Proceedings of the International Joint Conference on Neural Networks, 1-8 June 2008, Hong Kong. pp528-533

Prediction of Convergence Dynamics of Design Performance using Differential Recurrent Neural Networks

Yi Cao, *Member, IEEE*, Yaochu Jin, *Senior Member, IEEE*, Michal Kowalczykiewicz and Bernhard Sendhoff

***Abstract*— Computational Fluid Dynamics (CFD) simulations have been extensively used in many aerodynamic design optimization problems, such as wing and turbine blade shape design optimization. However, it normally takes very long time to solve such optimization problems due to the heavy computation load involved in CFD simulations, where a number of differential equations are to be solved. Some efforts have been seen using feedforward neural networks to approximate CFD models. However, feedforward neural network models cannot capture well the dynamics of the differential equations. Thus, training data from a large number of different designs are needed to train feedforward neural network models to achieve reliable generalization. In this work, a technique using differential recurrent neural networks has been proposed to predict the performance of candidate designs before the CFD simulation is fully converged. Compared to existing methods based on feedforward neural networks, this approach does not need a large number of previous designs. Case studies show that the proposed method is very promising.**

I. Introduction

In recent years, Computational Fluid Dynamics (CFD) techniques have been extensively used in many aerodynamic design optimization problems, such as wing and turbine blade shape design [1]. A computation flow chart to implement such techniques is shown in Figure 1.

In this approach, optimization is conducted in an outer loop by sending different design parameters to the CFD model and then evaluating the design performance according to the specified design criteria to decide either to alter the design parameters for further evaluation or to consider the optimization as converged hence to stop the computation. The design performance is evaluated through the inner loop of the CFD model. Upon receiving a set of new design parameters, a number of partial differential equations of fluid dynamics will be solved based on a grid scheme for the specified geometry. Note that the mesh grid must also be generated for a given shape before CFD simulations can be performed. Once the solution process is converged, the design performance can then be evaluated against given criteria. In such design loops, the convergence process of the CFD simulation is the most time consuming part of the whole computational overhead. For example, a three-dimensional CFD simulation using Navier-Stokes equations could take several hours to converge [2]. Since a sophisticated optimization algorithm requires hundreds or even thousands of criterion evaluations, the whole design process will need several days or weeks to achieve an acceptable design solution.

Yi Cao and Michal Kowalczykiewicz are with the School of Engineering, Cranfield University, Beford, MK43 0AL, UK. Yaochu Jin and Bernhard Sendhoff are with the Honda Research Institute Europe, 63073 Offenbach, Germany. Email: yaochu.jin@honda-ri.de.

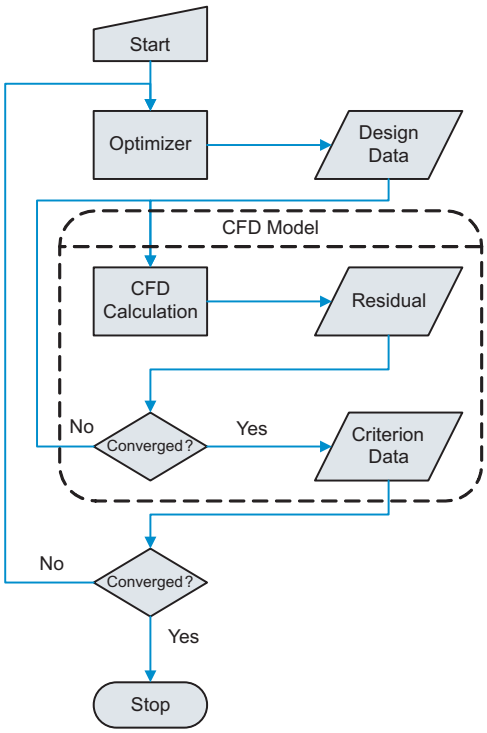

Fig. 1. Flow chart of CFD based design optimization

Some efforts have been reported in the literature to reduce the computation time of the above procedure. One approach is to replace the CFD simulation with a computationally efficient surrogate, such as a trained feedforward neural network model to significantly reduce the computation time for criteria evaluations [3]. However, empirical studies showed that the generalization performance of such neural network models becomes poor as the dimension of the design space increases [4], [5]. Other related work includes the use of feedforward neural networks to solve ordinary and partial differential equations [6], [7], and discrete-time neural networks [8], or continuous-time (differential) neural network [9] to model dynamic systems. The continuous-time neural networks bring further advantages and computational efficiency over the discrete formulation even if at the end both are represented on the computer using only discrete values [10].

Instead of approximating the stationary mapping from the design space (geometry) to the design criteria, an autonomous differential recurrent neural network (DRNN) model is sug-

978-1-4244-1821-3/08/$25.00©2008 IEEE

gested in this work to approximate the convergence process of CFD simulations, i.e., to learn the dynamics described by the differential equations. Once the DRNN model is appropriately trained, it can predict the convergence only using a few CFD runs such that the optimizer can alter design parameters much earlier than using the CFD simulation alone and thus the overall computation time can be significantly reduced.

The remainder of the paper is organized as follows. In Section II, the structure of the DRNN model is presented, followed by a description of an efficient DRNN training algorithm based on automatic differentiation (AD) techniques and the Levenberg-Marquardt algorithm. In Section III, we explain briefly how the DRNN can be applied to CFD convergence prediction. The performance of the DRNN for convergence prediction on a few cases of blade design is empirically studied in Section IV. Section V concludes the paper.

II. An Autonomous DRNN Model

The autonomous DRNN model used in this work is based on the multi-layer perceptron (MLP) structure with recurrency, as shown in Figure 2.

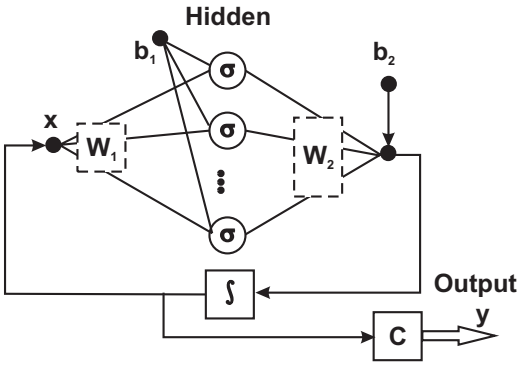

Fig. 2. Autonomous differential recurrent neural network structure.

The MLP based autonomous DRNN model in Figure 2 can be described as follows.

$$\begin{aligned} \dot{x} &= W_2\sigma(W_1 x + b_1) + b_2, \\ y &= Cx, \end{aligned} \tag{1}$$

where, $x \in \mathbb{R}^{n_x}$ are the internal states, $y \in \mathbb{R}^{n_y}$ the output to predict the output of a dynamic system, $\sigma \in \mathbb{R}^{n_h}$ the tanh-sigmoid function, whilst $W_1 \in \mathbb{R}^{n_h \times n_x}$, $W_2 \in \mathbb{R}^{n_x \times n_h}$, $b_1 \in \mathbb{R}^{n_h}$ and $b_2 \in \mathbb{R}^{n_x}$ are weight matrices and bias vectors, respectively. For simplicity, the output is directly connected to the first n_y internal states. Therefore, $C = \begin{bmatrix} I_{n_y} & 0_{n_y \times (n_x - n_y)} \end{bmatrix}$ is a constant matrix determined by n_y and n_x. Training of DRNNs involves the determination of the initial internal states and parameters in equation (1), which is done through parameter optimization based on a set of training data. To make the training algorithm efficient, the sensitivity of the model is analyzed through the recursive Taylor expansion approach. The number of internal states and the number of hidden nodes need to be predefined by the user.

A. Sensitivity analysis

Given a sequence of training data consisting of N samples $\{\tilde{y}(t_i), i = 1, 2, \cdots, N\}$, where $\tilde{y}(t_i) = \tilde{y}(ih) = \tilde{y}_i$ with h the virtual time unit to convert the discrete point to continuous time. It is required for the autonomous DRNN model to produce output $y(t_i)$ at the same set of time instances so that the model error $e(t_i) = y(t_i) - \tilde{y}(t_i)$ can be evaluated at these time instances.

Let $x_i = x(t_i)$ represent the initial state of the autonomous DRNN model at $t = t_i = ih$. For $0 \le \tau \le 1$, the state of the model, $x(t)$ at $t = (\tau + i)h$ can be represented using the Taylor series as follows:

$$x(t) = x_i^{[0]} + x_i^{[1]}\tau + x_i^{[2]}\tau^2 + \cdots + x_i^{[d]}\tau^d, \tag{2}$$

where the superscripts with square brackets represent the order of the Taylor coefficients, whilst the subscripts represent the reference time point where the Taylor coefficients are calculated. Let $z(t)$ be the right-hand-side of (1), *i.e.*

$$z(t) = W_2\sigma(W_1 x(t) + b_1) + b_2 = f(x, \theta), \tag{3}$$

where $\theta \in \mathbb{R}^{n_\theta}$ is the parameter vector defined as follows:

$$\theta = \begin{bmatrix} W_1^T & b_1^T & W_2^T & b_2^T \end{bmatrix}^T \in \mathbb{R}^{n_\theta}. \tag{4}$$

Clearly, $n_\theta = 2n_x \times n_h + n_x + n_h$. Using the automatic differentiation techniques [11], [12], the coefficients of the Taylor series $z(t) = \sum_{k=0}^{d} z_i^{[k]}(t - t_0)^k$ can be derived as follows:

$$z_i^{[k]} = f_i^{[k]}(x_i^{[0]}, x_i^{[1]}, \ldots, x_i^{[k]}, \theta). \tag{5}$$

On the other hand, since $\dot{x}(t) = z(t)$, the following recursive equations can be derived:

$$x_i^{[k+1]} = \frac{h}{k+1} z_i^{[k]} = \frac{h}{k+1} f_i^{[k]}(x_i^{[0]}, x_i^{[1]}, \ldots, x_i^{[k]}, \theta) \tag{6}$$

Using equation (6), all Taylor coefficients, $x_i^{[k]}$, $k = 1, \ldots, d$ can be iteratively obtained from $x_i^{[0]} = x_i$. Therefore, $x(t_{i+1})$ can be derived as follows:

$$x(t_{i+1}) = \sum_{k=0}^{d} x_i^{[k]}. \tag{7}$$

This process is repeatedly conducted for $i = 0, \ldots, N-1$ until states at all required time instances are evaluated. Then, the output can be simply calculated as $y(t_i) = Cx(t_i)$, $i = 0, \ldots, N$.

Based on the above expressions, the sensitivity of $y(t_i)$ against θ and x_0 can be derived as follows. From equation (5), it can be derived that the sensitivities of $z_i^{[k]}$ against $x_i^{[j]}$ for $j = 0 \ldots, k$ satisfy the following relations [13]:

$$A_{ix}^{[k-j]} := \frac{\partial z_i^{[k]}}{\partial x_i^{[j]}} = \frac{\partial z_i^{[k-j]}}{\partial x_i^{[0]}}, \tag{8}$$

$$A_{i\theta}^{[k]} := \frac{\partial z_i^{[k]}}{\partial \theta}. \tag{9}$$

The sensitivity matrices, $A_{ix}^{[k]}$ and $A_{i\theta}^{[k]}$, $k = 0, \ldots, d$ are readily to be obtained using some automatic differentiation software tools such as ADOL-C [14]. Using equations (8) and (9), the total derivative of $x_i^{[k+1]}$ against $x_i^{[0]}$ and θ can be derived from equation (6).

$$B_{i\theta}^{[k+1]} := \frac{dx_i^{[k+1]}}{d\theta} = \frac{h}{k+1}\left(\frac{dz_i^{[k]}}{d\theta} + \sum_{j=0}^{k}\frac{\partial z_{[k]}}{\partial x_i^{[j]}}\frac{dx_i^{[j]}}{d\theta}\right),$$
$$= \frac{h}{k+1}\left(A_{i\theta}^{[k]} + \sum_{j=0}^{k} A_{ix}^{[k-j]}B_i^{[j]}\right), \quad (10)$$
$$B_{ix}^{[k+1]} := \frac{dx_i^{[k+1]}}{dx_i^{[0]}} = \frac{h}{k+1}\sum_{j=0}^{k}\frac{\partial z_{[k]}}{\partial x_i^{[j]}}\frac{dx_i^{[j]}}{dx_i^{[0]}},$$
$$= \frac{h}{k+1}\sum_{j=0}^{k} A_{ix}^{[k-j]}B_{ix}^{[j]}, \quad k = 0, \ldots, d-1, \quad (11)$$
$$B_{ix}^{[0]} = I,$$
$$B_{i\theta}^{[0]} = 0.$$

Therefore, according to equation (7),

$$D_{i+1,x} := \frac{\partial x(t_{i+1})}{\partial x(t_i)} = \frac{\partial x(t_{i+1})}{\partial x_i^{[0]}} = \sum_{k=0}^{d} B_{ix}^{[k]}, \quad (12)$$

$$D_{i+1,\theta} := \frac{\partial x(t_{i+1})}{\partial \theta} = \sum_{k=0}^{d} B_{i\theta}^{[k]}. \quad (13)$$

$$(14)$$

Furthermore,

$$F_{i+1,x} := \frac{dx(t_{i+1})}{dx_0} = \frac{\partial x(t_{i+1})}{\partial x(t_i)}\frac{dx(t_i)}{dx_0},$$
$$= D_{i+1,x}F_{i,x} \quad (15)$$
$$F_{i+1,\theta} := \frac{dx(t_{i+1})}{dx_0} = \frac{\partial x(t_{i+1})}{\partial \theta} + \frac{\partial x(t_{i+1})}{\partial x(t_i)}\frac{dx(t_i)}{d\theta}$$
$$= D_{i+1,\theta} + D_{i+1,x}F_{i,\theta}. \quad (16)$$

Hence, the output sensitivity can be derived as follows:

$$\frac{dy(t_{i+1})}{dx_0} = CF_{i+1,x}, \quad (17)$$
$$\frac{dy(t_{i+1})}{d\theta} = CF_{i+1,\theta}. \quad (18)$$

B. Training algorithm

The optimization problem for the DRNN training is to minimize the total prediction error by adjusting θ and x_0:

$$\min_{\theta,x_0}\varphi = \min_{\theta,x_0}\frac{1}{2}\sum_{k=1}^{N} e_k^T e_k = \min_{\theta,x_0}\frac{1}{2}E^T E, \quad (19)$$

where $e_k := y(kh) - \tilde{y}(k)$ with $\tilde{y}$ being the actual CFD convergence sequence data, and $E := [e_1^T \cdots e_N^T]^T$. This is a nonlinear least square problem, which can be efficiently solved. Based on the Levenberg-Marquardt algorithm [15], the parameters and initial states can be iteratively updated as follows:

$$\begin{bmatrix} x_0^{k+1} \\ \theta^{k+1} \end{bmatrix} = \begin{bmatrix} x_0^{k} \\ \theta^{k} \end{bmatrix} - (J^T J + \lambda I)^{-1} J^T E, \quad (20)$$

where λ is determined by the algorithm to make sure the prediction error is reduced at each iterative step, whilst J is the Jacobian matrix defined as follows:

$$J := \begin{bmatrix} J_x & J_\theta \end{bmatrix},$$
$$J_x := \frac{\partial E}{\partial x_0} = \begin{bmatrix} J_{1,x} \\ \vdots \\ J_{N,x} \end{bmatrix},$$
$$J_\theta := \frac{\partial E}{\partial \theta} = \begin{bmatrix} J_{1,\theta} \\ \vdots \\ J_{N,\theta} \end{bmatrix},$$
$$J_{k,x} := \frac{\partial e_k}{\partial x_0} = \frac{\partial y(t_k)}{\partial x_0} = CF_{k,x},$$
$$J_{k,\theta} := \frac{\partial e_k}{\partial \theta} = \frac{\partial y(t_k)}{\partial \theta} = CF_{k,\theta}.$$

III. DRNN FOR PREDICTION OF CFD CONVERGENCE

Computational fluid dynamics (CFD) simulations use numerical methods to solve and analyze problems that involve gas or fluid flows, in which the flow interacts with a solid surface such as a stator or rotor blade of turbine engines. One popular approach to simulate such interactions is to discretize the continuous fluid into small two or three dimensional grids, and then apply a suitable algorithm to solve the equations of motion described by Euler equations for inviscid, and Navier-Stokes equations for viscous flow [16]. CFD simulations are highly time consuming even on supercomputers.

The convergence dynamics of CFD simulations is governed by a set of nonlinear autonomous ordinary differential equations. Therefore, an autonomous differential recurrent neural network (DRNN) is capable of approximating the dynamic system, thus predicting the convergence process. The aim of the DRNN model is to predict the final converged performance value as early as possible by feeding with the current and past convergence data. To enable the DRNN model to predict future convergence trend, a virtual time can be introduced to represent the convergence iteration, *i.e.* each iteration is converted to one unit of time.

The autonomous DRNN does not have any input, whilst the output represent the CFD convergence sequence. The model will evolve itself continuously to produce the output signal based on current (initial) states. Therefore, to make the output of the model match the actual CFD convergence sequence, the model parameters and initial states have to be appropriately trained. These parameters and initial states will be different from design to design and for different grid sizes. The autonomous DRNN is trained using the fluid dynamics performance data from a number of first simulation iterations of a candidate design. Then, the DRNN is used to predict the remaining iterations of the CFD performance for the same

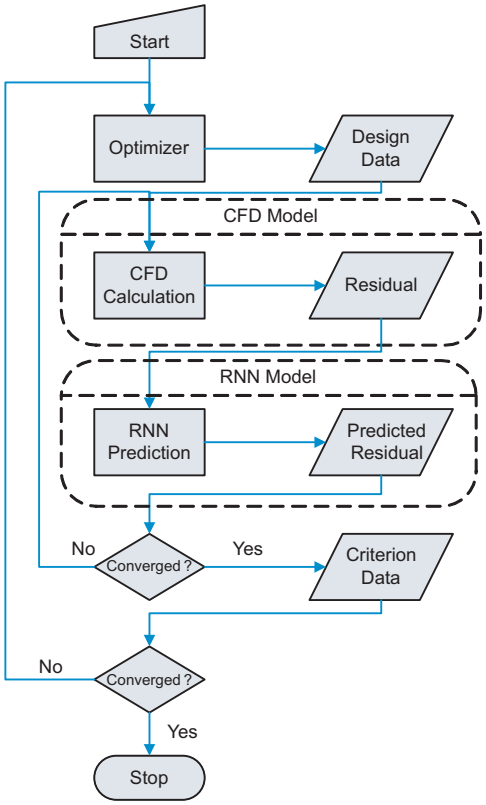

Fig. 3. Flow chart of CFD based design optimization with a convergence predictor using DRNN

design, refer to Figure 3 to see how the DRNN can be used in an optimization loop. In this way, the number of iterations needed for a CFD simulation can be reduced and thus the simulation time.

IV. Simulation Results

Case studies are conducted to investigate the feasibility of DRNN based performance prediction for turbine blade design. In the design, the geometry of the blades are represented by a B-spline, consisting of a number of control points. The target of the design optimization is to minimize the pressure loss and the deviation of the outflow angle from a predefined value. For two-dimensional (2D) CFD simulations, each control point has two design parameters, i.e., its x and y coordinates. Refer to [1], [4] for more details on B-spline based blade design.

To verify the prediction performance of DRNN, 2D CFD simulations are performed on 100 candidate designs created by making random modifications to an initial design. Four sets of CFD convergence data are collected for the candidate designs with four different grid sizes. In the CFD simulation, a maximum of 8000 iterations is set for data sets I, II and III, and 16000 iterations for data set IV to ensure that the simulations are converged. In data set I, where the grid size is 71×35, one of the 100 designs does not satisfy the mechanical constraint and therefore is removed. Refer to Table I for an overview of the specifications of the data sets, Table II for the statistics of the data sets, including mean and standard deviation of the different designs within the same data set, and Table III for the structural parameters of the DRNN model.

TABLE I
CFD convergence data sets

data set	grid size	# of designs	# of iterations
I	71×35	99	8,000
II	191×55	100	8,000
III	299×91	100	8,000
IV	299×91	100	16,000

TABLE II
Mean and standard deviation of the data sets.

data set	variable	mean	variance
I	pressure loss	0.12623	2.93×10^{-4}
I	outflow angle	60.011	0.018723
II	pressure loss	0.11599	0.008079
II	outflow angle	60.025	0.12887
III	pressure loss	0.11183	0.004250
III	outflow angle	60.116	0.10493
IV	pressure loss	0.11179	0.004217
IV	outflow angle	60.117	0.10502

The convergence data from the first certain number of CFD iterations, termed training length, are used for training. The trained model is then employed to predict the remaining part of convergence data. The training length used and total prediction error achieved are summarized in Table IV, where the prediction error is defined at the final point of convergence as $e = |y - \tilde{y}|/|\tilde{y}|$, whilst $\mu(e) = \sum(e)/N$ the mean error and $\sigma(e) = (\sum(e - \mu(e))^2/N)^{1/2}$ the standard deviation with N the number of designs in each data set. Details of all convergence sequences and prediction errors are presented in Figures 4–11.

TABLE III
DRNN model parameters

data set	variable	n_x	n_h	d
I	pressure loss	3	3	3
I	outflow angle	3	3	4
II	pressure loss	3	3	3
II	outflow angle	3	4	4
III	pressure loss	3	3	3
III	outflow angle	3	4	4
IV	pressure loss	3	3	3
IV	outflow angle	3	4	4

Figures 4– 11 show that all convergence sequences are able to be correctly predicted with a reasonable accuracy. Table IV shows that all relative prediction errors at the convergence point are less than 5%. The training lengths required are less than 50% of total number of iterations. This means that at least 50% of computation time can be saved using the DRNN convergence prediction approach. However, if we take a look at the statistics of the training data shown in Table II, we see that the prediction on the outflow angle is more reliable.

TABLE IV

DRNN MODEL TRAINING LENGTH AND PREDICTION ERROR

data set	variable	training length	$\mu(e)$	$\sigma(e)$
I	pressure loss	2000	0.004	0.0035
I	outflow angle	2000	5.21×10^{-5}	1.12×10^{-4}
II	pressure loss	4000	0.0284	0.0193
II	outflow angle	2000	0.0014	9.41×10^{-4}
III	pressure loss	4000	0.0340	0.0132
III	outflow angle	2000	0.0015	0.0019
IV	pressure loss	2000	0.0281	0.0131
IV	outflow angle	4000	9.44×10^{-4}	0.0011

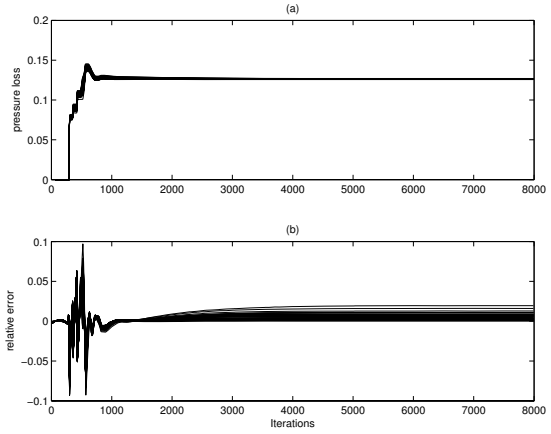

Fig. 4. Pressure loss convergence sequence (a) and prediction error (b) of all designs in data set I.

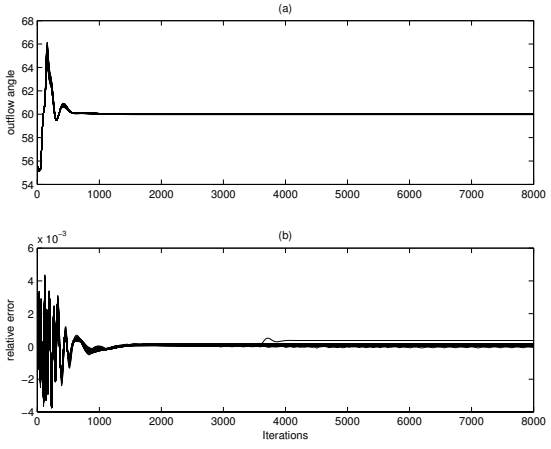

Fig. 5. Outflow angle convergence sequence (a) and prediction error (b) of all designs in data set I.

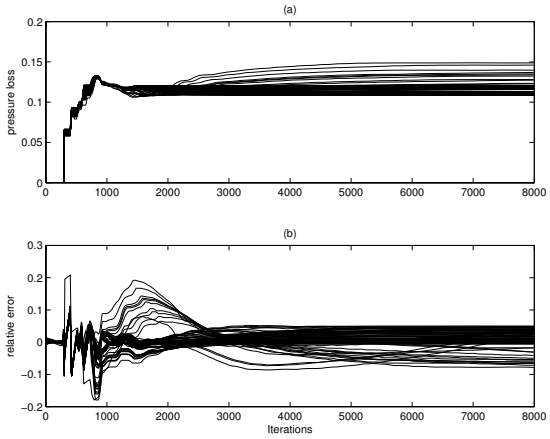

Fig. 6. Pressure loss convergence sequence (a) and prediction error (b) of all designs in data set II.

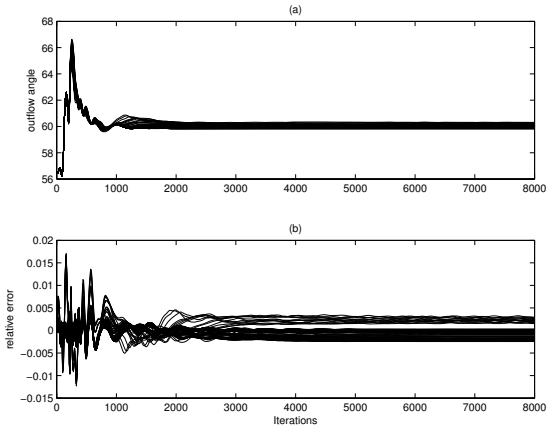

Fig. 7. Outflow angle convergence sequence (a) and prediction error (b) of all designs in data set II.

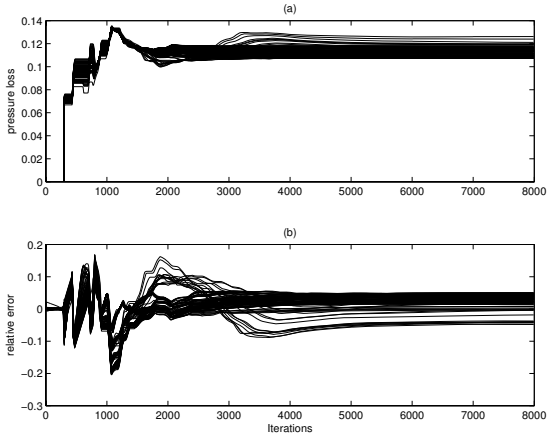

Fig. 8. Pressure loss convergence sequence (a) and prediction error (b) of all designs in data set III.

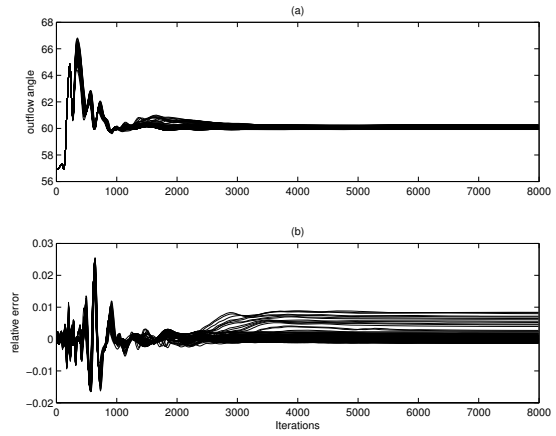

Fig. 9. Outflow angle convergence sequence (a) and prediction error (b) of all designs in data set III.

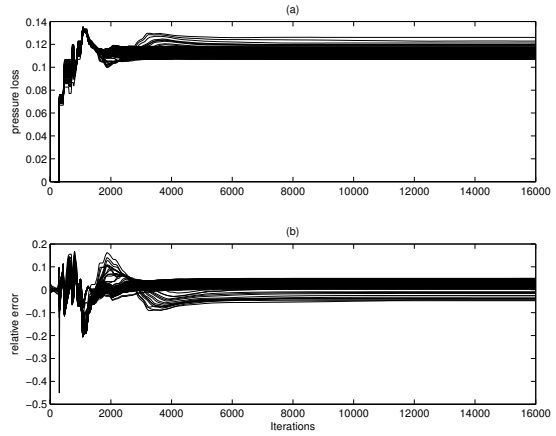

Fig. 10. Pressure loss convergence sequence (a) and prediction error (b) of all designs in data set IV.

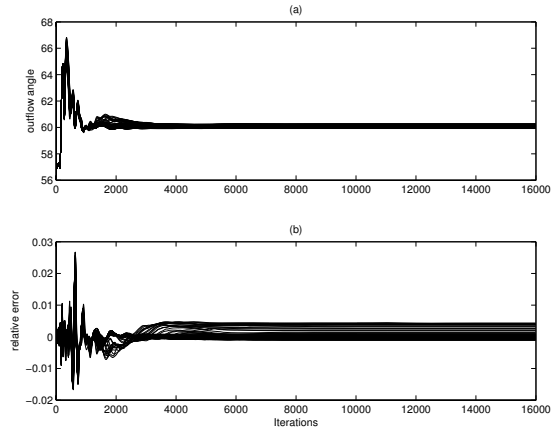

Fig. 11. Outflow angle convergence sequence (a) and prediction error (b) of all designs in data set IV.

V. Conclusions

CFD simulations have extensively been used in aerodynamic design optimization problems. However, the computational burden associated with the CFD convergence process is so large that it hinders these techniques to be adopted in wider engineering areas. In this work, a concept using a recurrent neural network model to approximate the convergence sequence of the CFD simulations is proposed. The model structure and training algorithm are developed and demonstrated with examples from turbine blade design optimization. The case study shows that the proposed model is able to predict the convergence sequence using less than half of the required number of iterations with a satisfying prediction accuracy. The next step of the work will integrate this prediction model into the design loop to reduce the computation time required by the optimization procedure.

References

[1] T. Sonoda, Y. Yamaguchi, T. Arima, M. Olhofer, B. Sendhoff, and H.-A. Schreiber, "Advanced high turning compressor airfoils for low reynolds number condition, Part 1: Design and optimization," *Journal of Turbomachinery*, vol. 126, pp. 350–359, 2004.

[2] M. Hasenjäger and B. Sendhoff, "Crawling along the pareto front: Tales from the practice," in *Congress on Evolutionary Computation*. IEEE, 2005, pp. 174–181.

[3] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary optimization," *Soft Computing*, vol. 9, no. 1, pp. 3–12, 2005.

[4] Y. Jin, M. Olhofer, and B. Sendhoff, "A framework for evolutionary optimization with approximate fitness functions," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 481–494, 2002.

[5] ——, "Reducing fitness evaluations using clustering techniques and neural network ensembles," in *Genetic and Evolutionary Computation Conference*, ser. LNCS 3102. Springer, 2004, pp. 688–699.

[6] I. Lagaris, A. Likas, and D. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[7] A. Meade and A. Fernandez, "Solution of nonlinear ordinary differential equations by feedforward neural networks," *Mathematial and Computer Modeling*, vol. 20, no. 9, pp. 19–44, 1994.

[8] J. M. Zamarreno and P. Vega, "State-space neural network, properties and application," *Neural Networks*, vol. 11, pp. 1099–1112, 1998.

[9] K. L. Funahashi and Y. Nakamura, "Approximation of dynamical systems by continues time recurrent neural networks," *Neural Networks*, vol. 6, pp. 183–192, 1993.

[10] B. A. Pearlmutter, "Gradient calculations for dynamic recurrent neural networks: A survey," *IEEE Trans. on Neural Networks*, vol. 6, no. 5, pp. 1212–1228, 1995.

[11] L. Rall, *Automatic Differentiation: Techniques and Applications*, ser. Lecture Notes in Computer Science, Vol. 120. Berlin: Springer Verlag, 1981.

[12] Y. Cao, "A formulation of nonlinear model predictive control using automatic differentiation," *Journal of Process Control*, vol. 15, pp. 851–858, 2005.

[13] B. Christianson, "Reverse accumulation and accurate rounding error estimates for taylor series," *Optimization Methods and Software*, vol. 1, pp. 81–94, 1992.

[14] A. Griewank, D. Juedes, and J. Utke, "ADOL-C: A package for the automatic differentiation of algorithms written in C/C++," *ACM Transactions on Mathematical Software*, vol. 22, pp. 131–167, 1996.

[15] D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters." *SIAM Journal of Applied Mathematics*, vol. 11, pp. 431–441, 1963.

[16] T. Chung, *Computational Fluid Dynamics*. Cambridge University Press, 2002.