Efficient Implementation of Newton-Raphson Methods for Sequential Data Prediction

Burak Cevat Civek and Suleyman Serdar Kozat, *Senior Member, IEEE*

Abstract—We investigate the problem of sequential linear data prediction for real life big data applications. The second order algorithms, i.e., Newton-Raphson Methods, asymptotically achieve the performance of the "best" possible linear data predictor much faster compared to the first order algorithms, e.g., Online Gradient Descent. However, implementation of these second order methods results in a computational complexity in the order of $O(M^2)$ for an M dimensional feature vector, where the first order methods offer complexity in the order of $O(M)$. Because of this extremely high computational need, their usage in real life big data applications is prohibited. To this end, in order to enjoy the outstanding performance of the second order methods, we introduce a highly efficient implementation where the computational complexity of these methods is reduced from $O(M^2)$ to $O(M)$. The presented algorithm provides the well-known merits of the second order methods while offering a computational complexity similar to the first order methods. We do not rely on any statistical assumptions, hence, both regular and fast implementations achieve the same performance in terms of mean square error. We demonstrate the efficiency of our algorithm on several sequential big datasets. We also illustrate the numerical stability of the presented algorithm.

Index Terms—Newton-Raphson, highly efficient, big data, sequential data prediction

1 Introduction

Technological developments in recent years have substantially increased the amount of data gathered from real life systems[1], [2], [3], [4]. There exists a significant data flow through the recently arising applications such as large-scale sensor networks, information sensing mobile devices and web based social networks [5], [6], [7]. The size as well as the dimensionality of this data strain the limits of current architectures. Since processing and storing such massive amount of data result in an excessive computational cost, efficient machine learning and data processing algorithms are needed [1], [8].

In this paper, we investigate the widely studied sequential prediction problem for high dimensional data streams. Efficient prediction algorithms specific to big data sequences have great importance for several real life applications such as high frequency trading [9], forecasting [10], trend analysis [11] and financial market [12]. Unfortunately, conventional methods in machine learning and data processing literatures are inadequate to efficiently and effectively process high dimensional data sequences [13], [14], [15]. Even though today's computers have powerful processing units, traditional algorithms create a bottleneck even for that processing power when the data is acquired at high speeds and too large in size [13], [14].

In order to mitigate the problem of excessive computational cost, we introduce sequential, i.e., online, processing, where the data is neither stored nor reused, and avoid "batch" processing. [15], [16]. One family of the well known online learning algorithms in the data processing literature is the family of first order methods, e.g., Online Gradient Descent [17], [18]. These methods only use the gradient information to minimize the overall prediction cost. They achieve logarithmic regret bounds that are theoretically guaranteed to hold under certain assumptions [17]. Gradient based methods are computationally more efficient compared to other families of online learning algorithms, i.e., for a sequence of M-dimensional feature vectors $\{\boldsymbol{x}_t\}_{t\geq 0}$, where $\boldsymbol{x}_t \in \mathbb{R}^M$, the computational complexity is only in the order of $O(M)$. However, their convergence rates remain significantly slow when achieving an optimal solution, since no statistics other than the gradient is used [3], [15], [18]. In most big data applications, the first order learning algorithms are adopted due to their low computational demands [19]. However, it is possible to obtain outstanding performance using the second order methods [15].

Different from the gradient based algorithms, the well known second order Newton-Raphson methods, e.g, Online Newton Step, use the second order statistics, i.e., Hessian of the cost function [17]. Hence, they asymptotically achieve the performance of the "best" possible predictor much faster[16]. Existence of logarithmic regret bounds is theoretically guaranteed for this family of algorithms as well [17]. Additionally, the second order methods are robust and prone to highly varying data statistics, compared to the first order methods, since they keep track of the second order information [16], [20]. Therefore, in the sense of convergence rate and steady state error performances, Newton-Raphson methods considerably outperform the first order methods [15], [16], [18]. However, the second order methods offer a quadratic computational complexity, i.e., $O(M^2)$, while the gradient based algorithms provide a linear relation, i.e., $O(M)$. As a consequence, it is not usually feasible for real-life big data applications to utilize the merits of the second order algorithms [19].

In this paper, we study sequential data prediction, where the consecutive feature vectors are the shifted versions of each other, i.e., for a feature vector of $\boldsymbol{x}_t = [x_t, x_{t-1}, \ldots, x_{t-M}]^T$, the upcoming vector is in the form of $\boldsymbol{x}_{t+1} = [x_{t+1}, x_t, \ldots, x_{t-M+1}]^T$. To this end, we introduce second order methods for this important problem with computational complexity only linear in the data dimension, i.e., $O(M)$. We achieve such an enormous reduction in computational complexity since there are only two entries changing from $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t+1}$, where we avoid unnecessary calculations in each update. We do not use any statistical assumption on the data sequence other than the shifted nature of the feature vectors. Therefore, we present an approach that is highly appealing for big data applications since it provides the merits of the Newton-Raphson methods with a much lower computational cost.

Overall, in this paper, we introduce an online sequential data prediction algorithm that i) processes only the currently available data without any storage, ii) efficiently implements the Newton-Raphson methods, i.e., the second order methods iii) outperforms the gradient based methods in terms of performance, iv) has $O(M)$ computational complexity same as the first order methods and v) requires no statistical assumptions on the data sequence. We illustrate the outstanding gains of our algorithm in terms of computational efficiency by using two sequential real life big datasets and compare the resulting error performances with the regular Newton-Raphson methods.

2 Problem Description

In this paper, all vectors are real valued and column-vectors. We use lower case (upper case) boldface letters to represent vectors (matrices). The ordinary transpose is denoted as $\boldsymbol{x}^T$ for the vector $\boldsymbol{x}$. The identity matrix is represented by $\boldsymbol{I}_M$, where the subscript is used to indicate that the dimension is $M \times M$. We denote the M-dimensional zero vector as $\boldsymbol{0}_M$.

• *The authors are with the Department of Electrical and Electronics Engineering, Bilkent University, Ankara 06800, Turkey.*
E-mail: {burak, kozat}@ee.bilkent.edu.tr.

Manuscript received 23 Oct. 2016; revised 24 May 2017; accepted 4 Sept. 2017. Date of publication 19 Sept. 2017; date of current version 3 Nov. 2017.
(Corresponding author: Burak Cevat Civek.)
Recommended for acceptance by T. Li.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2017.2754380

We study sequential data prediction, where we sequentially observe a real valued data sequence $\{x_t\}_{t\geq 0}$, $x_t \in \mathbb{R}$. At each time t, after observing $\{x_t, x_{t-1}, \ldots, x_{t-M+1}\}$, we generate an estimate of the desired data, $\hat{x}_{t+1} \in \mathbb{R}$, using a linear model as

$$\hat{x}_{t+1} = \boldsymbol{w}_t^T \boldsymbol{x}_t + c_t, \tag{1}$$

where $\boldsymbol{x}_t \in \mathbb{R}^M$ represents the feature vector of previous M samples, i.e., $\boldsymbol{x}_t = [x_t, x_{t-1}, \ldots, x_{t-M+1}]^T$. Here, $\boldsymbol{w}_t \in \mathbb{R}^M$ and $c_t \in \mathbb{R}$ are the corresponding weight vector and the offset variable respectively at time t. With an abuse of notation, we combine the weight vector $\boldsymbol{w}_t$ with the offset variable c_t, and denote it by $\boldsymbol{w}_t = [\boldsymbol{w}_t; c_t]$, yielding $\hat{x}_{t+1} = \boldsymbol{w}_t^T \boldsymbol{x}_t$, where $\boldsymbol{x}_t = [\boldsymbol{x}_t; 1]$. As the performance criterion, we use the widely studied instantaneous absolute loss as our cost function, i.e., $\ell_t(\boldsymbol{w}_t) = \|e_t\|$, where the prediction error at each time instant is given by $e_t = x_{t+1} - \hat{x}_{t+1}$.

We adaptively learn the weight vector coefficients to asymptotically achieve the best possible fixed weight vector $\hat{\boldsymbol{w}}_n$, which minimizes the total prediction error after n iteration, i.e.,

$$\hat{\boldsymbol{w}}_n = \arg\min_{\boldsymbol{w}\in\mathbb{R}^M} \sum_{t=0}^{n} \|x_{t+1} - \boldsymbol{w}^T \boldsymbol{x}_t\|,$$

for any n. The definition of $\hat{\boldsymbol{w}}_n$ is given for the absolute loss case. To this end, we use the second order Online Newton Step (ONS) algorithm to train the weight vectors. The ONS algorithm significantly outperforms the first order Online Gradient Descent (OGD) algorithm in terms of convergence rate and steady state error performance since it keeps track of the second order statistics of the data sequence [15], [17], [18]. The weight vector with fixed dimension M is updated at each time as

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \frac{1}{\mu} \boldsymbol{A}_t^{-1} \nabla_t, \tag{2}$$

where $\mu \in \mathbb{R}$ is the step size and $\nabla_t \in \mathbb{R}^M$ corresponds to the gradient of the cost function $\ell_t(\boldsymbol{w}_t)$ at time t w.r.t. $\boldsymbol{w}_t$, i.e., $\nabla_t \triangleq \nabla \ell_t(\boldsymbol{w}_t)$. Here, the $M \times M$ dimensional matrix $\boldsymbol{A}_t$ is given by

$$\boldsymbol{A}_t = \sum_{i=0}^{t} \nabla_i \nabla_i^T + \alpha \boldsymbol{I}_M, \tag{3}$$

where $\alpha > 0$ is chosen to guarantee that $\boldsymbol{A}_t$ is positive definite, i.e., $\boldsymbol{A}_t > 0$, and hence, invertible. Selection of the parameters μ and α is crucial for good performance [17]. Note that for the first order OGD algorithm, we have $\boldsymbol{A}_t = \boldsymbol{I}_M$ for all t, i.e., we do not use the second order statistics but only the gradient information.

Definition of $\boldsymbol{A}_t$ in (3) has a recursive structure, i.e., $\boldsymbol{A}_t = \boldsymbol{A}_{t-1} + \nabla_t \nabla_t^T$, with an initial value of $\boldsymbol{A}_{-1} = \alpha \boldsymbol{I}_M$. Hence, we get a straight update from $\boldsymbol{A}_{t-1}^{-1}$ to $\boldsymbol{A}_t^{-1}$ using the matrix inversion lemma [21]

$$\boldsymbol{A}_t^{-1} = \boldsymbol{A}_{t-1}^{-1} - \frac{\boldsymbol{A}_{t-1}^{-1} \nabla_t \nabla_t^T \boldsymbol{A}_{t-1}^{-1}}{1 + \nabla_t^T \boldsymbol{A}_{t-1}^{-1} \nabla_t}. \tag{4}$$

Multiplying both sides of (4) with ∇_t and inserting in (2) yields

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \frac{1}{\mu} \left[\frac{\boldsymbol{A}_{t-1}^{-1} \nabla_t}{1 + \nabla_t^T \boldsymbol{A}_{t-1}^{-1} \nabla_t} \right]. \tag{5}$$

Although the second order update algorithms provide faster convergence rates and better steady state performances, computational complexity issue prohibits their usage in most real life applications [18], [21]. Since each update in (4) requires the multiplication of an $M \times M$ dimensional matrix with an M dimensional vector for $\boldsymbol{x}_t \in \mathbb{R}^M$, the computational complexity is in the order of $O(M^2)$, while the first order algorithms just need $O(M)$ multiplication and addition. As an example, in protein structure prediction, we have $M = 1000$ deeming the second order methods 1,000 times slower than the first order OGD algorithm [22].

In the next section, we introduce a sequential prediction algorithm, which achieves the performance of the Newton-Raphson methods, while offering $O(M)$ computational complexity same as the first order methods.

3 Efficient Implementation for Complexity Reduction

In this section, we construct an efficient implementation that is based on the low rank property of the update matrices. Instead of directly implementing the second order methods as in (4) and (5), we use unitary and hyperbolic transformations to update the weight vector $\boldsymbol{w}_t$ and the inverse of the Hessian-related matrix $\boldsymbol{A}_t^{-1}$.

We work on time series data sequences, which directly implies that the feature vectors $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t+1}$ are highly related. More precisely, we have the following relation between these two consecutive vectors as

$$[x_{t+1}, \boldsymbol{x}_t^T] = [\boldsymbol{x}_{t+1}^T, x_{t-M+1}]. \tag{6}$$

This relation shows that consecutive data vectors carry quite the same information, which is the basis of our algorithm. We use the instantaneous absolute loss, which is defined as

$$\ell_t(\boldsymbol{w}_t) = \|x_{t+1} - \boldsymbol{w}_t^T \boldsymbol{x}_t\|. \tag{7}$$

Although the absolute loss is widely used in the data prediction applications, it is not differentiable when $e_t = 0$. However, we resolve this issue by setting a threshold ϵ close to zero and not updating the weight vector when the absolute error is below this threshold, $\|e_t\| < \epsilon$. From (4) and (5), the absolute loss results in the following update rules for $\boldsymbol{w}_t$ and $\boldsymbol{A}_t^{-1}$,

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} \pm \frac{1}{\mu} \left[\frac{\boldsymbol{A}_{t-1}^{-1} \boldsymbol{x}_t}{1 + \boldsymbol{x}_t^T \boldsymbol{A}_{t-1}^{-1} \boldsymbol{x}_t} \right], \tag{8}$$

$$\boldsymbol{A}_t^{-1} = \boldsymbol{A}_{t-1}^{-1} - \frac{\boldsymbol{A}_{t-1}^{-1} \boldsymbol{x}_t \boldsymbol{x}_t^T \boldsymbol{A}_{t-1}^{-1}}{1 + \boldsymbol{x}_t^T \boldsymbol{A}_{t-1}^{-1} \boldsymbol{x}_t}, \tag{9}$$

since $\nabla_t = \pm \boldsymbol{x}_t$ depending on the sign of the error.

It is clear that the complexity of the second order algorithms essentially results from the matrix-vector multiplication, $\boldsymbol{A}_{t-1}^{-1} \boldsymbol{x}_t$ as in (8). Rather than getting matrix $\boldsymbol{A}_{t-1}^{-1}$ from $\boldsymbol{A}_{t-2}^{-1}$ and then calculating the multiplication $\boldsymbol{A}_{t-1}^{-1} \boldsymbol{x}_t$ individually at each iteration, we develop a direct and compact update rule, which calculates $\boldsymbol{A}_{t-1}^{-1} \boldsymbol{x}_t$ from $\boldsymbol{A}_{t-2}^{-1} \boldsymbol{x}_{t-1}$ without any explicit knowledge of the $M \times M$ dimensional matrix $\boldsymbol{A}_{t-1}^{-1}$.

Similar to [21], we first define the normalization term of the update rule given in (8) as

$$\eta_t = 1 + \boldsymbol{x}_t^T \boldsymbol{A}_{t-1}^{-1} \boldsymbol{x}_t. \tag{10}$$

Then, the difference between the consecutive terms η_t and η_{t-1} is given by

$$\eta_t - \eta_{t-1} = \boldsymbol{x}_t^T \boldsymbol{A}_{t-1}^{-1} \boldsymbol{x}_t - \boldsymbol{x}_{t-1}^T \boldsymbol{A}_{t-2}^{-1} \boldsymbol{x}_{t-1}. \tag{11}$$

We define the $(M+1) \times 1$ dimensional extended vector $\tilde{\boldsymbol{x}}_t = [x_t, \boldsymbol{x}_{t-1}^T]^T$ and get the following two equalities using the relation given in (6),

$$\eta_t = 1 + \tilde{\boldsymbol{x}}_t^T \begin{bmatrix} \boldsymbol{A}_{t-1}^{-1} & \boldsymbol{0}_M \\ \boldsymbol{0}_M^T & 0 \end{bmatrix} \tilde{\boldsymbol{x}}_t, \tag{12}$$

$$\eta_{t-1} = 1 + \tilde{\boldsymbol{x}}_t^T \begin{bmatrix} 0 & \boldsymbol{0}_M^T \\ \boldsymbol{0}_M & \boldsymbol{A}_{t-2}^{-1} \end{bmatrix} \tilde{\boldsymbol{x}}_t. \tag{13}$$

Therefore, (11) becomes

$$\eta_t - \eta_{t-1} = \tilde{\boldsymbol{x}}_t^T \Delta_{t-1} \tilde{\boldsymbol{x}}_t, \tag{14}$$

where the update term Δ_{t-1} is defined as

$$\Delta_{t-1} \triangleq \begin{bmatrix} \boldsymbol{A}_{t-1}^{-1} & \boldsymbol{0}_M \\ \boldsymbol{0}_M^T & 0 \end{bmatrix} - \begin{bmatrix} 0 & \boldsymbol{0}_M^T \\ \boldsymbol{0}_M & \boldsymbol{A}_{t-2}^{-1} \end{bmatrix}. \tag{15}$$

This equation implies that we do not need the exact values of $\boldsymbol{A}_{t-1}^{-1}$ and $\boldsymbol{A}_{t-2}^{-1}$ individually and it is sufficient to know the value of the defined difference Δ_{t-1} for the calculation of η_t. Moreover, we observe that the update term can be expressed in terms of rank 2 matrices, which is the key point for the reduction of complexity.

Initially, we assume that $x_t = 0$ for $t < 0$, which directly implies $\boldsymbol{A}_{-1}^{-1} = \boldsymbol{A}_{-2}^{-1} = \frac{1}{\alpha}\boldsymbol{I}_M$ using (3). Therefore, Δ_{-1} is found as

$$\Delta_{-1} = \frac{1}{\alpha}\ \mathrm{diag}\{1, 0, \ldots, 0, -1\}. \tag{16}$$

At this point, we define the $(M+1) \times 2$ dimensional matrix Λ_{-1} and the 2×2 dimensional matrix Π_{-1} as

$$\Lambda_{-1} = \sqrt{\frac{1}{\alpha}}\begin{bmatrix} 1 & 0 & \ldots & 0 & 0 \\ 0 & 0 & \ldots & 0 & 1 \end{bmatrix}^T, \Pi_{-1} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \tag{17}$$

to achieve the equality given by

$$\Delta_{-1} = \Lambda_{-1}\Pi_{-1}\Lambda_{-1}^T. \tag{18}$$

Here, we make an initial assumption that the low rank property of Δ_t holds for all $t \geq 0$. At the end of the discussion, we show that the assumption holds. Therefore, by using the reformulation of the difference term, we restate the η_t term given in (14) as

$$\eta_t = \eta_{t-1} + \tilde{\boldsymbol{x}}_t^T \Lambda_{t-1}\Pi_{t-1}\Lambda_{t-1}^T \tilde{\boldsymbol{x}}_t. \tag{19}$$

For the further discussion, we prefer matrix notation and represent (19) as

$$\begin{bmatrix} \sqrt{\eta_t} & \boldsymbol{0}_2^T \end{bmatrix} \begin{bmatrix} \sqrt{\eta_t} \\ \boldsymbol{0}_2 \end{bmatrix} = \begin{bmatrix} \sqrt{\eta_{t-1}} & \tilde{\boldsymbol{x}}_t^T \Lambda_{t-1} \end{bmatrix} \Theta_{t-1} \begin{bmatrix} \sqrt{\eta_{t-1}} \\ \Lambda_{t-1}^T \tilde{\boldsymbol{x}}_t \end{bmatrix}, \tag{20}$$

where Θ_{t-1} is defined as

$$\Theta_{t-1} \triangleq \begin{bmatrix} 1 & \boldsymbol{0}_2^T \\ \boldsymbol{0}_2 & \Pi_{t-1} \end{bmatrix}. \tag{21}$$

We first employ a unitary Givens transformation $\boldsymbol{H}_{G,t}$ in order to zero out the second element of the vector $[\sqrt{\eta_{t-1}}, \tilde{\boldsymbol{x}}_t^T \Lambda_{t-1}]$ and then use a Θ_{t-1}-unitary Hyperbolic rotation $\boldsymbol{H}_{HB}$, i.e., $\boldsymbol{H}_{HB,t}\Theta_{t-1}\boldsymbol{H}_{HB,t}^T = \Theta_{t-1}$, to eliminate the last term [23]. Consequently, we achieve the following update rule

$$\begin{bmatrix} \sqrt{\eta_t} & \boldsymbol{0}_2^T \end{bmatrix} = \begin{bmatrix} \sqrt{\eta_{t-1}} & \tilde{\boldsymbol{x}}_t^T \Lambda_{t-1} \end{bmatrix} \boldsymbol{H}_t, \tag{22}$$

where $\boldsymbol{H}_t$ represents the overall transformation process. Existence of these transformation matrices is guaranteed [21]. This update gives the next normalization term η_t, however, for the $(t+1)^{th}$ update, we also need the updated value of Λ_{t-1}, i.e., Λ_t, explicitly. Moreover, even calculating the Λ_t term is not sufficient, since we also need the individual value of the vector $\boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t$ to update the weight vector coefficients.

We achieve the following equalities based on the same argument that we used to get (12) and (13)

$$\begin{bmatrix} \boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t \\ 0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{t-1}^{-1} & \boldsymbol{0}_M \\ \boldsymbol{0}_M^T & 0 \end{bmatrix} \tilde{\boldsymbol{x}}_t, \tag{23}$$

$$\begin{bmatrix} 0 \\ \boldsymbol{A}_{t-2}^{-1}\boldsymbol{x}_{t-1} \end{bmatrix} = \begin{bmatrix} 0 & \boldsymbol{0}_M^T \\ \boldsymbol{0}_M & \boldsymbol{A}_{t-2}^{-1} \end{bmatrix} \tilde{\boldsymbol{x}}_t. \tag{24}$$

Here, by subtracting these two equations, we get

$$\begin{bmatrix} \boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{A}_{t-2}^{-1}\boldsymbol{x}_{t-1} \end{bmatrix} + \Delta_{t-1}\tilde{\boldsymbol{x}}_t. \tag{25}$$

We emphasize that the same transformation $\boldsymbol{H}_t$, which we used to get $\sqrt{\eta_t}$, also transforms Λ_{t-1} to Λ_t and $\boldsymbol{A}_{t-2}^{-1}\boldsymbol{x}_{t-1}$ to $\boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t$, if we extend the transformed vector as follows:

$$\begin{bmatrix} \sqrt{\eta_{t-1}} & \tilde{\boldsymbol{x}}_t^T \Lambda_{t-1} \\ \frac{1}{\sqrt{\eta_{t-1}}}\begin{bmatrix} 0 \\ \boldsymbol{A}_{t-2}^{-1}\boldsymbol{x}_{t-1} \end{bmatrix} & \Lambda_{t-1} \end{bmatrix} \boldsymbol{H}_t = \begin{bmatrix} \sqrt{\eta_t} & \boldsymbol{0}_2^T \\ \boldsymbol{q} & \boldsymbol{Q} \end{bmatrix}, \tag{26}$$

where we show that $\boldsymbol{q} = \frac{1}{\sqrt{\eta_t}}[\boldsymbol{x}_t^T \boldsymbol{A}_{t-1}^{-1}, 0]^T$ and $\boldsymbol{Q} = \Lambda_t$. We denote (26) as $\boldsymbol{B}\boldsymbol{H}_t = \tilde{\boldsymbol{B}}$, where $\boldsymbol{B}$ represents the input matrix and $\tilde{\boldsymbol{B}}$ states the output matrix of the transformation. Then, the following equality is achieved

$$\tilde{\boldsymbol{B}}\Theta_{t-1}\tilde{\boldsymbol{B}}^T = \boldsymbol{B}\Theta_{t-1}\boldsymbol{B}^T \tag{27}$$

since $\boldsymbol{H}_t$ is Θ_{t-1} unitary, i.e., $\boldsymbol{B}\boldsymbol{H}_t\Theta_{t-1}\boldsymbol{H}_t^T\boldsymbol{B}^T = \boldsymbol{B}\Theta_{t-1}\boldsymbol{B}^T$. Equating the elements of matrices in both sides of (27) yields

$$\begin{aligned} \boldsymbol{q}\sqrt{\eta_t} &= \begin{bmatrix} 0 \\ \boldsymbol{A}_{t-2}^{-1}\boldsymbol{x}_{t-1} \end{bmatrix} + \Delta_{t-1}\tilde{\boldsymbol{x}}_t, \\ \boldsymbol{q}\boldsymbol{q}^T + \boldsymbol{Q}\Pi_{t-1}\boldsymbol{Q}^T &= \frac{1}{\eta_{t-1}}\begin{bmatrix} 0 \\ \boldsymbol{A}_{t-2}^{-1}\boldsymbol{x}_{t-1} \end{bmatrix}\begin{bmatrix} 0 \\ \boldsymbol{A}_{t-2}^{-1}\boldsymbol{x}_{t-1} \end{bmatrix}^T + \Delta_{t-1}. \end{aligned} \tag{28}$$

We know from (25) that the left hand side of the first term in (28) equals to $[\boldsymbol{x}_t^T \boldsymbol{A}_{t-1}^{-1}, 0]^T$ and $\boldsymbol{q}$ is given by

$$\boldsymbol{q} = \frac{1}{\sqrt{\eta_t}}\begin{bmatrix} \boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t \\ 0 \end{bmatrix}. \tag{29}$$

Hence, we identify the value of $\boldsymbol{Q}$ matrix using the second term in (28) as

$$\begin{aligned} \boldsymbol{Q}\Pi_{t-1}\boldsymbol{Q}^T = &\begin{bmatrix} 0 & \boldsymbol{0}_M^T \\ \boldsymbol{0}_M & \frac{\boldsymbol{A}_{t-2}^{-1}\boldsymbol{x}_{t-1}\boldsymbol{x}_{t-1}^T\boldsymbol{A}_{t-2}^{-1}}{\eta_{t-1}} \end{bmatrix} \\ &+ \left(\begin{bmatrix} \boldsymbol{A}_{t-1}^{-1} & \boldsymbol{0}_M \\ \boldsymbol{0}_M^T & 0 \end{bmatrix} - \begin{bmatrix} 0 & \boldsymbol{0}_M \\ \boldsymbol{0}_M^T & \boldsymbol{A}_{t-2}^{-1} \end{bmatrix}\right) \\ &- \boldsymbol{q}\boldsymbol{q}^T, \end{aligned} \tag{30}$$

where we expand the Δ_{t-1} term using its definition given in (15). We know that the term $\frac{1}{\eta_{t-1}}\boldsymbol{A}_{t-2}^{-1}\boldsymbol{x}_{t-1}\boldsymbol{x}_{t-1}^T\boldsymbol{A}_{t-2}^{-1}$ equals to the difference $\boldsymbol{A}_{t-2}^{-1} - \boldsymbol{A}_{t-1}^{-1}$ using the update relation (9). Therefore, substituting this equality and inserting the value of $\boldsymbol{q}$ yields

$$\begin{aligned} \boldsymbol{Q}\Pi_{t-1}\boldsymbol{Q}^T &= \begin{bmatrix} \boldsymbol{A}_t^{-1} & \boldsymbol{0}_M \\ \boldsymbol{0}_M^T & 0 \end{bmatrix} - \begin{bmatrix} 0 & \boldsymbol{0}_M \\ \boldsymbol{0}_M^T & \boldsymbol{A}_{t-1}^{-1} \end{bmatrix} \\ &= \Delta_t \\ &= \Lambda_t\Pi_t\Lambda_t^T. \end{aligned} \tag{31}$$

This equality implies that Π is time invariant, i.e., $\Pi_{t-1} = \Pi_t$ and $\boldsymbol{Q}$ is given as

$$\boldsymbol{Q} = \Lambda_t. \tag{32}$$

Hence, we show that when the low rank property of the difference term Δ_t is achieved for $t = i-1$, it is preserved for the iteration $t = i$, for $i \geq 0$. Therefore, the transformation in (26) gives all the necessary information and provides a complete update rule. As a result, the weight vector is updated as

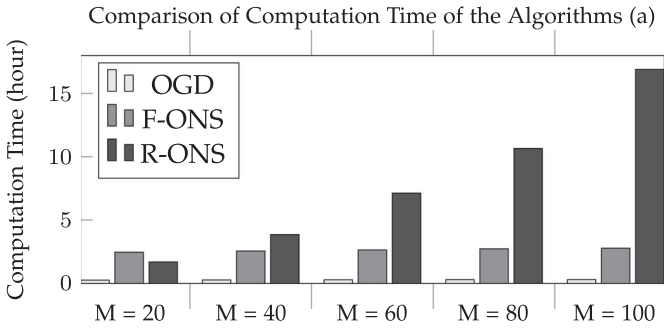

Fig. 1. Comparison of the total computation time with different feature dimension for processing 0.5 billion data points.

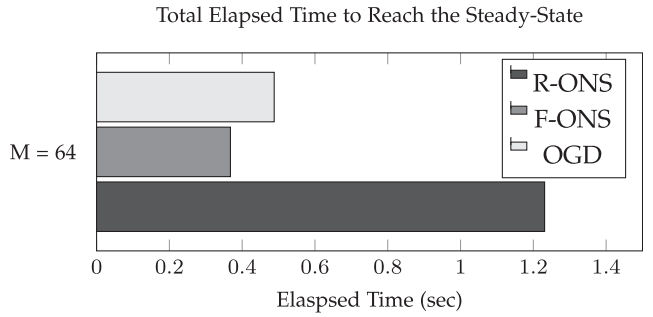

Fig. 2. Total elapsed time that the algorithms reach the steady-state.

$$\boldsymbol{w}_t = \begin{cases} \boldsymbol{w}_{t-1} + \operatorname{sgn}(e_t)\frac{1}{\mu}\left[\frac{\boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t}{\eta_t}\right], & \text{if } \|e_t\| > \epsilon \\ \boldsymbol{w}_{t-1}, & \text{otherwise}, \end{cases} \tag{33}$$

where the individual value of $\boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t$ is found by multiplying (29) by $\sqrt{\eta_t}$, which is the left upper most entry of the transformed matrix $\tilde{\boldsymbol{B}}$, and taking the first M elements. The complete algorithm is provided in Algorithm 1 with all initializations and required updates.

Algorithm 1. Fast Online Newton Step

Data: $\{\boldsymbol{x}_t\}_{t\geq 0}$ sequence
Choose $\alpha > 0$, window size M and the step size μ ;
$\Lambda_{-1} = \sqrt{\frac{1}{\alpha}}\begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix}^T$;
$\boldsymbol{\Pi} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$, $\boldsymbol{\Theta} = \begin{bmatrix} 1 & \boldsymbol{0}_2^T \\ \boldsymbol{0}_2 & \boldsymbol{\Pi} \end{bmatrix}$;
$\boldsymbol{x}_0 = \boldsymbol{0}_M$, $\boldsymbol{w}_0 = \boldsymbol{0}_M$, $\eta_{-1} = 1$, $\boldsymbol{\rho}_{-1} = \boldsymbol{0}_M$;
while $t \geq 0$ **do**
  $\tilde{\boldsymbol{x}}_t = [x_t, \boldsymbol{x}_t^T]^T$;
  $\hat{x}_{t+1} = \boldsymbol{w}_t^T \boldsymbol{x}_t$;
  $e_t = x_{t+1} - \hat{x}_{t+1}$;
  $\boldsymbol{B} = \begin{bmatrix} \sqrt{\eta_{t-1}} & \tilde{\boldsymbol{x}}_t^T \Lambda_{t-1} \\ \begin{bmatrix} 0 \\ \boldsymbol{\rho}_{t-1} \end{bmatrix} & \Lambda_{t-1} \end{bmatrix}$;
  Determine a Givens rotation $\boldsymbol{H}_{G,t}$ for $\boldsymbol{B}$;
  $\acute{\boldsymbol{B}} = \boldsymbol{B}\boldsymbol{H}_{G,t}$;
  Determine a Hyperbolic rotation $\boldsymbol{H}_{HB,t}$ for $\acute{\boldsymbol{B}}$;
  $\begin{bmatrix} \sqrt{\eta_t} & \boldsymbol{0}_2^T \\ \begin{bmatrix} \boldsymbol{\rho}_t \\ 0 \end{bmatrix} & \Lambda_t \end{bmatrix} = \acute{\boldsymbol{B}}\boldsymbol{H}_{HB,t}$;
  if $\|e_t\| > \epsilon$ **then**
  $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \operatorname{sgn}(e_t)\frac{1}{\mu}\left[\frac{\boldsymbol{\rho}_t\sqrt{\eta_t}}{\eta_t}\right]$;
  $\boldsymbol{x}_t = [x_t, x_{t-1}, \dots, x_{t-M+1}]^T$;
end

The processed matrix $\boldsymbol{B}$ has the dimensions $(M+2)\times 3$, which results in the computational complexity of $O(M)$. Since there is no statistical assumptions, we obtain the same error rates with the regular implementation.

4 Simulations

In this section, we illustrate the efficiency of our algorithm on widely used synthetic and real life sequential big datasets. We first implement the proposed fast Online Newton Step (F-ONS), the regular Online Newton Step (R-ONS) and the first order Online Gradient Descent (OGD) algorithms on a large chaotic sequence with 0.5 Billion samples and illustrate the total computation time and the corresponding mean square error (MSE) curves of each algorithm. Then we work on a large pseudo periodic time series with again 0.5 Billion time instances and represent the total elapsed time for each algorithm to reach the steady-state. We finally use two different real life big sequential datasets, one of which is a speech dataset with more than 50 million samples and the other is a time series composed of sequential temperature recordings with more than 0.6 million instances. Throughout the simulations, all data sequences are scaled to the range $[-1, 1]$.

4.1 Computational Complexity Analysis

As the first set of experiments, we examine the computation time of the proposed F-ONS, the standard R-ONS and the OGD algorithms. We first work on a chaotic sequence, e.g., Henon map [24], which is widely used in sequential regression literature. The sequence is generated from following structure

$$x_t = 1 - \alpha x_{t-1}^2 + \beta x_{t-2}. \tag{34}$$

This structure is known to exhibit a chaotic structure when the parameters are selected as $\alpha = 1.4$ and $\beta = 0.5$ [24]. In Fig. 1, we illustrate the total computation time of each algorithm when processing the whole dataset for different data dimensions. It is clear that F-ONS achieves a significant complexity reduction especially when the data dimension is high. Increasing data dimension results in only a linear increase on the computation time of the F-ONS, even though it is a second order method. The OGD, as expected, has the smallest computation time since F-ONS requires additional transformation operations.

As the second dataset, we use a Pseudo Periodic Synthetic Time Series [25], obtained from UCI KDD dataset archive. The sequence is generated by the following function

$$\bar{y} = \sum_{i=3}^{7} \sin\left(2\pi\left(2^{2+i} + \operatorname{rand}(2^i)\right)\bar{t}\right), \tag{35}$$

where the vector $\bar{t}$ consists of uniform samples on the $[0, 1]$ interval with a fixed step size of $2 \cdot 10^{-9}$, which makes a total of 0.5 Billion instances. Here, $\operatorname{rand}(2^i)$ generates a random value from the uniform distribution between 0 and 2^i. In Fig. 2, we represent the total elapsed time of each algorithm to reach their steady-state regions for $M = 64$ case. It is a significant observation that the second order F-ONS reaches the steady-state faster compared to the first order OGD algorithm. The reason is that F-ONS requires much less number of samples for convergence [15]. Even though the R-ONS processes the same number of samples with the F-ONS, it takes much more time for the R-ONS to complete processing.

We now work on two real life large sequences, where the first one is the CMU ARCTIC speech dataset [26]. The dataset includes speech recording of a male speaker. Here, we obtained two partitions of the dataset with lengths $n = 5 \cdot 10^7$ and $n = 2.5 \cdot 10^7$ (denoted by $*$) and measure the corresponding total processing time to observe the effect of increasing data length. In Fig. 3a, we

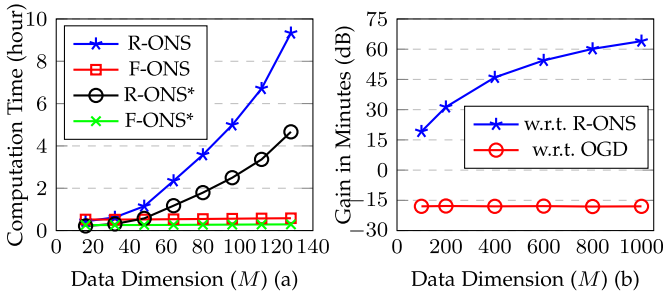

Fig. 3. (a) Comparison of the computation time. (b) Relative gain on the computation time with respect to the R-ONS and the OGD algorithms when the F-ONS algorithm is used.

demonstrate the computation time comparisons of the F-ONS and the R-ONS for several different M selections. Similar to the previous experiments, the reduction in the complexity becomes outstanding with an increasing dimensionality. We also observe that doubling the total length n results in doubled computation time for both algorithm.

We then consider the real life weather forecasting temperature dataset [27]. Here, we specifically concentrate on much larger M values and illustrate the relative computation time gain of the proposed F-ONS algorithm with respect to the R-ONS and the OGD algorithms in Fig. 3b. We observe that the relative gain w.r.t. the R-ONS shows a significant improvement as the data dimension increases. We also notice that the gain w.r.t. the OGD falls into the negative region but follows a linear structure. This is expected since it takes more time for the F-ONS to complete one iteration compared to the OGD.

4.2 Numerical Stability Analysis

We theoretically show that the introduced algorithm efficiently implements the R-ONS algorithm without any statistical assumptions or any information loss. Hence, both the R-ONS and the F-ONS offer the same error performances. However, there might occur negligible numerical differences as a consequence of the finite precision of real life computing systems. In the second part of the experiments, we examine the effects of the numerical calculations on the MSE curves. We also represent the MSE curve of the OGD for performance comparison. For each algorithm, the learning rates are selected to achieve the best performance in each case.

In Fig. 4, we illustrate the MSE curves of the algorithms for each dataset. For all datasets, except the temperature dataset, we consider $M = 64$. For the temperature dataset, we choose a higher dimension $M = 400$. It is clear that for all cases there is no observable difference between the MSE curve of the F-ONS and the R-ONS. Therefore, the proposed F-ONS is numerically stable even for high dimensional, e.g., $M = 400$, cases.

Considering the MSE curves of the OGD algorithm, we observe that the second order F-ONS and the R-ONS algorithms considerably outperforms the OGD algorithm in terms of both convergence and steady-state error rates. This reveals the significance of complexity reduction for the second order algorithms. By the proposed efficient F-ONS algorithm, we provide the merits of the second order methods with a reduced complexity that is on the same level with the first order algorithms.

5 Conclusion

In this paper, we investigate online sequential data prediction problem for high dimensional data sequences. Even though the second order Newton-Raphson methods achieve superior performance, compared to the gradient based algorithms, the problem of extremely high computational cost prohibits their usage in real life

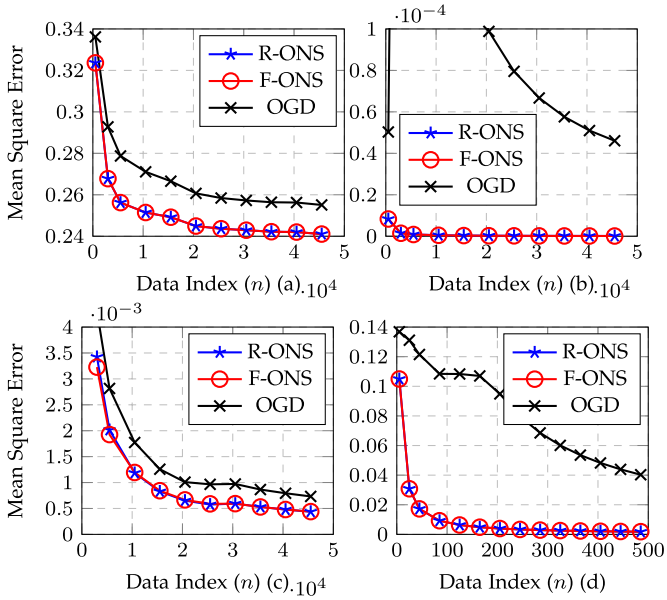

Fig. 4. (a) Chaotic Sequence: $M = 64$. (b) Pseudo Periodic Sequence: $M = 64$. (c) Speech Dataset: $M = 64$. (d) Temperature Dataset: $M = 400$.

big data applications. For an M dimensional feature vector, the computational complexity of these methods increases in the order of $O(M^2)$. To this end, we introduce a highly efficient implementation that reduces the computational complexity of the Newton-Raphson methods from $O(M^2)$ to $O(M)$. The presented algorithm does not require any statistical assumption on the data sequence. We only use the similarity between the consecutive feature vectors without any information loss. Hence, our algorithm offers the outstanding performance of the second order methods with the low computational cost of the first order methods. We illustrate that the efficient implementation of Newton-Raphson methods attains significant computational gains, as the data dimension grows. We also show that our algorithm is numerically stable.

Acknowledgments

This work is supported in part by the Turkish Academy of Sciences Outstanding Researcher Programme, TUBITAK Contract No. 113E517.

References

[1] X. Wu, X. Zhu, G. Q. Wu, and W. Ding, "Data mining with big data," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 1, pp. 97–107, Jan. 2014.
[2] C. Xu, Y. Zhang, R. Li, and X. Wu, "On the feasibility of distributed kernel regression for big data," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 11, pp. 3041–3052, Nov. 2016.
[3] R. D'Ambrosio, W. Belhajali, and M. Barlaud, "Boosting stochastic newton descent for bigdata large scale classification," in *Proc. IEEE Int. Conf. Big Data*, Oct. 2014, pp. 36–41.
[4] R. Couillet and M. Debbah, "Signal processing in large systems," *IEEE Signal Process. Mag.*, vol. 30, no. 1, pp. 211–317, 2013.
[5] R. Wolff, K. Bhaduri, and H. Kargupta, "A generic local algorithm for mining data streams in large distributed systems," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 4, pp. 465–478, Apr. 2009.
[6] T. Wu, S. H. Yu, W. Liao, and C. S. Chang, "Temporal bipartite projection and link prediction for online social networks," in *Proc. IEEE Int. Conf. Big Data*, Oct. 2014, pp. 52–59.
[7] Y. Yilmaz and X. Wang, "Sequential distributed detection in energy-constrained wireless sensor networks," *IEEE Trans. Signal Process.*, vol. 17, no. 4, pp. 335–339, Jun. 2014.
[8] T. Moon and T. Weissman, "Universal FIR MMSE filtering," *IEEE Trans. Signal Process.*, vol. 57, no. 3, pp. 1068–1083, Mar. 2009.
[9] R. Savani, "High-frequency trading: The faster, the better?" *IEEE Intell. Syst.*, vol. 27, no. 4, pp. 70–73, Jul. 2012.
[10] P. Ghosh and V. L. R. Chinthalapati, "Financial time series forecasting using agent based models in equity and fx markets," in *Proc. 6th Comput. Sci. Electron. Eng. Conf.*, Sep. 2014, pp. 97–102.

[11] L. Deng, "Long-term trend in non-stationary time series with nonlinear analysis techniques," in *Proc. 6th Int. Congr. Image Signal Process.*, Dec. 2013, pp. 1160–1163.
[12] W. Cao, L. Cao, and Y. Song, "Coupled market behavior based financial crisis detection," in *Proc. Int. Joint Conf. Neural Netw.*, Aug. 2013, pp. 1–8.
[13] L. Bottou and Y. Le Cun, "On-line learning for very large data sets," *Appl. Stochastic Models Bus. Ind.*, vol. 21, no. 2, pp. 137–151, 2005.
[14] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2008, pp. 161–168.
[15] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*. Cambridge, U.K.: Cambridge University Press, 2006.
[16] A. C. Singer, S. S. Kozat, and M. Feder, "Universal linear least squares prediction: Upper and lower bounds," *IEEE Trans. Inf. Theory*, vol. 48, no. 8, pp. 2354–2362, Aug. 2002.
[17] E. Hazan, A. Agarwal, and S. Kale, "Logarithmic regret algorithms for online convex optimization," *Mach. Learn.*, vol. 69, no. 2–3, pp. 169–192, 2007.
[18] D. Bertsimas and J. N. Tsitsiklis, *Introduction to Linear Optimization*. Belmont, MA, USA: Athena Scientific, 1997.
[19] E. K. P. Chong and S. H. Zak, *An Introduction to Optimization*. New York, NY, USA: Wiley, 2008.
[20] S. S. Kozat, A. T. Erdogan, A. C. Singer, and A. H. Sayed, "Steady-state MSE performance analysis of mixture approaches to adaptive filtering," *IEEE Trans. Signal Process.*, vol. 58, no. 8, pp. 4050–4063, Aug. 2010.
[21] A. H. Sayed, *Fundamentals of Adaptive Filtering*. Hoboken, NJ, USA: Wiley, 2003.
[22] J. Cheng, A. N. Tegge, and P. Baldi, "Machine learning methods for protein structure prediction," *IEEE Rev. Biomed. Eng.*, vol. 1, pp. 41–49, 2008.
[23] A. H. Sayed, *Adaptive Filters*. Hoboken, NJ, USA: Wiley, 2008.
[24] M. Henon, "A two-dimensional mapping with a strange attractor," *Commun. Math. Phys.*, vol. 50, pp. 69–77, 1976.
[25] D. L. S. Park and W. W. Chu, "Fast retrieval of similar subsequences in long sequence databases," in *Proc. 3rd IEEE Knowl. Data Eng. Exchange Workshop*, 1999, Art. no. 60.
[26] J. Kominek and A. W. Black, "Cmu arctic databases." (2017). [Online]. Available: http://www.festvox.org/cmu_arctic/index.html
[27] M. Liberatore, "Umass trace repository." (2017). [Online]. Available: http://traces.cs.umass.edu/index.php/Sensors/Sensors