International Journal of Chemical, Materials and Biomolecular Sciences
ISSN: 2415-6620
Vol:12, No:11, 2018

Dynamic Measurement System Modeling with Machine Learning Algorithms

Changqiao Wu, Guoqing Ding, Xin Chen

***Abstract*—**In this paper, ways of modeling dynamic measurement systems are discussed. Specially, for linear system with single-input single-output, it could be modeled with shallow neural network. Then, gradient based optimization algorithms are used for searching the proper coefficients. Besides, method with normal equation and second order gradient descent are proposed to accelerate the modeling process, and ways of better gradient estimation are discussed. It shows that the mathematical essence of the learning objective is maximum likelihood with noises under Gaussian distribution. For conventional gradient descent, the mini-batch learning and gradient with momentum contribute to faster convergence and enhance model ability. Lastly, experimental results proved the effectiveness of second order gradient descent algorithm, and indicated that optimization with normal equation was the most suitable for linear dynamic models.

***Keywords*—**Dynamic system modeling, neural network, normal equation, second order gradient descent.

I. Introduction

IN the industry, kinds of measuring systems are adapted to dynamic measurement scenarios. With the increasing requirements for the products, increasingly importance has been attached to the dynamic performance of sensors. Usually, one needs to model the system properly for better usage, or dynamically compensates for measurement error caused by dynamic property of the system. With knowing the orders of the model, conventional dynamic modeling takes use of attenuation amplitude to determine the dynamics parameters indirectly [1], or after the difference equation is established, one performs Z transform and bilinear transform to obtain discrete and continuous transfer function [2]. After the introduction of functional link artificial neural network (FLANN) [3], the method of learning differential model parameters is applied to sensor dynamic modeling and is well recognized [4]–[6], which describes the dynamic property of the sensor by constructing a shallow network with time delay characteristics. For conventional compensation methods, the zero-pole analysis method is based on the known modal parameters, which has no universality [7]; the premise of the deconvolution method is the accurate estimation of discrete transfer function [8]; the time domain optimization method works well, but it is not practical due to the heavy calculation [9]. As theoretically neural network is global function approximator [10], it is used to approximate the inverse compensation system function to correct errors [11]–[14]. And back-propagation (BP) [15] is applied to optimizing the network model, which uses the gradients descent algorithm to minimize the empirical risk [16]; Meanwhile, least square support vector machine (LS-SVM) is also used to optimize the dynamic model or inverse compensate system [5], [17].

We find out that the dynamic modeling of a system has certain similarity with the dynamic compensation. Both can be regarded as generating the output by a function approximator, which has time delay characteristics, with certain input. In the dynamic modeling process, the in-going signal of the system is also the input of function approximator. However, the out-going signal of the system is taken as the input of function approximator during dynamic compensation process. For different purposes, specific modeling methods differ. If one needs to analyze the amplitude-frequency characteristics of the system, firstly, the orders of the model should be calculated, then parameters are solved after establishing the difference equation. On the other hand, compensation is more prone to result orientation, and there are no strict requirements for the function approximator used.

Traditional optimization methods based on back-propagation iterate with only one sample, which leads to heavy bias of gradients approximation, and slows down the learning process. Meanwhile, some researchers validate the methods they proposed on the training set, and the generalization ability of the learned model is rarely discussed. In this paper, we introduce the mathematical essence of methods based on FLANN for modeling, and new way of optimization with normal equation. Finally, for different learning strategies, we propose a generalized model for dynamic modeling and compensation.

II. Dynamic Modeling and Objectives

A. Dynamic Model Based on FLANN

Usually, the dynamic system and the corresponding inverse compensation system of the sensor are single-input single-output, and the relationship between input and output can be represented with difference equation [6]. As for the dynamic modeling process (shown in box 1 of Fig. 1), the corresponding difference equation is:

$$y_t + a_1 y_{t-1} + \cdots + a_n y_{t-n} = b_0 x_t + b_1 x_{t-1} + \cdots b_m x_{t-m} + c + e_t \quad (1)$$

Changqiao Wu is with Department of Instrument Science and Engineering, School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, P. R. China (e-mail: innerpeace@sjtu.edu.cn).

Guoqing Ding is an Associate Professor in Department of Instrument Science and Engineering, School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, P. R. China (corresponding author, phone: +86-21-34204624, e-mail: gqding@sjtu.edu.cn).

Xin Chen is an Associate Professor in Department of Instrument Science and Engineering, School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, P. R. China (e-mail: xchen.ie@sjtu.edu.cn).

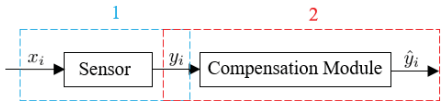

Fig. 1 Principle of dynamic modeling and compensating for transducer

where m and n represent orders of the model, $\{a_i\}$ and $\{b_i\}$ are the parameters, c is the bias and e_t means the random noise. x_t and y_t are the in-going and out-going signal of the system at time step t respectively.

Equation (1) can be vectorized as:

$$y_t = \boldsymbol{w}^T \boldsymbol{x}_t + c; \quad t = 1, 2, \cdots, N \tag{2}$$

where N is number of discrete samples at time step t, and weight vector is $\boldsymbol{w} = [-a_1, \cdots, -a_n, b_0, b_1, \cdots, b_m]^T$, the input vector of step t is $\boldsymbol{x}_t = [y_{t-1}, \cdots, y_{t-n}, x_t, \cdots, x_{t-m}]^T$.

Similarly, for the dynamic compensation process (shown in box 2 of Fig. 1), the corresponding difference equation is:

$$\begin{aligned}\hat{y}_t + a_1\hat{y}_{t-1} + \cdots + a_n\hat{y}_{t-n} =& b_0 y_t + b_1 y_{t-1} + \cdots b_m y_{t-m} \\ &+ c + e_t\end{aligned} \tag{3}$$

The vectorized version is:

$$\hat{y}_t = \boldsymbol{w}^T \boldsymbol{x}_t + c; \quad t = 1, 2, \cdots, N \tag{4}$$

Comparing to dynamic modeling, the difference is that the input of the model changes to $\boldsymbol{x}_t = [\hat{y}_{t-1}, \cdots, \hat{y}_{t-n}, y_t, \cdots, y_{t-m}]^T$. And it concludes that the dynamic modeling and compensation can be modeled with FLANN which can be solved by optimizing the parameters.

B. Objectives

To solve the parameters of (1), the intuitive method is taking the square of the bias between the estimated value and the ground truth as loss function. However, the reason behind the intuition is rarely analyzed. The mathematical essence of that is maximum likelihood estimation of output with noise under Independently, Identically Distributed (IID) Gaussian distribution [18]. For convenience, we define the weight parameter as $\boldsymbol{\theta} = [\theta_0, \theta_1, \cdots, \theta_{m+n+1}]^T$, where θ_0 is the bias c in (1), and the generalized formula is:

$$y^{(i)} = \boldsymbol{\theta}^T \boldsymbol{x}^{(i)} + \epsilon^{(i)} \tag{5}$$

where $\boldsymbol{x}^{(i)}$, $y^{(i)}$ represent the input and output of the i-th training sample respectively, and $\epsilon^{(i)}$ is the noise component of the model, which is under IID Gaussian distribution with mean as 0, and variance as σ^2. The Probability Density Function (PDF) of $\epsilon^{(i)}$ is:

$$p(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(- \frac{(\epsilon^{(i)})^2}{2\sigma^2} \right) \tag{6}$$

Along with (5), the conditional probability of output $y^{(i)}$ provided with weight $\boldsymbol{\theta}$ and input $\boldsymbol{x}^{(i)}$ is:

$$p(y^{(i)}|\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(- \frac{(y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)})^2}{2\sigma^2} \right) \tag{7}$$

As $\epsilon^{(i)}$ satisfies IID condition, for totally M training examples, the likelihood function of the probability is:

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{M} p(y^{(i)}|\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \tag{8}$$

According to principle of Maximum Likelihood, the weight should be chosen to maximize the probability over all training examples:

$$\boldsymbol{\theta} = \arg\max_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \tag{9}$$

The likelihood function $L(\boldsymbol{\theta})$ in (8) is the product over probabilities, which is difficult to analysis. Usually, we take the log function, which is strictly monotony, as proxy for optimization. Along with (7) and (8), the log likelihood of $L(\boldsymbol{\theta})$ is:

$$\begin{aligned}\log L(\boldsymbol{\theta}) &= \log \prod_{i=1}^{M} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(- \frac{(y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)})^2}{2\sigma^2} \right) \\ &= \sum_{i=1}^{M} \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(- \frac{(y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)})^2}{2\sigma^2} \right) \\ &= M \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^{M} (y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)})^2\end{aligned} \tag{10}$$

Due to the irrelevance between M, σ and $\boldsymbol{\theta}$, maximizing the likelihood is the same as maximizing the accumulation of square of bias between the estimation and ground truth for each example:

$$\max_{\boldsymbol{\theta}} \log L(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \frac{1}{2} \sum_{i=1}^{M} (y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)})^2 \tag{11}$$

From the view of machine learning, the process of dynamic modeling and compensation is the typical linear regression [18] problem set, and the common loss function for such problem is the same as (11), which is also referred as L_2 loss function, as the output is the 2 norm of the biases.

III. Optimizing Methods

A. Gradient Descent Optimization

The directional derivation in direction $\boldsymbol{u}$, which is a unit vector, is the slope of function $f(\boldsymbol{x})$ at that direction, and the slope can be represented as:

$$\frac{\partial}{\partial \alpha} f(\boldsymbol{x} + \alpha \boldsymbol{u}) = \boldsymbol{u}^T \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{12}$$

And it is found that the direction in which $f(\boldsymbol{x})$ decreases the fastest is the direction of the negative gradient of $f(\boldsymbol{x})$ with respect to variable $\boldsymbol{x}$. The optimizing method decreasing $f(\boldsymbol{x})$ in the direction of negative gradient is referred as Gradient Descent [18]. For now, most of the methods of dynamic modeling and compensating with neural networks use gradient descent as the optimizing algorithm, where the square of bias of estimation $\tilde{y}$ and ground truth y is the objective. Finally, optimization is conducted in the direction of negative gradient of objective with respect to weight vector $\boldsymbol{\theta}$.

During optimization, we refer weight $\boldsymbol{\theta}_{(i-1)}$, bias $b_{(i-1)}$ as the result after $i-1$ times iteration. For the ith iteration, the corresponding estimation is:

$$\tilde{y}^{(i)} = \boldsymbol{\theta}_{(i-1)}^T \boldsymbol{x}^{(i)} + b_{(i-1)} \tag{13}$$

According to (11), the objective function F is:

$$F = \frac{1}{2}(y^{(i)} - \tilde{y}^{(i)})^2 \tag{14}$$

Along with (13) and (14), the process of ith iteration is:

$$\begin{aligned} \boldsymbol{\theta}_{(i)} &= \boldsymbol{\theta}_{(i-1)} - \alpha \frac{\partial F}{\partial \boldsymbol{\theta}} \\ &= \boldsymbol{\theta}_{(i-1)} + \alpha(y^{(i)} - \tilde{y}^{(i)})\boldsymbol{x}^{(i)} \\ b_{(i)} &= b_{(i-1)} + \alpha(y^{(i)} - \tilde{y}^{(i)}) \end{aligned} \tag{15}$$

where the α is a hyper-parameter and is usually referred as learning rate, which is used to control the stride of each iteration. If α is too small, it would slow down convergence, and large α would result in oscillation or even divergence of objective.

The primary of gradient descent is estimation of gradient, which requires the objective should be continuous and differentiable under certain circumstances. Meanwhile the correctness of the gradient estimation is vital for the ability of the learned model.

B. Improved Gradient Descent

Inferred from the work of Dehui et al. [6] that conventional training of FLANN took use of only one sample to do gradient estimation and updated weights. With (15), we learned that the bias of input vector $\boldsymbol{x}$ would influence the correctness of gradient estimation, which is inevitable due to the noisy input signal. But we would alleviate the influence by averaging the gradients over several samples, as the noise is usually distributed under Gaussian distribution.

For a training set with totally M training samples, we randomly choose s samples as a mini-batch to estimate the gradient:

$$\begin{aligned} \boldsymbol{\theta}_{(i)} &= \boldsymbol{\theta}_{(i-1)} + \alpha \frac{1}{s} \sum_{i=1}^{s} (y^{(i)} - \tilde{y}^{(i)})\boldsymbol{x}^{(i)} \\ b_{(i)} &= b_{(i-1)} + \alpha \frac{1}{s} \sum_{i=1}^{s} (y^{(i)} - \tilde{y}^{(i)}) \end{aligned} \tag{16}$$

where s is referred as batch size, which is also a hyper-parameter. When $s = 1$, the iteration process is the same as traditional one, and when $s = M$, all the samples are used to do estimation which is impractical for tasks like where the input is image. So for specific task, one could choose a proper batch size to do gradient estimation and accelerates convergence of learning process. In Machine Learning field, such learning strategy is referred as Stochastic Gradient Descent (SGD) [18].

As the popularity of SGD grows in Machine Learning field, Ning [19] proposed SGD with momentum based on naive gradient descent. What Ning was trying to improve was that the iteration process will oscillate and convergence slowed down as the difference of gradients for each direction grew. To solve that, SGD with momentum accumulates the gradients of each direction to accelerate the learning process. If one component gradient follows the same direction, the accumulated gradient will result in a larger stride and contributes to convergence.

We use momentum to accumulate the gradients in (16), and update weights with the momentum:

$$\begin{aligned} \boldsymbol{v}_{(i)} &= \mu \cdot \boldsymbol{v}_{(i-1)} + \alpha \frac{1}{s} \sum_{i=1}^{s} (y^{(i)} - \tilde{y}^{(i)})\boldsymbol{x}^{(i)} \\ \boldsymbol{\theta}_{(i)} &= \boldsymbol{\theta}_{(i-1)} + \boldsymbol{v}_{(i)} \end{aligned} \tag{17}$$

where μ is momentum, which is another hyper-parameter and is set as 0.9 usually, and when $\mu = 0$, it degrades to common gradient descent. The interval variable $\boldsymbol{v}$ is initialized 0. Likewise, the iteration for bias b can be inferred. However, for the first order gradient descent to work properly, one has to do several experiments to choose the proper learning rate and other hyper-parameters, which has the uncertainty to achieve a good result.

C. Second Order Gradient Descent

With (14), we learned that the learning objective is to achieve the minimum of the objective function F. And for the minimum of F, the derivative should satisfy $\partial F / \partial \boldsymbol{\theta} = \mathbf{0}$. Then we could solve the equation where the partial derivative is 0 to find out the proper weights. for finding successively better approximations of zeros of a function, we could apply Newton's method [18].

For function $f(\boldsymbol{x})$ and variable $\boldsymbol{x}^{(0)}$, we Taylor expand function $f(\boldsymbol{x})$ as:

$$\begin{aligned} f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) &+ (\boldsymbol{x} - \boldsymbol{x}^{(0)})^T \boldsymbol{g} \\ &+ \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^T \boldsymbol{H} (\boldsymbol{x} - \boldsymbol{x}^{(0)}) \end{aligned} \tag{18}$$

where the $\boldsymbol{g}$ is the first order gradient of f with respect to $\boldsymbol{x}$, and $\boldsymbol{H}$ is Hessian matrix, which represents the second order gradient.

Then take the derivative of $\boldsymbol{x}$ and set to 0.

$$\mathbf{0} = \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = \boldsymbol{g} + \boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x}^{(0)}) \tag{19}$$

Solving (19) to obtain the iteration process for $\boldsymbol{x}$:

$$\boldsymbol{x} = \boldsymbol{x}^{(0)} - \boldsymbol{H}^{-1}\boldsymbol{g} \tag{20}$$

For applying to dynamic model represented with (2) and (4), we combine the weight and bias as $\boldsymbol{\theta} = [\theta_0, \theta_1, \cdots, \theta_{m+n}, b]^T$, and add constant 1 to the end of input as $\boldsymbol{x} = [x_0, x_1, \cdots, x_{m+n}, 1]^T$. Together with (14), we obtain the first and second order gradient as:

$$\begin{aligned} \boldsymbol{g} &= \frac{\partial F}{\partial \boldsymbol{\theta}} = -(y^{(i)} - \tilde{y}^{(i)}) \cdot \boldsymbol{x}^{(i)} \\ \boldsymbol{H} &= \frac{\partial^2 F}{\partial \boldsymbol{\theta}^2} = \boldsymbol{x}^{(i)} \boldsymbol{x}^{(i)^T} \end{aligned} \tag{21}$$

In (21), if the input $\boldsymbol{x}^{(i)}$ is a vector, then the corresponding $\boldsymbol{H}$ is singular matrix which is not invertible. For optimizing

FLANN models, one needs to use mini-batch strategy as discussed in III-B. For batch size s, the input matrix and output vectors are:

$$\boldsymbol{X}^{(i)} = \begin{bmatrix} (\boldsymbol{x}^{(1)})^T \\ (\boldsymbol{x}^{(2)})^T \\ \vdots \\ (\boldsymbol{x}^{(s)})^T \end{bmatrix} \qquad \boldsymbol{y}^{(i)} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(s)} \end{bmatrix} \tag{22}$$

And the estimated output is:

$$\tilde{\boldsymbol{y}}^{(i)} = \boldsymbol{X}^{(i)}\boldsymbol{\theta} \tag{23}$$

The gradients of the mini-batch data are:

$$\begin{aligned} \boldsymbol{g} &= -(\boldsymbol{X}^{(i)})^T(\boldsymbol{y}^{(i)} - \tilde{\boldsymbol{y}}^{(i)}) \\ \boldsymbol{H} &= (\boldsymbol{X}^{(i)})^T\boldsymbol{X}^{(i)} \end{aligned} \tag{24}$$

Along with (24) and (20), the iteration process is:

$$\boldsymbol{\theta}_{(i)} = \boldsymbol{\theta}_{(i-1)} - \boldsymbol{H}^{-1}\boldsymbol{g} \tag{25}$$

From (25) we learn that we have no need to tune the learning rate, and the second order gradient includes the trending of first order gradient which accelerates convergence. However, this method is not widely used in modern deep learning due to the modern neural network models have great amount of parameters which results in a huge Hessian matrix, and computing the inverse of that matrix is impractical. For the dynamic modeling scenario, the order of the model is relatively small which makes the algorithm applicable.

D. Optimize with Normal Equation

Apart from the conventional ways of optimizing objective with gradient descent algorithms, researchers have proposed numerical solution with Support Vector Machine (SVM), where the Lagrangian equations are established to solve the parameters directly [5], [17]. With numerical solution, the computing time could be reduced and it is less prone to get stuck in sub-optimal or oscillating. However, optimizing with SVM needs to solve multi-variable linear equations, which is complicated. Instead, we propose optimizing with normal equation, where the global optimal is achieved by matrix manipulation. Similar to second order gradient descent algorithm, we find the minimum of a function by solving the zeros of first order gradient with only matrix manipulations.

For totally M training samples, the input of ith sample is $\boldsymbol{x}^{(i)} \in \mathbb{R}^{m+n+2}$ (where the constant 1 was added to the end), the output is $y^{(i)} \in \mathbb{R}$, and weight is $\boldsymbol{\theta} \in \mathbb{R}^{m+n+2}$. For convenience, we represent the whole training set with matrix and vector as:

$$\boldsymbol{X} = \begin{bmatrix} (\boldsymbol{x}^{(1)})^T \\ (\boldsymbol{x}^{(2)})^T \\ \vdots \\ (\boldsymbol{x}^{(M)})^T \end{bmatrix} \qquad \boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(M)} \end{bmatrix} \tag{26}$$

The bias of output is:

$$\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y} = \begin{bmatrix} (\boldsymbol{x}^{(1)})^T\boldsymbol{\theta} - y^{(1)} \\ \vdots \\ (\boldsymbol{x}^{(M)})^T\boldsymbol{\theta} - y^{(M)} \end{bmatrix} \tag{27}$$

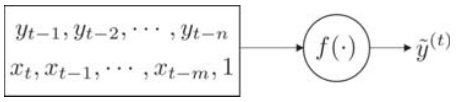

Fig. 2 Learning strategy A

For a vector $\boldsymbol{z}$, we have equation $\boldsymbol{z}^T\boldsymbol{z} = \sum_i z_i^2$, and we substitute $\boldsymbol{z}$ with $\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}$ in (27):

$$\begin{aligned} \frac{1}{2}(\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y})^T(\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}) &= \frac{1}{2}\sum_{i=1}^{M}(y^{(i)} - \boldsymbol{\theta}^T\boldsymbol{x}^{(i)})^2 \\ &= J(\boldsymbol{\theta}) \end{aligned} \tag{28}$$

It is obvious that $J(\boldsymbol{\theta})$ is the same as the objective in (11). And we can solve the zeros of $J(\boldsymbol{\theta})$ with respect to weight parameter $\boldsymbol{\theta}$ to find the optimal solution for (11).

Following the properties of trace of matrix and matrix derivation, the $\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$ can be derived as:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}) &= \frac{1}{2}\nabla_{\boldsymbol{\theta}}\mathrm{tr}(\boldsymbol{\theta}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{\theta}^T\boldsymbol{X}^T\boldsymbol{y} \\ &\quad - \boldsymbol{y}^T\boldsymbol{X}\boldsymbol{\theta} + \boldsymbol{y}^T\boldsymbol{y}) \\ &= \frac{1}{2}\nabla_{\boldsymbol{\theta}}(\mathrm{tr}\boldsymbol{\theta}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\theta} - 2\mathrm{tr}\boldsymbol{y}^T\boldsymbol{X}\boldsymbol{\theta}) \\ &= \frac{1}{2}(2\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\theta} - 2\boldsymbol{X}^T\boldsymbol{y}) \end{aligned} \tag{29}$$

Then, we set $\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}) = \mathbf{0}$, and obtain the final solution for normal equation:

$$\boldsymbol{\theta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{30}$$

With (30), we have no need to solve linear equations, and the computing process is faster. Most of all, it is less prone to stuck in local optimal.

IV. General Sequence Model

During the process of optimizing FLANN model with gradient descent algorithm, we have two kinds of learning strategies depending on whether taking the i-th output as the input for $i+1$-th iteration and iterations after. As shown in Figs. 2 and 3.

$f(\cdot)$ in Fig. 2 is linear operation $f(\boldsymbol{x}) = \boldsymbol{\theta}^T\boldsymbol{x}$, where the input vector is $\boldsymbol{x}^{(t)} = [y^{(t-1)}, y^{(t-2)}, \cdots, y^{(t-n)}, x^{(t)}, x^{(t-1)}, \cdots, x^{(t-m)}, 1]^T$, and the corresponding output is $\tilde{y}^{(t)}$. One should note that the outputs $\{y^{(t-i)}\}$ before time step t are the ground truth of each time step, and it is used in the conventional learning strategy. Meanwhile, when using the ground truth as the input, there is no strict order between each iteration, which makes it possible to randomly select one or more samples to do updating. We will discuss the effect of randomness during sampling in the following experiments.

The learning strategy B in Fig. 3 has an extra step of input updating represented by symbol "+". When $t = 1$, do initialization as $y^{(-1)} = y^{(-2)} = \cdots = y^{(-n)} = 0$, $x^{(-1)} = x^{(-2)} = \cdots = x^{(-m)} = 0$ and after the linear operation $f(\cdot)$, the output is $\tilde{y}^{(1)}$. when $t = 2$, provided the estimated output of previous step $\tilde{y}^{(1)}$ and the input of current time step, we update the input vector as $\boldsymbol{x}^{(2)} = [\tilde{y}^{(1)}, y^{(-1)}, \cdots, y^{(-n-1)}, x^{(2)}, x^{(1)}, x^{(-1)}, \cdots, x^{(-m-1)}, 1]^T$.

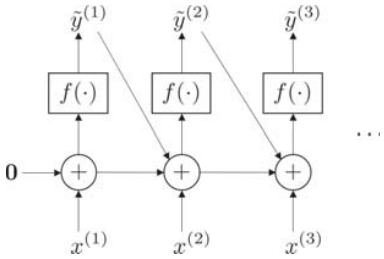

Fig. 3 Learning strategy B

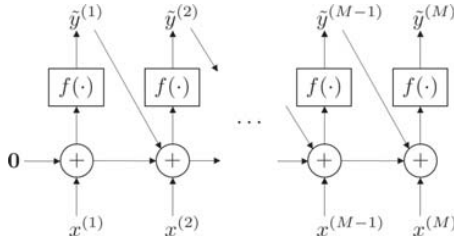

Fig. 4 General sequence model

Generally, at time step t, the updated input vector is $\boldsymbol{x}^{(t)} = [\tilde{y}^{(t-1)}, \tilde{y}^{(t-2)},\cdots,\tilde{y}^{(t-n)},x^{(t)}, x^{(t-1)}]$. It is obvious that this strategy imposes restriction on the order of iteration, and when deriving the gradient of output difference at time step t with respect to $\boldsymbol{\theta}$, the gradient of previous timesteps should also be considered as $\{\tilde{y}^{(t-i)}\}$ is also a function of $\boldsymbol{\theta}$.

Wu et al. [6] proposed an improved algorithm based on the learning strategy B, and obtained a more accurate gradient estimation.

$$\frac{\partial \tilde{y}^{(i)}}{\partial \boldsymbol{\theta}} = \boldsymbol{x}^{(i)} + \sum_{j=0}^{n-1} \boldsymbol{\theta}_j^{(i-1)} \frac{\partial \tilde{y}^{(i-j-1)}}{\partial \boldsymbol{\theta}} \tag{31}$$

where the $\boldsymbol{\theta}_j^{(i-1)}$ is j-th element of weight vector after $i-1$ times iteration.

For (31), the improved algorithm only considered gradient of current step and n steps before that. However, gradient estimation of $\tilde{y}^{(i-j)}$ is also relevant to $\tilde{y}^{(i-j-1)}\cdots\tilde{y}^{(i-j-n)}$. Generally, all the gradients of the first step to the current step should be considered, and it does not depend on the order of model n. Then we proposed the General Sequence Model as shown in Fig. 4.

Fig. 4 represents the situation where the input sequence has length M. And $\{\tilde{y}^{(i)}\}$, $\{x^{(i)}\}$ is also initialized to 0. After adding square of bias for each step we obtain the same loss function as (28):

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{T}(y^{(i)} - \tilde{y}^{(i)})^2 \tag{32}$$

And the corresponding gradient estimation is:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -\sum_{i=1}^{M}(y^{(i)} - \tilde{y}^{(i)})\frac{\partial \tilde{y}^{(i)}}{\partial \boldsymbol{\theta}} \tag{33}$$

We could keep iterating with (31) to obtain the gradient $\partial\tilde{y}^{(k)}/\partial\boldsymbol{\theta}$ for any step $k \in [1, M]$. Finally, the gradient of the whole sequence could be obtained by substituting gradient of each step into (33).

As for the optimizing process, we also use back-propagation to estimate the gradients. To avoid the gradient vanishing or exploding problem caused by long sequence data, one would choose a proper length as the special case in the work of Wu et al. [6], where the length is set to n. The process of optimizing sequence data is referred as back-propagation through time (BPTT) [18] in deep learning. Meanwhile, modeling nonlinear model is possible by altering the updating strategy of inputs and transform function $f(\cdot)$, and it could be extended further to recurrent and recursive networks [18].

V. Experiments and Analysis

Among some of the researches about this topic, the ability of the learned model is evaluated on the training set and the generalization on new data is neglected. Wang et al. [5] took the training set without noise as the validation set, but these two datasets are quiet similar to each other. However, the data could be differ a lot in the real world scenario.

During the following experiments, the system is simulated with the same linear system, whose differential equation is:

$$\begin{aligned} y^{(i)} - 1.0y^{(i-1)} - 0.4y^{(i-2)} + 0.6y^{(i-3)} &= 0.3x^{(i)} \\ + 0.2x^{(i-1)} - 0.1x^{(i-2)} + 0.3x^{(i-3)} + e^{(i)} + 0.0 \end{aligned} \tag{34}$$

where $e^{(i)}$ is the Gaussian noise, the distribution is $e^{(i)} \sim \mathcal{N}(0, 0.01)$.

The excitation is step signal added with Gaussian noise, which has mean 0, and standard variance 0.001, and the time expanding is 500 ms, sampling interval is 1 ms. Meanwhile, we use the learning strategy A to do iteration, and the condition of convergence is the loss over the whole set with (32) is less than a threshold, or the difference of loss between adjacent iterations $\Delta J(\boldsymbol{\theta})$ is less than $\epsilon = 0.0001$. Finally, we evaluate the model with square signal and signal collected from real sensor.

A. Influence of Batch Size

Firstly, the experiment compares the impact of batch size on the training process under the same conditions. With experimental setting mentioned above, the learning rate is set to $\alpha = 0.01$, and the batch size iterates over $s \in [1, 5, 10, 50]$. The learning curve on training set and validation set are shown in Fig. 5 and Fig. 6, respectively.

From the result of the experiment, batch learning helps to increase the accuracy of gradient estimation, and accelerates the convergence. Most of all, it enhances the ability of the model and achieves a smaller loss on the validation set. However, with the batch size increasing, the acceleration seems to saturate. The possible explanation is that the parameter of the model is small, and the gradient is accurately enough with small batch size.

B. Influence of Random Sampling

As we choose the learning strategy A, it is possible to sample randomly. Then we investigate the impact of shuffling

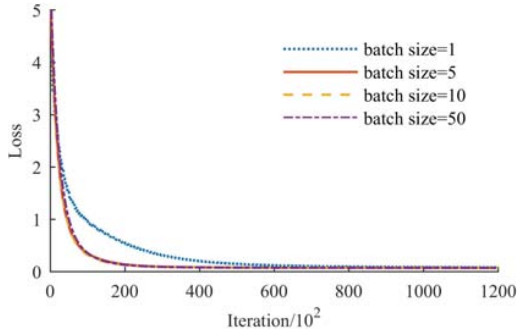

Fig. 5 Learning curve on training set

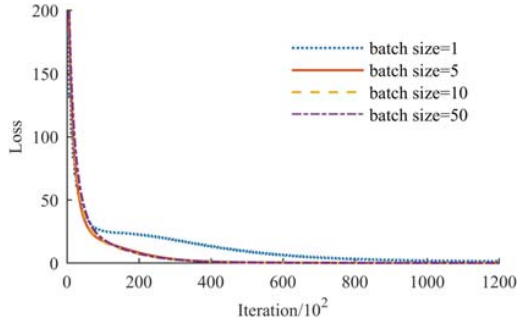

Fig. 6 Learning curve on validation set

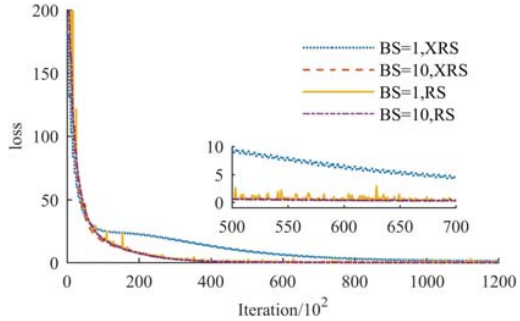

Fig. 7 Impact of shuffling training data

training data, and the learning curve of random sample and sampling in order is shown in Fig. 7, where "RS" stands for random sample and "XRS" represents sampling following the order.

The result indicates that, when iteration over one sample, the data shuffling contributes to faster convergence and enhancement of model ability, which is similar to batch learning. Meanwhile, for the curve "BS=1, RS" in partial enlarged view, it has the phenomenon of bouncing up and down, which proves that the inaccuracy of gradient estimation with only one sample.

C. SGD with Momentum

To investigate the effectiveness of momentum learning, we set the parameter momentum as $\mu = 0.85$, and conduct experiments with and without momentum. The result learning curve in shown in Fig. 8.

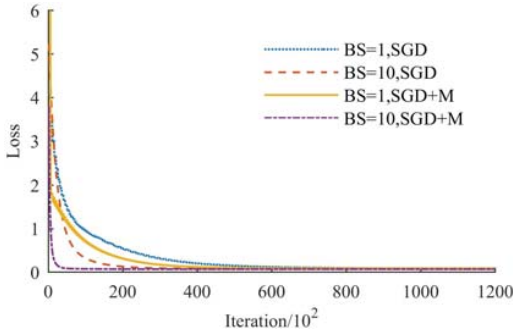

Fig. 8 Comparison of SGD and SGD with momentum

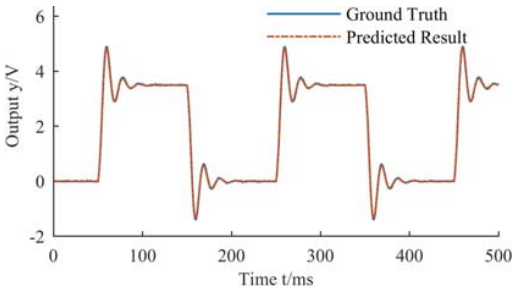

Fig. 9 Modeling result of SGD with momentum algorithm

The result shows that optimization with SGD and momentum greatly accelerates convergence process, which is much faster than other methods. And the resulting parameters are:

$$\boldsymbol{\theta} = [1.0518, 0.2985, -0.5453, 0.3034, \\ 0.1766, -0.0665, 0.2678, -4.88 \times 10^{-4}]$$

The loss of the excitation with square signal is $J(\boldsymbol{\theta}) = 0.1072$, and the comparison of original dynamic response and that of the learned model is shown in Fig. 9.

D. Second Order Gradient Descent

To evaluate the effectiveness of second order gradient descent, we need to choose the proper batch size to enable the inverse of Hessian matrix in (24), and all the inputs are randomly sampled. Meanwhile, we also compared the different parameter initialization with initialized to 0 or normal distribution of $\mathcal{N}(0, 0.01)$. The condition of convergence is the loss of the whole training set satisfies $J(\theta) < 0.1$. The comparison with the naive gradient descent is shown in Fig. 10, where "RI" means random initialization, "M" means optimized with momentum and "SOG" is second order gradient descent.

The experiment indicates that the iteration of second order gradient descent is much faster than that of naive gradient descent. It achieves a reasonably small loss after the first iteration. However, the further iteration process is not stable. Along with the advantages of first and second order gradient descent, we could try to use the result of second order gradient descent after the first iteration to initialize the weights, then following the naive gradient descent to optimize the

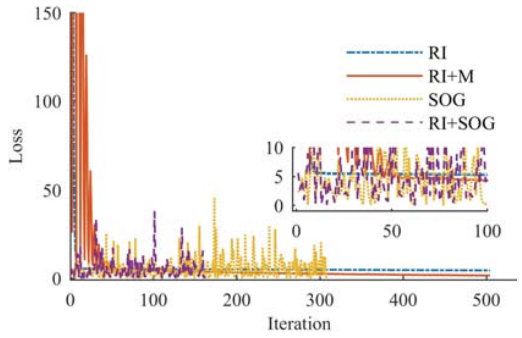

Fig. 10 Second order gradient descent algorithm

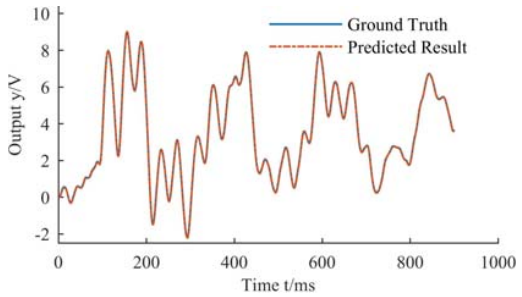

Fig. 11 Modeling result of second order gradient descent algorithm

objective. The experiment shows that combining those two algorithm leads to a faster convergence than conventional gradient descent. The modeling result of second order gradient descent on the output signal of a dynamic weighting system is shown in Fig. 11.

E. Optimization with Normal Equation

Optimization algorithm based on gradient descent need to tune hyper-parameters like learning rate to achieve a better result, and it takes time to convergence. However, for dynamic model based on FLANN, we can utilize (30) to directly solve the optimal parameter. The experiment shows that the process is extremely fast with small dataset, and the result is comparably good as that of gradient descent. With the same dataset as previous experiment, the modeling result of optimization with normal equation is shown in Fig. 12, and the weight parameters are:

$$\boldsymbol{\theta} = [1.0491, 0.2967, -0.5419, 0.3029, \\ 0.1781, -0.0660, 0.2722, -5.75 \times 10^{-4}]$$

The loss on the excitation of square signal is $J(\boldsymbol{\theta}) = 0.1644$, which is close to the best of gradient based algorithms. For different training size, the loss on validation set and computing time is shown in Fig. 13, where the experiments are conducted on computer with i5 CPU.

The experiment indicates that the ability of model and size of training set has positive correlation. While the size of dataset is small, the computing time is relatively close to each other. The comprehensive analysis reveals that optimization with normal equation is the best choice of solving models with differential equation.

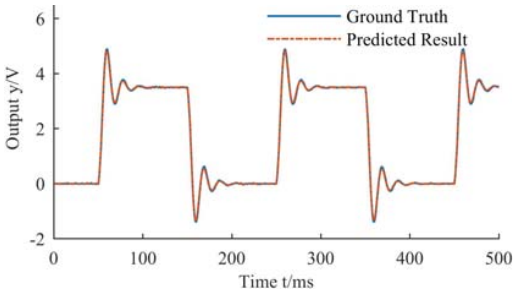

Fig. 12 Modeling result with normal equation

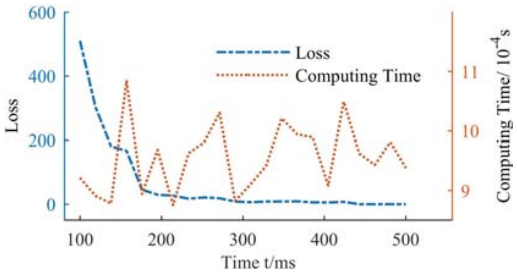

Fig. 13 Impact on modeling result and computing time with training set size increasing

VI. Conclusion

For linear system with single-input and single-output, it can be modeled with FLANN, and different order of the model is chosen to satisfy accuracy requirements. Experiments reveal that FLANN model is suitable for both dynamic modeling and compensation with different source as the input. Based on the existing model research, an optimized gradient descent algorithm is proposed. Meanwhile, with batch learning and momentum optimization methods, the convergence would be greatly accelerated and the generalization ability of the learned model is enhanced. And the convergence is even faster with second order gradient algorithm. On the other hand, the model optimization can be transformed into solving the minimum value, and normal equation is used to directly solve the weight parameter, which is faster and more stable comparing to gradient based algorithms. Experimental simulations indicate that the algorithms we discussed all achieved model identification with reasonable precision. Considering both computing speed and performance, optimization with normal equation is the optimal solution. Finally, we also propose the generalized model for sequence data based on the different learning strategy.

References

[1] J. P. Xiang, "Dynamic properties of force transducer," *Process Automation instrumentation*, no. 6, pp. 12–21+102, 1981.
[2] A. Simpkins, "System identification: Theory for the user, 2nd edition," *IEEE Robotics Automation Magazine*, vol. 19, no. 2, pp. 95–96, June 2012.
[3] K. J. Xu and M. Yin, "A dynamic modeling method based on flann for wrist force sensor," *Chinese Journal of Scientific Instrument*, vol. 21, no. 1, pp. 92–94, 2000.
[4] S. P. Tian, P. P. Jiang, and G. Z. Yan, "Application o-f recurrent neural network to dynamic modeling of sensors," *Chinese Journal of Scientific Instrument*, vol. 25, no. 5, pp. 574–576, 2004.

[5] X. D. Wang, C. J. Zhang, and H. R. Zhang, "Sensor dynamic modeling using least square support vector machines," *Chinese Journal of Scientific Instrument*, vol. 27, no. 7, pp. 730–733, 2006.
[6] D. H. Wu, W. Zhao, S. L. Huang, and S. K. He, "Research on improved flann for sensor dynamic modeling," *Chinese Journal of Scientific Instrument*, no. 2, pp. 362–367, 2009.
[7] W. J. Yang, "Research on dynamic characteristics and compensation technology of pressure sensors," Master's thesis, North University of China, Shanxi, 2017.
[8] T. Dabóczi, "Uncertainty of signal reconstruction in the case of jittery and noisy measurements," *IEEE Transactions on Instrumentation and Measurement*, vol. 47, no. 5, pp. 1062–1066, 1998.
[9] J. Brignell, "Software techniques for sensor compensation," *Sensors and Actuators A: Physical*, vol. 25, no. 1-3, pp. 29–35, 1990.
[10] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.
[11] S. P. Tian, Y. Zhao, W. H. Yu, and Z. W. Wang, "Nonlinear compensation of sensors based on bp neural network," *Journal of Test and Measurement Technology*, vol. 21, no. 1, pp. 84–89, 2007.
[12] She Ping Tian, "Nonlinear dynamic compensation of sensors based on recurrent neural network model," *Journal of Shanghai Jiao Tong University*, vol. 37, no. 1, pp. 13–16, 2003.
[13] H. M. Huang, "Dynamical compensation method for weighting sensor based on flann," *Transducer and Microsystem Technologies*, vol. 25, no. 8, pp. 25–28, 2006.
[14] L. Q. Hou, W. G. Tong, and T. X. He, "Nonlinear errors correcting method of sensors based on rbf neural network," *Journal of Transducer Technology*, vol. 17, no. 4, pp. 643–646, 2004.
[15] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.
[16] H. Li, *Statistical learning method*. Beijing: Tsing Hua University Press, 2012.
[17] D. H. Wu, "Dynamic compensating method for transducer based on flann inverse system constructed by ls-svm," *Journal of Data Acquisition and Processing*, vol. 22, no. 3, pp. 378–383, 2007.
[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
[19] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145–151, 1 1999.

Changqiao Wu received his B.Sc. degree in 2016 from Huazhong University of Science and Technology. Now he is a Master in Shanghai Jiao Tong University. His main research interests include intelligent measurement and control.