An Application of Component-Wise Iterative Optimization to Feed-Forward Neural Networks

Yachen Lin

Fidelity National Information Services, Inc.
11601 Roosevelt Blvd-TA76
Saint Petersburg, FL 33716
yachen.lin@fnf.com

Abstract. Component-wise Iterative Optimization (CIO) is a method of dealing with a large data in the OLAP applications, which can be treated as the enhancement of the traditional batch version methods such as least squares. The salient feature of the method is to process transactions one by one, optimizes estimates iteratively for each parameter over the given objective function, and update models on the fly. A new learning algorithm can be proposed when applying CIO to feed-forward neural networks with a single hidden layer. It incorporates the internal structure of feed-forward neural networks with a single hidden layer by applying the algorithm CIO in closed-form expressions to update weights between the output layer and the hidden layer. Its optimally computational property is a natural consequence inherited from the property of the algorithm CIO and is also demonstrated in an illustrative example.

1 Introduction

In recent years, the development of technology has paved the way for the industry to use more sophisticated analytics for making business decisions in on-line analytical processing (OLAP). In the check and credit card processing business, for example, the advanced artificial intelligence is widely used to authorize transactions. This is mainly due to the fact that the fraud activities nowadays are more and more aggressive than ever before. Once certain forms of fraud were identified or caught by the industry, a new form will be intercepted in a very short period of time. People are convinced that fighting fraud requires a non-stopping effort, i.e. constantly updating fraud pattern recognition algorithms and timely incorporating new fraud patterns in the algorithm development process.

The traditional method of least squares presents a challenge for updating the model on the fly of transactions in both a general linear model and a general non-linear model. This challenge is mainly due to the matrix manipulation in the implementation of the computation.

In dealing with the challenge, both industry and academic researchers have made substantial efforts. The most successful stories, however, are for the linear case only and mainly for the improvement of computational properties of least squares. Since the method known as recursive least squares - RLS was derived; many variants of

Y. Shi et al. (Eds.): ICCS 2007, Part II, LNCS 4488, pp. 470–477, 2007.

RLS have been proposed for a variety of applications. A method so called a sliding window RLS was discussed in many papers such as [2] and [3]. By applying QR decomposition, U-D factorization, and singular value decomposition (SVD), more computationally robust implementations of RLS have been discussed in papers such as [1]. Certainly, these researches have substantially increased the computational efficiency of RLS, of cause LS algorithms, but they are limited for the linear case only. Furthermore, the core ingredient in the computation of the algorithm of RLS and its variants are still a matrix version, although some matrices are more re-usable this time in the implementation.

In view of the difficulties that traditional least squares have when updating models on the fly of transactions, a new procedure – Component-wise Iterative Optimization (CIO) was proposed in [10]. Using the new method of CIO, updating models on the fly of transactions becomes straightforward for both linear and non-linear cases. More importantly, the method itself yields an optimal solution with the objective function of sum of squares, in particular, least square estimates when a general linear model is used. The method CIO can be described as follows.

Let X be a new pattern, F be the objective function, and $E = (e_1^{(0)}, \cdots, e_p^{(0)})^t \in \Theta^p$ be the initial estimates, where Θ^p is the domain of the parameter vector. Given the notation below

$$e_1^{(1)} = \underset{e_1 \in \Theta}{Arg\_OP} \; F\,(e_1, e_2^{(0)}, \cdots, e_p^{(0)}, X),$$

then $e_1^{(1)}$ is an optimal solution for the parameter of e_1 over the objective function F given the sample pattern of X and $e_2, \ldots, e_p$ being held fixed.

With the above given notations, the method CIO can be described by the following procedure. Given the initial estimates of $E = (e_1^{(0)}, \cdots, e_p^{(0)})^t \in \Theta^p$, the method of CIO updates the estimates in p steps below:

Step 1. Compute $e_1^{(1)} = \underset{e_1 \in \Theta}{Arg\_OP} \; F\,(e_1, e_2^{(0)}, \cdots, e_p^{(0)}, X)$

Step 2. Compute $e_2^{(1)} = \underset{e_2 \in \Theta}{Arg\_OP} \; F\,(e_1^{(1)}, e_2, e_3^{(0)} \cdots, e_p^{(0)}, X)$ by substituting $e_1^{(1)}$ for $e_1^{(0)}$

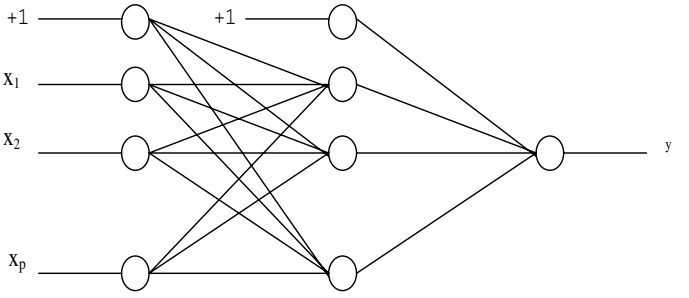
…

Step p. Compute $e_p^{(1)} = \underset{e_p \in \Theta}{Arg\_OP} \; F\,(e_1^{(1)}, ..., e_{P-1}^{(1)}, e_p, X)$ by substituting $e_k^{(1)}$ for $e_k^{(0)}$, $k = 1, 2, ...p-1$.

After these steps, the initial estimates $(e_1^{(0)}, \cdots, e_p^{(0)})^t$ can be updated by $(e_1^{(1)}, \cdots, e_p^{(1)})^t$.

The idea of CIO is no stranger to one who is familiar with the Gibbs sampler by [6]. It is no harm or may be easier to understand the procedure of CIO in terms of the non-Bayesian version of Gibbs sampling. The difference is that CIO generates optimal estimates for a given model and the Gibbs sampler or MCMC generates samples from some complex probability distribution. It has been shown in [10] that the procedure of CIO converges.

How can the algorithm CIO be applied in the neural network field? It is well known that a useful learning algorithm is developed for generating optimal estimates based on specific neural network framework or structure. Different forms and structures of f represent the different types of neural networks. For example, the architecture of the feed-forward networks with a single hidden layer and one output can be sketched in the following figure:



However, too many learning algorithms proposed in the literatures were just ad hoc in nature and their reliability and generalization were often only demonstrated on applications for limited empirical studies or simulation exercises. Such analyses usually emphasize on a few successful cases of applications and did not necessarily establish a solid base for more general inferences. In the study of this kind, much theoretical justification is needed. In the present study, a useful learning algorithm can be derived from a well established theory of the algorithm CIO. To establish the link between CIO and neural networks, we can look at any types of neural networks in a view of a functional process. A common representation of the processing flow based on neural networks can be written as follows:

$$y_i = f(x_{ij}, w_j, j = 0, 1, ..., p) + \varepsilon_i \tag{1.1}$$

where ε_i is an error random term, $i = 1,2,\cdots,n$, and $w = (w_0, w_1, \cdots, w_p)^t$ is the parameter vector.

Information first is summarized into $\alpha^t X_i$, where $X_i = (1, x_{1i}, ..., x_{pi})^t$, and $\alpha_i = (\alpha_{0i}, ..., \alpha_{pi})^t \in \Theta \subseteq R^{p+1}$ and $\alpha = [\alpha_1, \alpha_2, ..., \alpha_m]$. Secondly it is activated by a function ϕ in the way of $\Phi(\alpha^t X_i) = (1, \phi(\alpha_1^t X_i), ..., \phi(\alpha_1^t X_i))^t$. Then it is further summarized in the

output node by applying new weights to the activation function, i.e. $\beta^t \Phi(\alpha^t X_i)$, where $\beta = (\beta_0, \beta_1, ..., \beta_m)^t \in R^{m+1}$. Thus,

$$f = \beta^t \Phi(\alpha^t X_i).$$

From the above discussion, we can see that feed-forward neural network with a single hidden layer fit into the general modeling frame work of (1.1) very well. Only in this setting, the targeted or response variable Y can be expressed by the function $f = \beta^t \Phi(\alpha^t X_i)$. Therefore, one can image that the algorithm CIO can be applied to feed-forward neural networks with a single hidden layer because the function $f = \beta^t \Phi(\alpha^t X_i)$ is a special case of (1.1).

The rest of the paper is structured in the following way: In the next section, we discuss an application of CIO to feed-forward neural networks with a single hidden layer. Certainly, a new learning algorithm originated from the application of CIO will be discussed in a relatively detailed fashion there. At the final, an illustration example by using the new learning algorithm is given.

2 An Application of CIO to Feed-Forward Neural Networks

Training neural networks to reveal the proper pattern in a data set is a non-trivial task. The performance of a network is often highly associated with the effectiveness of a training algorithm. The well-known back-propagation algorithm [12] is a popular approach to train a multi-layer perceptron, i.e. feed-forward networks by minimizing the square errors. Some of its properties have been studied through a number of applications. With the development of high power computing equipment, many alternative algorithms were proposed such as the development of second-order learning algorithm and classical approaches of Gauss-Newton and Newton-Raphson. [8] gave a learning algorithm by using the projection pursuit technique for optimizing one node at a time. The approach was further developed in [7]. Some other existing algorithms in optimization approaches were also used. The comparisons of these algorithms have been conducted for some cases in [13]. [9] proposed a learning algorithm by using Hessian matrix in the update recursive formula - a variation of the second-order learning algorithm.

In training feed-forward neural networks with a single hidden layer, its special structure for the processing flow can be exploited. From the discussion in section 1, we know that the output can be expressed in the following model:

$$y_i = \beta^t \Phi(\alpha^t X_i) + \varepsilon_i, \tag{2.1}$$

where $\Phi(\alpha^t X_i) = (1, \phi(\alpha_1^t X_i), ..., \phi(\alpha_1^t X_i))^t$ and ε_i is an error random term, $i = 1, 2, \cdots, n$. If the objective function of the mean squared errors is given, then training the neural network is equivalent to finding least square estimates of the weight parameters.

Since (2.1) is only a special form of (1.1), given the objective function of the mean squared errors, we can apply the algorithm CIO to train the network. There are two options: (a) directly to apply CIO to (2.1), and (b) only apply CIO to $\beta = (\beta_0, \beta_1, ..., \beta_m)^t \in R^{m+1}$, the weight parameters vector between the hidden layer and the output layer. Considering the condition of the theorem for CIO, we only discuss option (b) in this paper.

Simple computations for (2.1) based on CIO lead to $g'(\beta_j^{(k)}(i)) = \phi(\alpha_j^t X_i)$. Thus, updating for the estimates of $\beta = (\beta_0, \beta_1, ..., \beta_m)^t \in R^{m+1}$, under the condition that given $\alpha = [\alpha_1, \alpha_2, ..., \alpha_m], \alpha_i = (\alpha_{0i}, ..., \alpha_{pi})^t \in \Theta \subseteq R^{p+1}$, can be done simply following the procedure below.

Step 0. Initial value: Choose an initial value of $\beta^{(0)} = (\beta_0^{(0)}, \beta_1^{(0)}, \cdots, \beta_p^{(0)})^t$ for β randomly or by certain optimal rules such as least squares.

Step 1. Compute $\beta^{(1)} = (\beta_0^{(1)}, \beta_1^{(1)}, \cdots, \beta_p^{(1)})^t$:

Compute $\beta_0^{(1)}$: Given a sample pattern $(y_i, x_{i1}, ..., x_{ip})$, we can find the solution of the equation $y_i - g(\beta_0) = 0$, and denoted by $\beta_0^{(1)}(i)$. Repeat n times for all sample patterns $(y_i, x_{i1}, ..., x_{ip}), i = 1, ..., n$, then let

$$\beta_0^{(1)} = \frac{1}{n} \sum_i^n \beta_0^{(1)}(i).$$

Compute $\beta_1^{(1)}$: First, substitute $\beta_0^{(1)}$ for $\beta_0^{(0)}$ in function of $g(\beta_1)$, where $g(\beta_1) = f(\beta_1, given\ x_{ij}, \beta_0^{(1)}, \beta_k^{(0)}, k \neq 0, 1,\ k = 0,1,..., p)$. Then, solve the equation $y_i - g(\beta_1) = 0$, and the solution is denoted by $\beta_1^{(1)}(i)$ for given sample pattern $(y_i, x_{i1}, ..., x_{ip})$. Repeat n times for all sample patterns $(y_i, x_{i1}, ..., x_{ip}), i = 1, ..., n$, then let

$$\beta_1^{(1)} = \sum_{i=1}^n \frac{\left(\phi(\alpha_1^t X_i)\right)^2}{\sum_1^n \left(\phi(\alpha_1^t X_i)\right)^2} \beta_1^{(1)}(i).$$

(p) Compute the last component $\beta_p^{(1)}$: By the p-1 steps above, components of β are taken as $\beta_l^{(1)}, l = 0,1,..., p-1$ in the function of $g(\beta_p)$. Then, solve the equation $y_i - g(\beta_p) = 0$, and the solution is denoted by $\beta_p^{(1)}(i)$ for given

sample patterns $(y_i, x_{i1},..., x_{ip})$, $i = 1,...,n$, and then let

$$\beta_p^{(1)} = \sum_{i=1}^{n} \frac{\left(\phi(\alpha_p^t X_i)\right)^2}{\sum_1^n \left(\phi(\alpha_p^t X_i)\right)^2} \beta_p^{(1)}(i).$$

Step k. Compute $\beta^{(k)} = (\beta_0^{(k)}, \beta_1^{(k)}, \cdots, \beta_p^{(k)})^t$: Repeat the Step 1 k times, then get $\beta^{(k)}$.

Let us denote the above procedure by CIO(β ; α), which means that given α, update β by CIO(β ; α). The other group of weight parameters in the network $\alpha = [\alpha_1, \alpha_2,..., \alpha_m], \alpha_i = (\alpha_{0i},..., \alpha_{pi})^t \in \Theta \subseteq R^{p+1}$ can be updated by one of many commonly used iterative procedures such as Gauss-Newton, Newton-Raphson, and Conjugate Gradient, denoted by CUIP(α; β), which means that given β, update α by CUIP(α; β).

Given the above notations, let Φ be the activation function, *x* be input features, and *y* be the response, the following figure shows the updating procedure.

Algorithm by Neural-CIO (α, β, Φ, *X, Y*)

1. $\alpha^{(\text{old})} \leftarrow \alpha^{(0)}$
2. $\beta^{(\text{old})} \leftarrow \beta^{(0)}$
3. SSR ← Criterion($\alpha^{(0)}$, $\beta^{(0)}$, Φ, *X, Y*)
4. **While** SSR > ε do;
   $\beta^{(\text{new})} \leftarrow$ CIO($\beta^{(\text{old})}$; $\alpha^{(\text{old})}$)
   $\alpha^{(\text{new})} \leftarrow$ CUIP($\alpha^{(\text{old})}$; $\beta^{(\text{new})}$)
   SSR ← Criterion($\alpha^{(\text{new})}$, $\beta^{(\text{new})}$, Φ, *X, Y*)
5. **Return** $\alpha^{(\text{new})}$, $\beta^{(\text{new})}$, SSR

The advantage of function CIO($\beta^{(\text{old})}$; .) over the other available learning algorithm is its closed form, i.e. $\beta^{(\text{new})}$ CIO($\beta^{(\text{old})}$; .). To update the weight parameter vector β, we do not need to apply iterations while updating by the closed form. Therefore, it is more efficient computationally. In the section, we will show this computational efficiency by a numeric example. For the function Criterion($\alpha^{(\text{new})}$, $\beta^{(\text{new})}$, Φ, *X, Y*), it can be of many forms such like the mean squared error, the number of iterations, or the lackness of training progress.

3 An Illustrative Example

This section gives a numeric example of a classification problem using Fisher's famous Iris data. A comprehensive study on applying neural networks to this data set was given in [11]. In the following, both algorithms Neural-CIO and Backpropagation are implemented and compared over the data in the framework of three

feed-forward neural networks with a single hidden layer, 4-2-1, 4-3-1, 4-4-1, i.e. 2, 3, and 4 hidden nodes, respectively.

For the data, a sample of three records of the original data can be seen in the following table.

Table 1. Three records of Fisher's Iris data

Sepal Length	Sepal width	Petal length	Petal width	Class
5.1	3.5	1.4	0.2	Setosa
7.0	3.2	4.7	1.4	Versicolor
6.3	3.3	6.0	2.5	Verginica

All measurements are in centimeters. It is well-known that the class Setosa can be linearly separated from the other two and the other two can not be separated linearly. Thus, we only apply the algorithms to the network to separate these two classes. The data is further transformed using the following formulas and then each record is assigned a class number either 0.0999 or 0.0001.

Sepal length == (Sepal length - 4.5) / 3.5; Sepal width == (Sepal width - 2.0) / 1.5;
Petal length == (Pepal length - 3.0) / 3.5; Petal width == (Pepal width - 1.4) / 1.5;

Table 2. Two records of transformed Fisher's Iris data

Sepal Length	Sepal width	Petal length	Petal width	Class
0.7143	0.8000	0.4857	0.0000	0.0999
0.5143	0.8667	0.4286	0.7333	0.0001

Note: There are total 100 records in the data set and 50 for each class.

In the training process, for both algorithms, the stopping rule is chosen to be the mean squared error less than a pre-assigned number. The activation function is taken the form of $\phi(x) = (1 + e^{-x})^{-1}$.

The training results and performance are summarized in the table in the next page.

The results in the table clearly shows the advantages that the new learning algorithm incorporates the internal structure of feed-forward neural networks with a single hidden layer by applying the algorithm CIO in closed-form expressions to update weights between the output layer and the hidden layer. Its optimally computational property is a natural consequence inherited from the property of the algorithm CIO and this point has been further verified in the above illustrative example.

Table 3. Comparison between Neural-CIO and Back-propagation

		#Misclassification		# Iteration		CPU Time(second)	
Structure	Error	CIO	Back	CIO	Back	CIO	Back
4-2-1	0.0002	6	6	2433	9092	68	93
4-3-1	0.0002	5	6	1935	8665	96	120
4-4-1	0.0002	5	6	811	6120	60	110

Note: CIO means Neural-CIO and Back means Back-propagation.

References

1. Baykal, B., Constantinids, A.: Sliding window adaptive fast QR and QR-lattice algorithm. IEEE Signal Process 46 (11), (1998) 2877-2887
2. Belge, M., Miller, E.: A sliding window RLS-like adaptive algorithm for filtering alpha-stable noise. IEEE Signal Process Letter 7 (4), (2000) 86-89
3. Choi, B., Bien, Z.: Sliding-windowed weighted recursive least-squares method for parameter estimation. Electron Letter 25 (20), (1989) 1381-1382
4. Fisher, R: The use of multiple measurements in taxonomic problems, Ann. Eugencis 7, Pt II, (1939). 197-188
5. Fletcher, R.: Practical Methods of Optimization Vol I: Unconstrained Optimization, Comput. J. 6, (1980). 163 – 168
6. Geman, S., Geman, D.: Stochastic relaxation, Gibbs distribution and Bayesian restoration of images. IEEE Transactions on Pattern Analysis and Machine Intelligence 6, (1984) 721-741
7. Hwang, J. N., Lay, S. R., Maechler, M., and Martin, D. M.: Regression modeling in back-propagation and projection pursuit learning, IEEE Trans. on Neural networks vol. 5, 3, (1994)342 - 353
8. Jones, L. K.: A simple lemma on greedy approximation in Hilbert space and convergence rates for projection pursuit regression and neural network training, Ann. Statist. Vol. 20 (1992) 608- 613
9. Karayiannis, N. B., Venetsanopoulos, A. N.: Artificial Neural Networks: Learning algorithms, Performance evaluation, and Applications KLUWER ACADEMIC PUBLISHERS(1993).
10. Lin, Y.: Component-wise Iterative Optimization for Large Data, submitted (2006)
11. Lin, Y.: Feed-forward Neural Networks – Learning algorithms, Statistical properties, and Applications, Ph.D. Dissertation, Syracuse University (1996)
12. Rumelhart, D. E., Hinton, E. G., Williams, R. J.: Learning internal representations by error propagation, Parallel Distributed Processing. Chap. 8, MIT, Cambridge, Mass. (1986)
13. Watrous, R. L.: Learning algorithms for connectionist networks: applied gradient methods of nonlinear optimization, IEEE First Int. Conf. Neural Networks, San Diego, (1987) II 619-627