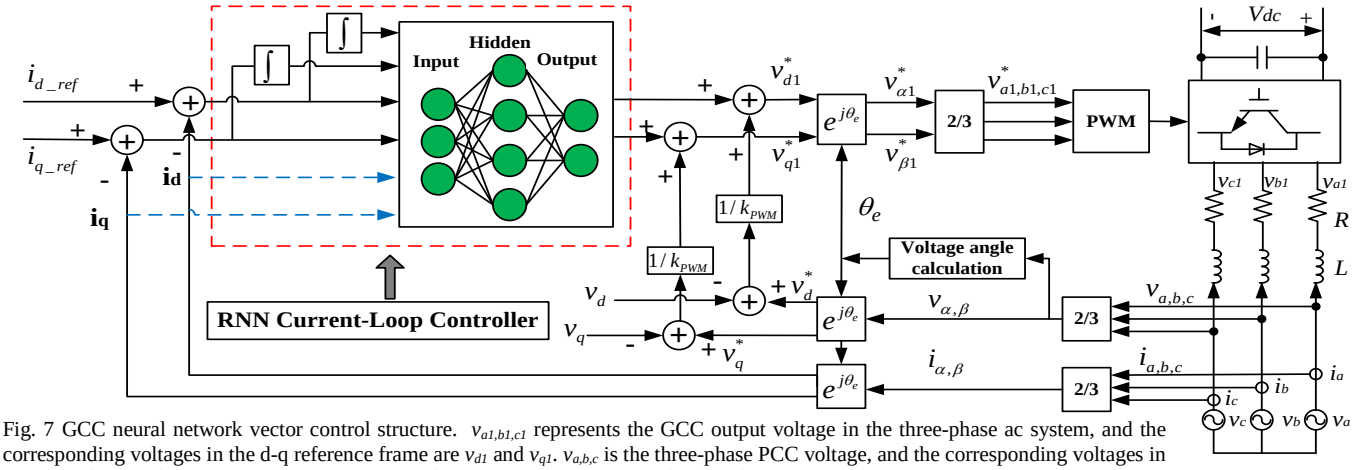


Fig. 7 GCC neural network vector control structure. $v_{a1,b1,c1}$ represents the GCC output voltage in the three-phase ac system, and the corresponding voltages in the d-q reference frame are v_{d1} and v_{q1}. $v_{a,b,c}$ is the three-phase PCC voltage, and the corresponding voltages in the d-q reference frame are v_d and v_q. $i_{a,b,c}$ stands for the three-phase current flowing from the PCC to the GCC, and the corresponding currents in the d-q reference frame are i_d and i_q. v_{d1}^* and v_{q1}^* are the d- and q-axis voltages from the RNN controller, and the corresponding control voltage in the three-phase domain is $v_{a1,b1,c1}$.

Algorithm 1: BPTT algorithm for training a GCC RNN vector controller

1: $C \leftarrow 0, \overrightarrow{e_{dq}}(0) \leftarrow \vec{0}, \overrightarrow{s_{dq}}(0) \leftarrow \vec{0},$

2: {Unroll a full trajectory}

3: **for** k = 0 to N-1 **do**

4: $\overrightarrow{v_{dq1}}(k) \leftarrow k_{PWM} R\left(\overrightarrow{i_{dq}}(k), \overrightarrow{e_{dq}}(k), \overrightarrow{s_{dq}}(k), \vec{w}\right)$

5: $\overrightarrow{i_{dq}}(k+1) \leftarrow \mathbf{A}\overrightarrow{i_{dq}}(k) + \mathbf{B}\left[\overrightarrow{v_{dq1}}(k) - \overrightarrow{v_{dq}}\right]$

6: $\overrightarrow{e_{dq}}(k+1) \leftarrow \overrightarrow{i_{dq}}(k+1) - \overrightarrow{i_{dq\_ref}}(k+1)$

7: $\overrightarrow{s_{dq}}(k+1) \leftarrow \overrightarrow{s_{dq}}(k) + \frac{T_s}{2}\left(\overrightarrow{e_{dq}}(k+1) + \overrightarrow{e_{dq}}(k)\right)$

8: $C \leftarrow C + \gamma^k U(\overrightarrow{e_{dq}}(k+1))$

9: **end for**

10: {Backward pass along trajectory}

11: $\frac{\partial C}{\partial \vec{w}} \leftarrow 0, \quad \frac{\partial C}{\partial \overrightarrow{i_{dq}}(N)} \leftarrow 0, \quad \frac{\partial C}{\partial \overrightarrow{s_{dq}}(N)} \leftarrow 0$

12: **for** $k = N\text{-}1$ to 0 step -1 **do**

13: $\frac{\partial C}{\partial \overrightarrow{v_{dq1}}(k)} \leftarrow \mathbf{B}^T \frac{\partial C}{\partial \overrightarrow{i_{dq}}(k+1)}$

14: $\frac{\partial C}{\partial \overrightarrow{s_{dq}}(k)} \leftarrow \frac{\partial C}{\partial \overrightarrow{s_{dq}}(k+1)} + k_{PWM} \frac{\partial R(k)}{\partial \overrightarrow{s_{dq}}(k)} \frac{\partial C}{\partial \overrightarrow{v_{dq1}}(k)}$

15: $\frac{\partial C}{\partial \overrightarrow{i_{dq}}(k)} \leftarrow k_{PWM} \frac{\partial R(k)}{\partial \overrightarrow{i_{dq}}(k)} \frac{\partial C}{\partial \overrightarrow{v_{dq1}}(k)} + \mathbf{A}^T \frac{\partial C}{\partial \overrightarrow{i_{dq}}(k+1)} + \frac{T_s}{2}\left[\frac{\partial C}{\partial \overrightarrow{s_{dq}}(k+1)} + \frac{\partial C}{\partial \overrightarrow{s_{dq}}(k)}\right] + \gamma^k \frac{\partial U(\overrightarrow{e_{dq}}(k))}{\partial \overrightarrow{i_{dq}}(k)}$

16: $\frac{\partial C}{\partial \vec{w}} \leftarrow \frac{\partial C}{\partial \vec{w}} + k_{PWM} \frac{\partial R(k)}{\partial \vec{w}(k)} \frac{\partial C}{\partial \overrightarrow{v_{dq1}}(k)}$

17: **end for**

18: {on exit, $\frac{\partial C}{\partial \vec{w}}$ holds for the whole trajectory}

BPTT is gradient descent on $C\left(\overrightarrow{i_{dq}}(j), \vec{w}\right)$ with respect to the weight vector of the recurrent neural network. In general, the BPTT algorithm consists of two steps, a forward pass that unrolls a trajectory, followed by a backward pass along the whole trajectory, which accumulates the gradient descent derivative.

Alg. 1 provides pseudo code for both stages of this process. Lines 1-9 evaluate a trajectory of length N using Eqs. (4) and (5). The second half of the algorithm, from lines 10-18, calculates the desired gradient $\partial C / \partial \vec{w}$. The derivation of the gradient computation part of the algorithm (lines 11-18) is exact and follows the method detailed in [27], which is referred to as generalized backpropagation [28], or automatic differentiation [29]. The derivatives $\partial R(k) / \partial \vec{w}$, $\partial R(k) / \partial \overrightarrow{i_{dq}}(k)$, and $\partial R(k) / \partial \overrightarrow{s_{dq}}(k)$ are calculated according to the standard neural network back-propagation rule [30], which is basically the chain rule for computing the derivatives of the composition of two or more functions.

During the evaluation of the trajectory forward pass (lines 3-9 of Alg. 1), and in all subsequent pseudo code in this paper, the integral input term of Eq. (17) was evaluated using the following trapezoid formula in Eq. (19), instead of the forward Euler formula in [6]:

$$\overrightarrow{s_{dq}}(k) \approx T_s \sum_{j=1}^{k} \frac{\overrightarrow{e_{dq}}(j-1) + \overrightarrow{e_{dq}}(j)}{2}, \quad \overrightarrow{e_{dq}}(0) = \vec{0} \qquad (19)$$

This approximation can also be rewritten as a recurrence relation:

$$\overrightarrow{s_{dq}}(k) = \overrightarrow{s_{dq}}(k-1) + T_s \frac{\overrightarrow{e_{dq}}(k-1) + \overrightarrow{e_{dq}}(k)}{2}, \overrightarrow{s_{dq}}(0) \quad \vec{0} \qquad (20)$$

Hence, Eq. (20) appears as line 7 of Alg. 1.

D. Levenberg-Marquardt algorithm, and its applicability to RNNs

The Levenberg-Marquardt (LM) algorithm is widely used to train feed-forward networks and provides a nice compromise between the speed of Newton's method and the guaranteed convergence of the steepest descent. Thus, LM appears to be the fastest neural network training algorithm for

a moderate number of network parameters [30]. Although LM has achieved great success in training feed-forward networks, it is not sufficiently straightforward for use in training RNNs directly. This paper develops a mechanism by which to train a recurrent network based on LM, which can be applied to any problem in which the RNN has fixed target outputs at each time step, such as the GCC tracking control problem.

LM can minimize a non-linear sum-of-squares cost function, such as in a case in which the cost function can be written as $C(\vec{w}) = \sum_p V(p)^2$, where p is the training "pattern" index, and $V(p)$ is the error for pattern p. Then, the LM algorithm consists of the following weight update [30]:

$$\Delta\vec{w} = -\left[J(\vec{w})^T J(\vec{w}) + \mu \mathbf{I} \right]^{-1} J(\vec{w})^T \vec{V} \tag{21}$$

where $J(\vec{w})$ is the Jacobian matrix of $\vec{V}$ with respect to the weight vector of the neural network, $\vec{V}$ is defined by

$$\vec{V} = \begin{bmatrix} V(1) \\ \vdots \\ V(N) \end{bmatrix} \tag{22}$$

where $\mathbf{I}$ is the identity matrix, and μ is a scalar regulation parameter that is dynamically adjusted during learning (see Fig. 8). The quantity $J(\vec{w})^T J(\vec{w})$ is called the Gauss-Newton matrix, and it approximates the Hessian matrix for the cost function [31]. Hence, LM approximates the Newton method in solving the cost-minimization problem.

To extend the applicability of LM to the RNN case, we simply define $V(k)$ as the error of the output of the RNN at time step k. Hence, if there are N time steps in the state trajectory and M weights in the neural network, then the Jacobian matrix $J(\vec{w})$ is defined for a RNN as

$$J(\vec{w}) = \begin{bmatrix} \frac{\partial V(1)}{\partial w_1} & \cdots & \frac{\partial V(1)}{\partial w_M} \\ \vdots & \ddots & \vdots \\ \frac{\partial V(N)}{\partial w_1} & \cdots & \frac{\partial V(N)}{\partial w_M} \end{bmatrix} \tag{23}$$

This completes the definition that extends LM for applicability to RNNs. Derivatives of the form $\partial V(k)/\partial w_i$ will be dependent upon $\partial V(j)/\partial w_i$ for $j < i$ (by the chain rule); therefore, the Jacobian matrix must be calculated with care. The FATT algorithm presented in Section III.F shows the correct way to perform this calculation.

E. Levenberg-Marquardt algorithm for non-sum-of-squares cost functions

If the performance error function is not a sum of squares, then the LM weight update equation (Eq. (21)) is not directly applicable.

In the definition of the cost function $C(\cdot)$ and the local cost $U(\cdot)$, there is a constant number α. When $\alpha = 1$, the cost function is just the sum of squares of errors. In such cases, LM can be applied directly to train the RNN. However, the selection of the constant number α has an important impact on RNN training. Better convergence in RNN training can be achieved when α is a fractional number, as demonstrated in Section IV.

In order to use LM for any α values, the cost function $C(\cdot)$ defined in Eq. (6) must be modified. Consider the cost function $C = \sum_{k=j}^{\infty} \gamma^{k-j} U(\vec{e}_{dq}(k))$ in which $\gamma = 1$, $j = 1$, and $k = 1, \ldots, N$. Then, $C(\cdot)$ can be written as

$$C = \sum_{k=1}^{N} U(\vec{e}_{dq}(k)) \xrightarrow{\text{define } V(k) = \sqrt{U(\vec{e}_{dq}(k))}} C \quad \sum_{k=1}^{N} (V(k))^2 \tag{24}$$

and the gradient $\partial C / \partial \vec{w}$ can be written in a matrix form as

$$\frac{\partial C}{\partial \vec{w}} = \frac{\partial \sum_{k=1}^{N} (V(k))^2}{\partial \vec{w}} \quad \sum_{k=1}^{N} 2V(k) \frac{\partial V(k)}{\partial \vec{w}} \quad 2J(\vec{w})^T \vec{V} \tag{25}$$

F. Forward Accumulation Through Time (FATT)

In order to calculate the Jacobian matrix $J(\vec{w})$ for a RNN efficiently, a new algorithm, Forward Accumulation Through Time (FATT), is proposed. FATT combines the computation of the Jacobian matrix $J(\vec{w})$ and the unrolling of the system trajectory, and calculates both system states $\vec{i}_{dq}(k)$ and $\partial V(k)/\partial \vec{w}$, the k-th row of the Jacobian matrix $J(\vec{w})$, at time step $t = kT_s$. The following proposed FATT is suitable for a general constant α.

To find the k-th row of the Jacobian matrix $J(\vec{w})$, the derivative $\partial V(k)/\partial \vec{w}$ is expressed as

$$\frac{\partial V(k)}{\partial \vec{w}} = \frac{\partial V(k)}{\partial \vec{e}_{dq}(k)} \frac{\partial \vec{e}_{dq}(k)}{\partial \vec{w}} \tag{26}$$

According to the definition of $V(k)$ in Eq. (24),

$$\begin{aligned} \frac{\partial V(k)}{\partial \vec{e}_{dq}(k)} &= \frac{\partial}{\partial \vec{e}_{dq}(k)} \left[\left(e_d^2(k) \quad e_q^2(k) \right)^{\frac{\alpha}{2}} \right] \\ &= \alpha \left(e_d^2(k) + e_q^2(k) \right)^{\frac{\alpha}{2}-1} \left[e_d(k) \quad e_q(k) \right] \end{aligned} \tag{27}$$

Differentiating $\vec{e}_{dq}(k)$ of Eq. (16) yields:

$$\frac{\partial \vec{e}_{dq}(k)}{\partial \vec{w}} = \frac{\partial \vec{i}_{dq}(k)}{\partial \vec{w}} \tag{28}$$

The derivative $\partial \vec{i}_{dq}(k+1)/\partial \vec{w}$ is found based on the recursive formula in Eq. (4) and the definition of the RNN controller in Eq. (18):

$$\frac{\partial \vec{i}_{dq}(k+1)}{\partial \vec{w}} = \mathbf{A} \frac{\partial \vec{i}_{dq}(k)}{\partial \vec{w}} + \mathbf{B} \frac{\partial \vec{u}_{dq}(k)}{\partial \vec{w}} \tag{29}$$

Differentiating Eq. (18) and utilizing Eq. (28) yields

$$\frac{\partial \vec{u}_{dq}(k)}{\partial \vec{w}} = k_{PWM} \left(\frac{\partial R(k)}{\partial \vec{e}_{dq}(k)} \frac{\partial \vec{i}_{dq}(k)}{\partial \vec{w}} + \frac{\partial R(k)}{\partial \vec{s}_{dq}(k)} \frac{\partial \vec{s}_{dq}(k)}{\partial \vec{w}} + \frac{\partial R(k)}{\partial \vec{w}} \right) \tag{30}$$

where $\partial\overrightarrow{s_{dq}}(k)/\partial\vec{w}$ is calculated according to Eq. (19)

$$\begin{aligned}\frac{\partial\overrightarrow{s_{dq}}(k)}{\partial\vec{w}} &= +\frac{T_s}{2}\sum_{j=1}^{k}\left(\frac{\partial\overrightarrow{e_{dq}}(k-1)}{\partial\vec{w}} \quad \frac{\partial\overrightarrow{e_{dq}}(k)}{\partial\vec{w}}\right) \\ &= T_s\left(\sum_{j=0}^{k}\frac{\partial\overrightarrow{i_{dq}}(j)}{\partial\vec{w}} \quad \frac{1}{2}\frac{\partial\overrightarrow{i_{dq}}(k)}{\partial\vec{w}}\right)\end{aligned} \tag{31}$$

Calculating $\partial\overrightarrow{i_{dq}}(k)\big/\partial\vec{w}$ requires a loop from $j=0$ to $j=k$. Thus, the process for calculating $\partial\overrightarrow{i_{dq}}(k)\big/\partial\vec{w}$ and $\partial V(k)\big/\partial\vec{w}$ is integrated into the process of unrolling the trajectory.

Algorithm 2: FATT algorithm to calculate the Jacobian matrix.

1: $C \leftarrow 0, \overrightarrow{e_{dq}}(0) \leftarrow \vec{0}, \overrightarrow{s_{dq}}(0) \leftarrow \vec{0}, \frac{\partial\overrightarrow{i_{dq}}(0)}{\partial\vec{w}} \leftarrow \vec{0}, \frac{\partial\vec{\varphi}(0)}{\partial\vec{w}} \leftarrow \vec{0}$

2: {Calculate Jacobian matrix $J(\vec{w})$ }

3: **for** k = 0 to N-1 **do**

4: $\overrightarrow{u_{dq}}(k) \leftarrow k_{PWM} R(\overrightarrow{e_{dq}}(k), \overrightarrow{s_{dq}}(k), \vec{w}) - \overrightarrow{v_{dq}}$

5: $\frac{\partial\overrightarrow{s_{dq}}(k)}{\partial\vec{w}} \leftarrow T_s\left(\frac{\partial\vec{\varphi}(k)}{\partial\vec{w}} - \frac{1}{2}\frac{\partial\overrightarrow{i_{dq}}(k)}{\partial\vec{w}}\right)$

6: $\frac{\partial\overrightarrow{u_{dq}}(k)}{\partial\vec{w}} \leftarrow k_{PWM}\left(\frac{\partial R(k)}{\partial\overrightarrow{e_{dq}}(k)}\frac{\partial\overrightarrow{i_{dq}}(k)}{\partial\vec{w}} + \frac{\partial R(k)}{\partial\overrightarrow{s_{dq}}(k)}\frac{\partial\overrightarrow{s_{dq}}(k)}{\partial\vec{w}} + \frac{\partial R(k)}{\partial\vec{w}}\right)$

7: $\frac{\partial\overrightarrow{i_{dq}}(k+1)}{\partial\vec{w}} \leftarrow \mathbf{A}\frac{\partial\overrightarrow{i_{dq}}(k)}{\partial\vec{w}} + \mathbf{B}\frac{\partial\overrightarrow{u_{dq}}(k)}{\partial\vec{w}}$

8: $\overrightarrow{i_{dq}}(k+1) \leftarrow \mathbf{A}\overrightarrow{i_{dq}}(k) + \mathbf{B}\overrightarrow{u_{dq}}(k)$

9: $\overrightarrow{e_{dq}}(k+1) \leftarrow \overrightarrow{i_{dq}}(k+1) - \overrightarrow{i_{dq\_ref}}(k+1)$

10: $\overrightarrow{s_{dq}}(k+1) \leftarrow \overrightarrow{s_{dq}}(k) + \frac{T_s}{2}\left(\overrightarrow{e_{dq}}(k+1) + \overrightarrow{e_{dq}}(k)\right)$

11: $C \leftarrow C + U(\overrightarrow{e_{dq}}(k+1))$

12: $\frac{\partial\vec{\varphi}(k+1)}{\partial\vec{w}} \leftarrow \frac{\partial\vec{\varphi}(k)}{\partial\vec{w}} + \frac{\partial\overrightarrow{i_{dq}}(k+1)}{\partial\vec{w}}$

13: $\frac{\partial V(k+1)}{\partial\vec{w}} \leftarrow \frac{\partial V(k+1)}{\partial\overrightarrow{e_{dq}}(k+1)}\frac{\partial\overrightarrow{i_{dq}}(k+1)}{\partial\vec{w}}$

14: the $(k+1)$th row of $J(\vec{w}) \leftarrow \frac{\partial V(k+1)}{\partial\vec{w}}$

15: **end for**

16: {on exit, the Jacobian matrix $J(\vec{w})$ is finished for the whole trajectory}

Alg. 2 shows the entire FATT process for calculating the Jacobian matrix $J(\vec{w})$ for a complete trajectory. In the algorithm, $\vec{\varphi}(k) = \sum_{j=1}^{k}\overrightarrow{i_{dq}}(j)$ and $\frac{\partial\vec{\varphi}(k)}{\partial\vec{w}} = \sum_{j=1}^{k}\frac{\partial\overrightarrow{i_{dq}}(j)}{\partial\vec{w}} = \frac{\partial\vec{\varphi}(k-1)}{\partial\vec{w}} + \frac{\partial\overrightarrow{i_{dq}}(k)}{\partial\vec{w}}$. Lines 5, 6, 7 and 13 of Alg. 2 come from Eqs. (31), (30), (29) and (26), respectively.

The proposed FATT algorithm also can be applied to develop neural network vector controllers of other dynamic systems. To utilize FATT on another dynamic system, the new state-space model of the system must be incorporated into the FATT algorithm (Alg. 2), i.e., only those formulas that are related to the system's state-space model need to be modified. This shows the generalizability and broad scope of the potential impact of our novel approach.

G. Computational complexity study

In Alg. 2, N denotes the trajectory length. The most time-consuming part of Alg. 2 is the matrix multiplications that involve m by M dimensional matrices, where M stands for the number of all the weights, and m represents the dimension of the RNN output layer, which is also the dimension of the state vector $\overrightarrow{i_{dq}}$, i.e., $m=2$. For example, line 6 of Alg. 2 involves matrix multiplication between an $m\times m$ matrix $\partial R(k)/\partial\overrightarrow{e_{dq}}(k)$ and an $m\times M$ matrix $\partial\overrightarrow{i_{dq}}(k)/\partial\vec{w}$. Using standard matrix multiplication, this will take m^2M floating-point operations (flops). Thus, combining it into the loop of N time steps for the entire length of the trajectory, the FATT algorithm takes $O(m^2NM)$ flops to complete $J(\vec{w})$.

H. Training algorithm for RNN: LM plus FATT

Fig. 8 illustrates the entire process for incorporating FATT into LM. In Fig. 8, $\mu_{\max}$ stands for the maximum acceptable μ, β_{de} and β_{in} signify the decreasing and increasing factors, respectively, that are used to adjust the learning rate during the training, $\text{Epoch}_{\max}$ represents the maximum number of training epochs, and $\left\|\partial C/\partial\vec{w}\right\|_{\min}$ denotes the norm of the minimum acceptable gradient.

Besides calculating the Jacobian matrix, FATT can also output the DP cost, as shown in line 11 of Alg. 2. FATT* in Fig. 8 refers to the process for calculating the DP cost, which involves limiting the running of lines 5-7 and 12-13 in Alg. 2 to save calculation time. The standard LM procedure was followed during the training in [16] and [30] (Fig. 8). In order to accelerate the calculation, the weights in Eq. (21) were updated using Cholesky factorization, which is roughly twice as efficient as LU decomposition for solving systems of linear equations [32].

The procedure for adjusting μ also appears in Fig. 8. The parameter μ is dynamically adjusted to ensure that the training follows the decreasing direction of the DP cost function. When μ increases, it approaches the steepest descent algorithm with a small learning rate; when μ decreases, the algorithm approaches Gauss-Newton, which provides faster convergence. The following three training stopping conditions used are : 1) when the training epoch reaches the maximum number of training epochs, $\text{Epoch}_{\max}$, 2) when μ is larger

than $\mu_{\max}$, and 3) when the gradient is smaller than the predefined minimum gradient $\left\|\partial C / \partial \vec{w}\right\|_{\min}$.

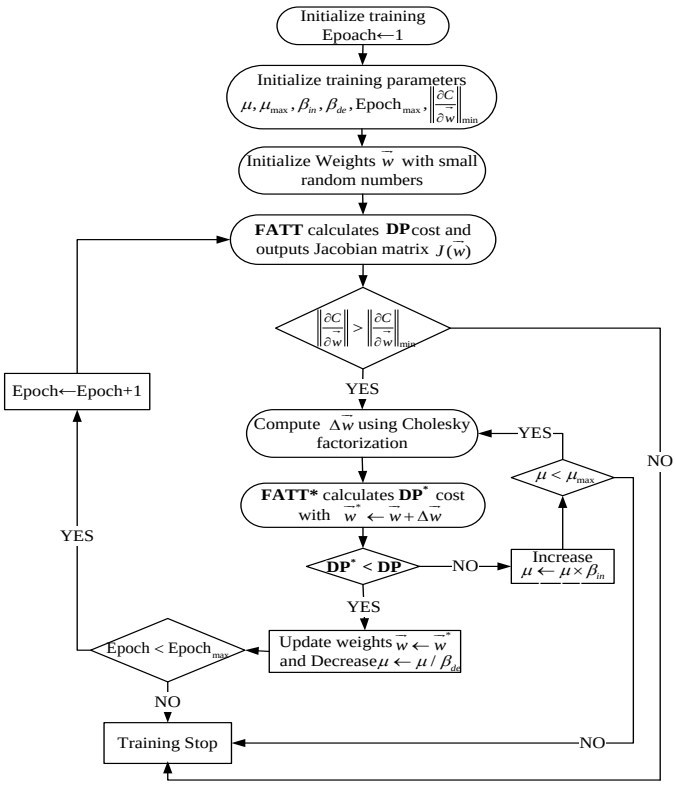


Fig. 8 Training algorithm for RNN: LM+FATT

I. Off-line training vs on-line training

The proposed FATT algorithm for calculating the Jacobian matrix does not require backward computing. Thus, it is suitable for on-line training and also can be combined with other algorithms, such as RPROP, for on-line learning. To compute the gradient $\partial V(k+1)/\partial \vec{w}$ at each time step, FATT can yield the needed gradient as the system propagates.

LM is not appropriate for on-line learning because each training epoch may contain several iterations; thus, LM + FATT training in this paper was conducted off-line. The neural network trained off-line with the proposed LM + FATT demonstrated great performance under variable system parameters and noise conditions, as demonstrated in Section V.

IV. Comparison of FATT-LM and BPTT in Training an RNN Controller

To train the RNN controller, data for a typical GCC in renewable energy conversion applications were specified [33], [34], and [35]. These included: 1) a three-phase 60Hz, 690V voltage source signifying the grid, 2) a reference voltage of 1200V for the dc link, and 3) a resistance of 0.012Ω and an inductance of 2mH for the grid filter. Training took the following policies: 1) randomly generating a sample initial state $\vec{i}_{dq}(0)$, 2) randomly generating a sample reference d-q current trajectory with consideration of the rated current and PWM saturation constraints [35] and [5], 3) selecting the sampling time as $T_s = 1ms$ and the entire duration of training as 1 second, and 4) randomly generating initial network weights using a Gaussian distribution with zero means and 0.1 variances.

A. FATT is equivalent to BPTT

To verify the proposed algorithm, the gradients generated using FATT (Eq. (25)) were compared with those computed using BPTT (Alg. 1). Table I shows the comparison results, which demonstrates the equivalence of the two methods. The matrix W3 in Table I denotes the weights of the output layer of the RNN and its dimension, 7x2. The mean square error (MSE) for calculated weights W3 is 2.7043×10^{-13} with a 95% confidence interval [1.3460×10^{-13}, 6.2458×10^{-13}]. For all calculated neural network weights, the total MSE value is 4.4377×10^{-14} with a 95% confidence interval [3.3819×10^{-14}, 5.9372×10^{-14}]. The MSE was calculated using 32-bit MATLAB with double precision. Thus, the gradients calculated using FATT and BPTT are basically the same; their differences are limited to rounding errors.

TABLE I
GRADIENT COMPARISON BETWEEN FATT AND BPTT

W3 (1.0e+009)			
FATT		**BPTT**	
0.092796562629715	0.020751225861687	0.092796562629715	0.020751225861687
0.229650202454692	0.020683288705771	0.229650202454692	0.020683288705771
0.175811456841742	0.015729812462136	0.175811456841742	0.015729812462136
0.126106210024035	0.010156322474025	0.126106210024035	0.010156322474025
0.144791337207981	0.013380664777951	0.144791337207981	0.013380664777951
0.24191071040987**5**	0.013627701859040	0.24191071040987**4**	0.013627701859040
2.16869114299510**10**	0.039567808952663	2.16869114299510**00**	0.039567808952663

B. RNN training algorithm comparison: LM+FATT and BPTT+RPROP

Fig. 9 shows the average DP cost per trajectory time step for training the neural network vector controller using FATT+LM and BPTT+RPROP, respectively. The training of both algorithms uses the same parameters, including the same initial weights, starting d-q currents, and d-q reference currents.

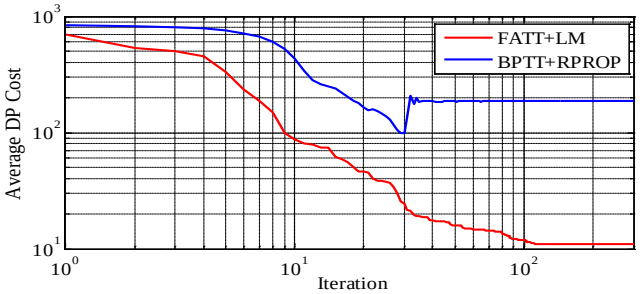


Fig. 9 Comparison of average DP cost per trajectory time step for training RNN controller using FATT+LM and FATT+RPROP

In theory, LM is faster than RPROP (in [36]), as verified by Fig. 9. The overall average trajectory cost decreased to a small number much faster using LM than using RPROP, demonstrating the excellent learning ability of the proposed FATT algorithm with LM for training the RNN controller. The results also revealed a major drawback of RPROP, an

oscillation problem during the training, as illustrated in Fig. 9, which may cause the training to get stuck at a high average DP cost level.

C. Different constant α effects in training a RNN

Fig. 10 compares the average DP cost for training the recurrent network with different α values using FATT+LM. For $\alpha = 1$, we used the square roots of the average DP cost in Fig. 10 in order to compare the average DP cost corresponding to $\alpha = 1/2$. Fig. 10 shows that $\alpha = 1/2$ yielded better and faster convergence. When $\alpha = 1$, the training had difficulty converging to the required result. This also indicates the necessity of transformation Eq. (24). Directly using the MSE as a RNN training objective, which is the general case for feed-forward neural networks, does not work well.

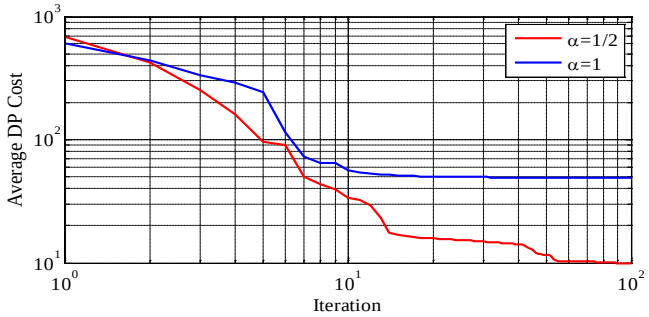


Fig. 10 Comparison of average DP cost per trajectory time step with different α values using FATT+LM

D. Comparison of RNN controllers with different input structures

Fig. 11 compares the average DP cost for training the recurrent network using the FATT+LM when 1) all six inputs, including $\vec{i}_{dq}$, indicated by the blue dashed lines in Fig. 7, are used as the network inputs, and 2) only two error terms and two integral terms are used as the network inputs, as in Fig. 6.

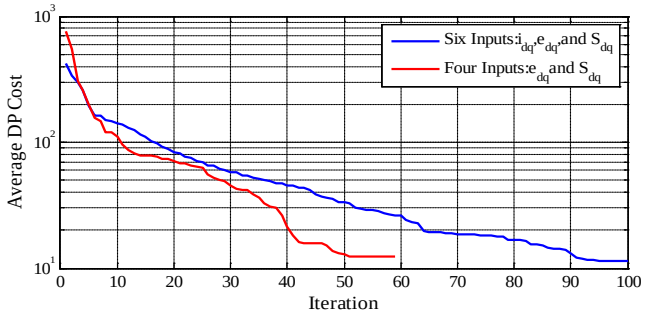


Fig. 11 Average DP cost per trajectory time step for training RNN controller with two different network input schemes using FATT+LM

As illustrated in Fig. 11, with the RNN input structure shown in Fig. 6, the training actually converged to good performance faster. In addition, reducing the RNN input variables reduces the calculation efforts needed in real-time control, which allows the proposed RNN controller to be applied easily in hardware. This optimized RNN controller would make it more practical to develop neural network vector controllers for many other power and energy system applications, making RNN controllers a reality.

E. Comparison of optimal controller and RNN controller

Fig. 12 compares the performance of an ideal optimal controller, a suboptimal controller and a RNN controller. The RNN controller used for this comparison was trained sufficiently well when the average DP cost per trajectory dropped to a small value and stabilized there, as indicated by Fig. 11. Basically no steady-state error existed for the RNN controller after the twentieth time step, according to Fig. 12. Compared to the 20-step suboptimal controller, the RNN controller had a smaller overshoot. Fig. 12 also indicates that the RNN controller properly approximated the ideal optimal controller.

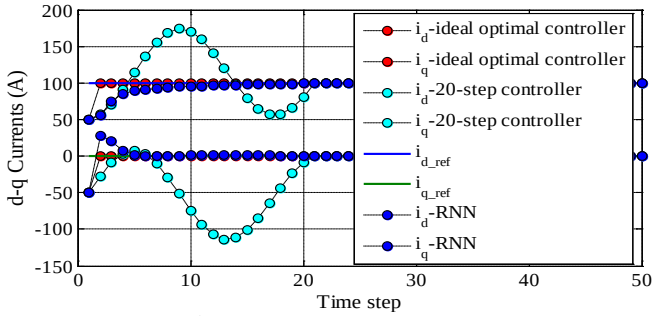


Fig. 12 Controller performance comparison between ideal optimal controller, 20-step controller and RNN controller.

V. Performance Evaluation of Trained Neural Network Vector Controller

To evaluate the performance of the neural network vector controller trained with the proposed FATT+LM algorithm, and to compare the performance of the ideal optimal controller and the neural network vector controller, an integrated transient simulation system of a GCC system was developed using SimPowerSystems (Fig. 13). The converter switching frequency was 3000Hz. In the switching environment, the performance was evaluated under close to real-life conditions [35]. The PCC bus was connected to the grid through a transmission line modeled by an impedance. For digital control implementation, the measured instantaneous three-phase PCC voltage and grid current passed through a zero-order-hold (ZOH) block [37].

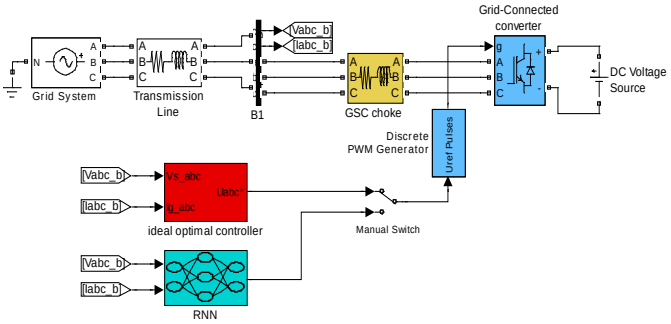


Fig.13 Vector control of GCC in power converter switching environment

Section V.A compares the performance of the ideal optimal controller and the neural network controller, while Sections V.B-V.D evaluate the performance of the neural

network vector controller under different stringent conditions, including distorted grid voltage, PWM saturation and voltage control mode. In each experiment, the proposed neural network controller structure (only two error terms and two integral terms fed into the neural network vector controller) was used as the network inputs, as indicated in Fig. 6.

A. Performance comparison of neural network vector controller and ideal optimal controller

Fig. 14 compares the performance of the neural network vector controller and the ideal optimal controller in tracking the d-q reference currents. The first four plots show the performance of the ideal optimal controller under the different sampling rates of $T_s = 0.001s$, $T_s = 0.0001s$, $T_s = 0.00001s$, and $T_s = 2e-6s$. The fifth plot shows the performance of the neural network vector controller under a sampling rate of $T_s = 0.001s$.

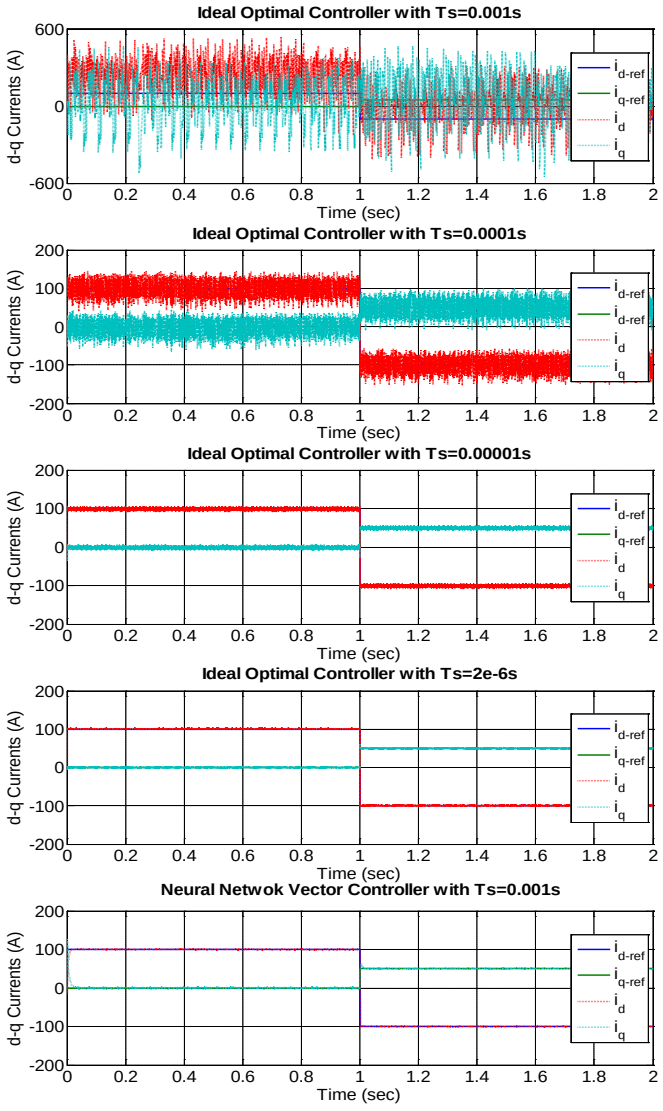


Fig. 14 Performance comparison between neural network vector controller and ideal optimal controller: d-q currents.

Small discretization is required to make an ideal optimal controller work properly. When the sampling time became sufficiently small, e.g., $T_s = 2e-6s$, which is also the sampling time generally used to simulate hardware systems, the ideal optimal controller exhibited very good performance, as would be expected theoretically (Fig. 4). However, such a high sampling frequency with $f = 1/T_s = 1/2e-6 = 50MHz$ is computationally expensive and can cause potential overrun problems in hardware implementation. In addition, the ideal optimal controller could not tolerate any changes in the system parameters. All of these factors make the practical application of ideal optimal controllers difficult. The fifth plot in Fig. 14 demonstrates the good tracking performance of the proposed neural network vector controller with a sampling time of $T_s = 0.001s$, which is very close to the performance of the ideal optimal controller with a sampling time of $T_s = 2e-6s$, corresponding to the fourth plot in Fig. 14. However, under a sampling rate of $T_s = 0.001s$, the ideal optimal controller performed very poorly. Fig. 14 illustrates that the proposed neural network controller achieved very good tracking performance, close to that of the ideal optimal controller, under a low sampling rate, which makes optimal control feasible in reality using neural networks.

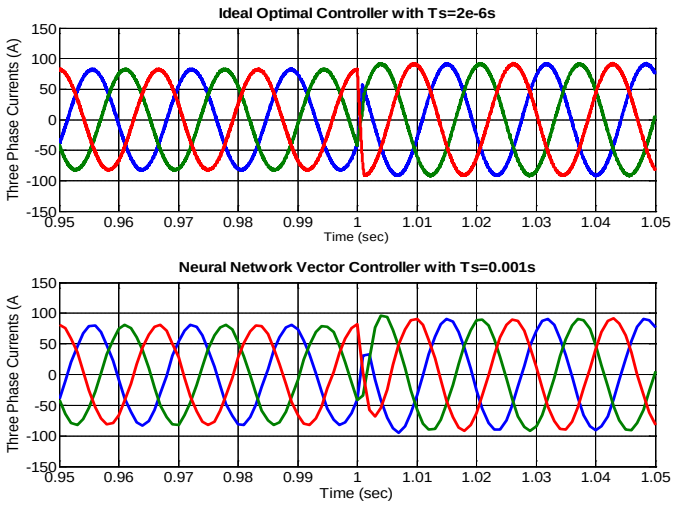


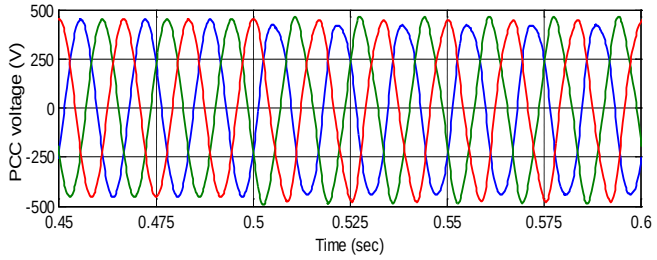
Fig. 15 Performance comparison between neural network vector controller and ideal optimal controller: three-phase currents.

Fig. 15 compares the actual three-phase currents using the ideal optimal controller with a sampling time of $T_s = 2e-6s$ and the neural network controller with a sampling time of $T_s = 0.001s$. Even though the actual d-q current oscillated around the reference current because of the switching impact (as indicated by Fig. 14), the actual three-phase grid current was properly balanced, and the neural network vector controller performed close to the ideal optimal controller.

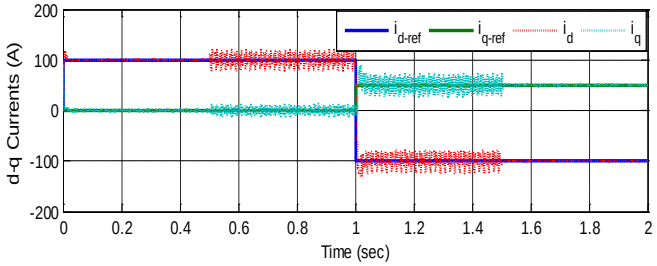
B. Performance evaluation of neural network vector controller under distorted grid voltage conditions

The grid voltage often is distorted in reality and shows high-order harmonics, which is caused by nonlinear loads in general. Fig. 16 shows the performance of the neural network vector controller under a distorted grid voltage condition. The

voltage distortion appears between 0.5s and 1.5s. This distorted grid voltage (Fig. 16a) would cause difficulty with the vector control. However, the neural network vector controller still performed very well under this condition, as shown in Fig. 16b, which demonstrates the robustness of the neural network vector controller trained using the proposed FATT+LM algorithm.



a) PCC voltage



b) d- and q-axis currents

Fig. 16 d-q current under distorted grid voltage

C. *Performance evaluation of neural network vector controller under PWM saturation*

In practice, outer-loop controllers (Fig. 3) may generate a d-q reference current signal that could cause the converter to exceed its PWM saturation limit. When the actual inductance is higher than the nominal inductance, it is easier for the GCC to reach PWM saturation, as explained in [35], particularly under reactive power generation conditions. Under PWM saturation, the conventional vector control method will malfunction [4]. However, the RNN controller trained by FATT+LM can automatically maintain the effectiveness of the d-axis current control while satisfying the q-axis control needs as much as possible, as shown in Fig. 17. The controller automatically operates in this mode during the period between 1s and 2s when there is a high demand for reactive power. This property would ensure the appropriate operation of the GCC under PWM saturation constraints.

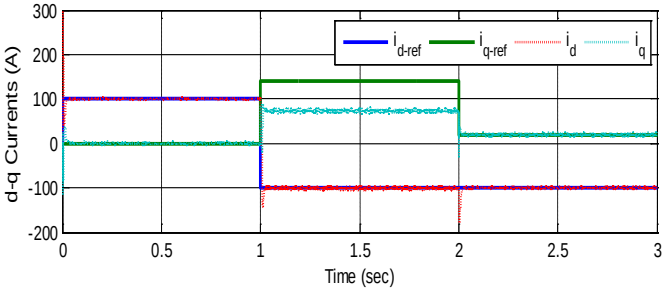
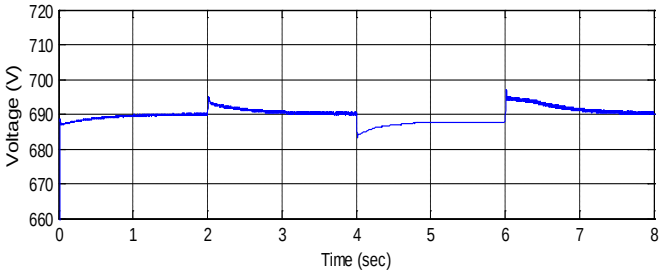


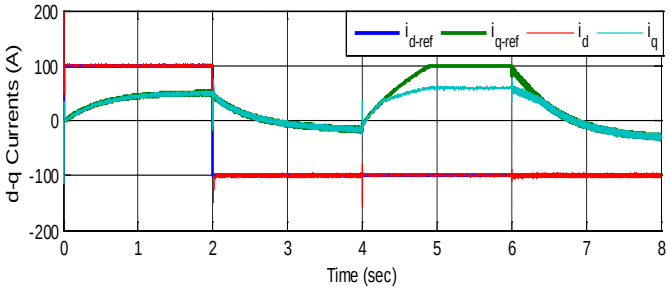
Fig. 17 d-q current under PWM saturation

D. *Performance evaluation of neural network vector controller under grid voltage control mode*

Fig. 18 depicts the performance of the trained neural network vector controller in PCC voltage control mode. In the figure, a grid voltage fault simulated by connecting with a fault load appears between 4s and 6s, causing a sudden PCC voltage reduction. However, the controller trained using the proposed FATT+LM algorithm performed very well, as shown in Fig. 18. Due to the converter's PWM saturation constraint, the RNN controller could not maintain the PCC voltage at 1 per unit to compensate for the voltage reduction during the fault (Fig. 18a). Instead, it operated by maintaining the effectiveness of the d-axis current control while providing PCC voltage support control as much as possible (Fig. 18b). At t=6s, when the fault was cleared, the neural network vector controller returned to its normal operating condition, and the PCC bus voltage quickly recovered to the rated bus voltage.



a) PCC voltage



b) d-q current

Fig. 18 RNN controller under PCC voltage control mode

VI. Conclusions

This paper investigated how to use Levenberg–Marquardt (LM) to train a RNN for optimal control of a GCC and studied the relationship between the RNN, ideal optimal controller and suboptimal controller for GCCs. In particular, we explained how to extend the LM algorithm for training a RNN by showing how the Jacobian matrix can be defined and found for RNNs. The paper demonstrates that the proposed LM-FATT algorithm is efficient and reliable and converges faster. The training results show that the proposed LM-FATT algorithm can solve the RNN tracking problem very well.

The study showed that although an ideal optimal controller for a GCC can track a target in one time step, it usually requires a control voltage that is beyond the physical system constraints. Other issues associated with an ideal controller include sensitivity to variations in system parameters and noises, and poor performance at a low sampling rate.

However, the proposed RNN vector controller achieved close to ideal controller performance, even under many challenging dynamic, variable, and power converter switching conditions. The RNN controller can use a much lower sampling rate than that used for the ideal optimal controller while maintaining performance equivalent to that of the ideal optimal controller at a high sampling rate. This would significantly reduce the computing time and enhance the deployment of the proposed RNN controller to real-life systems.

REFERENCES

[1] G. Buticchi, D. Barater, E. Lorenzani, and G. Franceschini, "Digital Control of Actual Grid-Connected Converters for Ground Leakage Current Reduction in PV Transformerless Systems," *IEEE Transactions on Industrial Informatics*, Vol. 8, No. 3, August 2012.

[2] J. Shen, H. Jou, and J. Wu, "Novel Transformerless Grid-Connected Power Converter with Negative Grounding for Photovoltaic Generation System," *IEEE Transactions on Power Electronics*, Vol. 27, No. 4, April 2012.

[3] X. Zhou, S. Lukic, S. Bhattacharya, and A. Huang, "Design and Control of Grid-Connected Converter in Bi-Directional Battery Charger for Plug-In Hybrid Electric Vehicle Application," *Vehicle Power and Propulsion Conference*, 2009.

[4] S. Li, T.A. Haskew, Y. Hong, and L. Xu, "Direct-Current Vector Control of Three-Phase Grid-Connected Rectifier-Inverter," *Electric Power System Research (Elsevier)*, Vol. 81, Issue 2, February 2011, pp. 357-366.

[5] S. Li, M. Fairbank, D. C. Wunsch, and E. Alonso, "Vector Control of a Grid-Connected Rectifier/Inverter Using an Artificial Neural Network," *Proceedings of 2012 IEEE World Congress on Computational Intelligence*, Brisbane, Australia, June, 10-15, 2012.

[6] S. Li, M. Fairbank, C. Johnson, D.C. Wunsch, and E. Alonso, "Artificial Neural Networks for Control of a Grid-Connected Rectifier/Inverter under Disturbance, Dynamic and Power Converter Switching Conditions," *IEEE Transactions on Neural Networks and Learning Systems*, 2014, 25(4), 738-750 .

[7] S. Li, M. Fairbank, X. Fu, D. C. Wunsch, and E. Alonso, " Nested-Loop Neural Network Vector Control of Permanent Magnet Synchronous Motors," *Proceedings of 2013 International Joint Conference on Neural Networks*, Dallas, Texas, USA, August 3-8, 2013.

[8] R. J. Williams and D. Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," *Neural Computation*, Vol.1, Issue 2, Summer 1989, pp. 270-280.

[9] R. J. Williams and D. Zipser, "Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity," in: *Back-Propagation: Theory, Architectures and Applications*, Hillsdale, NJ: Lawrence Erlbaum Associates,1995, ch.13, pp. 433-486.

[10] H. Jaeger, "Tutorial on Training Recurrent Neural Networks, Covering BPPT,RTRL, EKF and the 'Echo State Network' Approach," *GMD Report 159*, German National Research Center for Information Technology, 2002.

[11] S. Li, D. C. Wunsch, E. O'Hair, and M. G. Giesselmann, "Extended Kalman Filter Training of Neural Netowrks on a SIMD Parallel Machine," *Journal of Parallel and Distributed Computing*, Vol. 62, Issue 4, April 2002, pp 544 – 562.

[12] G. V. Puskorius and L. A. Feldkamp, "Neurocontrol of Nonlinear Dynamical Systems with Kalman Filter Trained Recurrent Networks," *IEEE Transactions on Neural Networks*, Vol. 5, No. 2, March 1994, pp. 279-297.

[13] R. Ilin, R.T Kozma and P. J, Werbos, "Beyond Feedforward Models Trained by Backpropagation: A Practical Training Tool for a More Efficient Universal Approximator," *IEEE Transactions on Neural Networks*, Vol. 19, No.6, June 2008, pp. 929-937.

[14] K. Levenberg, "A Method for the Solution of Certain Non-Linear Problems in Least Squares," *Quart. Appl. Math.*, Vol. 2, 1944, pp. 164–168.

[15] D. W. Marquardt, "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," *Journal of the Society for Industrial and Applied Mathematics*, Vol. 11, June 1963, pp. 431–441.

[16] M. T. Hagan and M. B. Menhaj, "Training Feedforward Networks with the Marquardt Algorithm," *IEEE Transactions on Neural Networks*, Vol. 5, No. 6, November 1994, pp. 989-993.

[17] Y. Tanoto, W. Ongsakul and C. O.P. Marpaung, "Levenberg-Marquardt Recurrent Networks for Long Term Electricity Peak Load Forecasting,"*TELKOMNIKA*, Vol. 9, No. 2, August 2011, pp. 257-266.

[18] L. Chan and C. Szeto, "Training Recurrent Network with Block-Diagonal Approximated Levenberg-Marquardt Algorithm," *Proceedings of 1999 International Joint Conference on Neural Networks*, Washington, DC, Vol. 6, 1999, pp. 4043-4047.

[19] L. Chan and C. Szeto, "Weight Groupings in the Training of Recurrent Networks," Proceedings of 2000 International Joint Conference on Neural Networks, Como, Italy, Vol. 3, 2000, pp. 21-26.

[20] D. Zhi, L. Xu, and B.W. Williams, "Improved Direct Power Control of Grid-Connected DC/AC Converters," *IEEE Transactions on Power Electronics*, Vol. 24, No. 5, May 2009, pp. 1280-1292.

[21] J.A. Restrepo, J.M. Aller, J.C. Viola, A. Bueno, and T.G. Habetler, "Optimum Space Vector Computation Technique for Direct Power Control," *IEEE Transactions on Power Electronics*, Vol. 24, No. 6, June 2009, pp. 1637-1645.

[22] N. Mohan, T. M. Undeland, and W. P. Robbins, Power Electronics: Converters, Applications, and Design, 3rd Ed., John Wiley & Sons Inc., October 2002.

[23] R. E. Bellman, *Dynamic Programming.* Princeton, NJ: Princeton Univ. Press, 1957.

[24] D. V. Prokorov and D. C. Wunsch, "Adaptive Critic Designs," *IEEE Transactions on Neural Networks*, Vol. 8, No. 5, 1997, pp. 997–1007.

[25] M. Fairbank, S.Li, X. Fu, E. Alonso, and D. Wunsch, "An Adaptive Recurrent Neural-Network Controller Using a Stabilization Matrix and Predictive Inputs to Solve a Tracking Problem under Disturbances," *Neural Networks (2013)*,http://dx.doi.org/10.1016/j.neunet.2013.09.010

[26] S. Haykin, *Neural Networks – A Comprehensive Foundation*, Upper Saddle River, NJ: Prentice Hall (1999).

[27] P. J. Werbos, "Backpropagation Through Time: What It Does and How to Do It," *Proceedings of the IEEE*, Vol. 78, No. 10, 1550-1560, 1990.

[28] P. J. Werbos, "Neural Networks, System Identification, and Control in the Chemical Process Industries," in Handbook of Intelligent Control, White and Sofge, Eds. New York: Van Nostrand Reinhold, 1992, ch. 10, sec. 10.6.1–10.6.2, pp. 339–343. [Online]. Available: www.werbos.com

[29] P. J. Werbos, "Backwards Differentiation in AD and Neural Nets: Past Links and New Opportunities," in Automatic Differentiation: Applications, Theory, and Implementations, ser. Lecture Notes in Computational Science and Engineering, H. M. B¨ucker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, Eds. Springer, 2005, pp. 15–34.

[30] M. T. Hagan, H. B. Demuth, and M. H. Beale, "Neural Network Design," Boston: PWS, 2002, ch.12, pp 19-23.

[31] M. Fairbank and E. Alonso, "Efficient Calculation of the Gauss-Newton Approximation of the Hessian Matrix in Neural Networks," *Neural Computation*, Vol. 24(3), 2012, pp. 607–610.

[32] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, "Numerical Recipes in C: The Art of Scientific Computing (second edition)," Cambridge University Press, October 1992, pp. 994.

[33] A. Mullane, G. Lightbody, and R. Yacamini, "Wind-Turbine Fault Ride-Through Enhancement," *IEEE Transactions on Power Systems*, Vol. 20, No. 4, Nov. 2005.

[34] R. Pena, J.C. Clare, and G. M. Asher, "Doubly Fed Induction Generator Using Back-to-Back PWM Converters and Its Application to Variable Speed Wind-Energy Generation," *IEEE Proc.-Electr. Power Appl.*, Vol. 143, No 3, May 1996, pp. 231-241.

[35] S. Li, T.A. Haskew, and L. Xu, "Control of HVDC Light Systems Using Conventional and Direct-Current Vector Control Approaches," *IEEE Transactions on Power Electronics*, Vol. 25, No. 12, December 2010, pp. 3106-3118.

[36] M. Riedmiller and H. Braun, "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm," *Proc. of the IEEE Intl. Conf. on Neural Networks*," San Francisco, CA, 1993, pp. 586-591.

[37] G.F. Franklin, J.D. Powell, M.L. Workman, *Digital Control of Dynamic Systems*, 3rd edition, Addison-Wesley, 1998.