Neural Networks 50 (2014) 142–153

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

A linear recurrent kernel online learning algorithm with sparse updates

Haijin Fan *, Qing Song

School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798, Singapore

ARTICLE INFO

Article history:
Received 11 September 2012
Received in revised form 16 August 2013
Accepted 13 November 2013

Keywords:
Kernel methods
Linear recurrent
Hybrid recurrent training
Weight convergence

ABSTRACT

In this paper, we propose a recurrent kernel algorithm with selectively sparse updates for online learning. The algorithm introduces a linear recurrent term in the estimation of the current output. This makes the past information reusable for updating of the algorithm in the form of a recurrent gradient term. To ensure that the reuse of this recurrent gradient indeed accelerates the convergence speed, a novel hybrid recurrent training is proposed to switch on or off learning the recurrent information according to the magnitude of the current training error. Furthermore, the algorithm includes a data-dependent adaptive learning rate which can provide guaranteed system weight convergence at each training iteration. The learning rate is set as zero when the training violates the derived convergence conditions, which makes the algorithm updating process sparse. Theoretical analyses of the weight convergence are presented and experimental results show the good performance of the proposed algorithm in terms of convergence speed and estimation accuracy.

1. Introduction

In the last decades, a lot of kernel-based methods have been proposed for practical applications, including time series prediction, channel equalization, pattern classification, and dimensionality reduction. The basic idea behind kernel methods is that the Mercer kernels are applied to map the original input feature into a higher-dimensional or even infinite-dimensional feature space. In such a feature transformation, linear solutions could be found in the higher-dimensional feature space even if the original feature is highly nonlinear (Schōlkopf, Herbrich, & Smola, 2001). The well-developed kernel methods include support vector machines (SVMs) (Burges, 1998; Sebald & Bucklew, 2000; Š Raudys, 2000), kernel embedding methods (Guo, Gao, & Kwan, 2008; Memisevic, Sigal, & Fleet, 2012), and different kinds of kernel least mean square (KLMS) algorithms (Campbell, 2002; Rasmussen, 2004). However, the main problem of kernel methods is the growing network size with the increasing number of training samples. The complexity of their structures makes most of the current kernel methods only suitable for off-line learning or batch learning.

In classical kernel methods, the kernel function number becomes very large as the amount of training data continuously increases. The large number of kernel functions makes the algorithms not only be in danger of overfitting but also to have a high computational complexity growing superlinearly with the number of kernel functions. To prevent the network size being too large, a compact dictionary should be chosen as the kernel function centers. Recently, different sparsification methods have been proposed to address this problem. They aimed to select a compact dictionary with finite size using different criteria (Engel, Mannor, & Meir, 2004; Kivinen, Smola, & Williamson, 2004; Liu, Park, & Príncipe, 2009; Richard, Bermudez, & Honeine, 2009). Based on these sparsification methods, many kernel online learning algorithms were proposed. These algorithms include the kernel least mean square (KLMS) algorithm (Chen, Zhao, Zhu, & Príncipe, 2012; Liu, Pokharel, & Príncipe, 2008), the kernel normalized least mean square (KNLMS) algorithm (Richard et al., 2009), the kernel affine projection (KAP) algorithm (Liu & Príncipe, 2008; Slavakis & Theodoridis, 2008), and the kernel recursive least square (KRLS) algorithm (Engel et al., 2004). They generalized kinds of linear least square algorithms and found their solutions in the reproducing kernel Hilbert spaces (RKHS). In online learning settings, at each training iteration, only one training sample is applied for the updating of the algorithm. In this fashion, the above-mentioned online kernel algorithms used only the current training sample to update themselves and their outputs only depended on the current feature inputs, without any relations to the previous outputs. In this sense, they are a kind of *feedforward networks*. On the other hand, the *recurrent networks* have been widely examined for their excellent efficiency and ability in nonlinear modeling as in recurrent neural networks (RNNs) (Baltersee & Chambers, 1998; Song, Wu, & Soh, 2008; Sperduti, 1997), recurrent SVMs

* Corresponding author. Tel.: +65 85525016.
E-mail addresses: hfan1@e.ntu.edu.sg, fanhiking@gmail.com (H. Fan).

0893-6080/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.neunet.2013.11.011

(Qu, Oussar, Dreyfus, & Xu, 2009; Suykens & Vandewalle, 2000; Xie, 2009), and recurrent radial basis function (RBF) networks (Billings, Wei, & Balikhin, 2007; Mimura, Hamada, & Furukawa, 2002). In recurrent networks, the current output depends not only on the current feature inputs but also on the previous outputs. In the training of recurrent networks, the recurrent part provides the past gradient information in the form of a Jacobian matrix in many recurrent neural networks (Pearlmutter, 1995; Song, 2011; Song et al., 2008). The reuse of the past information is shown to be able to accelerate the convergence speed in many nonlinear adaptive networks (Diniz & Werner, 2003; Qiu et al., 2006; Soni, Gallivan, & Jenkins, 2004). However, the Jacobian matrix is in a nonlinear form of the previous gradient information, and the evaluation of it is very complex.

This paper presents a novel linear recurrent kernel online learning algorithm. A linear feedback of the one-step previous output is incorporated into the algorithm in a special way. This is very significant for the online training since it provides the past gradient information in a linear form. In this way, the updating gradient information can be evaluated recursively with a very low computational complexity. To guarantee the weight convergence, in our training method, three adaptive parameters are automatically determined to adjust the training process. A hybrid learning rate is chosen to determine the proper learning of the recurrent gradient information to accelerate the convergence rate. An adaptive learning rate and a normalization factor are used to guarantee the weight convergence at each training iteration according to the derived convergence conditions. The adaptive learning rate features the sparseness of the updating, where only the training iterations without violating the convergence criteria are active. Similar sparse updates and adaptive learning have already applied in many adaptive systems (Bhotto & Antoniou, 2012; Nagaraj, Gollamudi, Kapoor, & Huang, 1999; Ozay, Sznaier, Lagoa, & Camps, 2012). The main contributions of this study are that, on the one hand, it introduces a linear recurrent version of kernel online learning algorithm where the reuse of past information is able to accelerate the convergence speed; on the other hand, it generalizes the adaptive training methods in the RKHS in a novel way which can ensure the weight convergence at each training step in online learning. To curb the increasing growth of the kernel function number, we use a coherence-based criterion for sparsification and derive a linear recurrent kernel online learning (LRKOL) algorithm for nonlinear signal processing and system modeling.

The organization of this paper is as follows. In Section 2, we introduce some fundamental ideas of kernel methods. In Section 3, the proposed linear recurrent kernel online learning algorithm is presented. The detailed training method is described followed by the theoretical analysis of the weight convergence. In Section 5, experimental and simulation results of several examples are presented, and finally a conclusion is given in Section 6.

2. Fundamentals of kernel methods

For many nonlinear signal processing problems, it is difficult to find linear models in their original low-dimensional feature spaces. However, by using a nonlinear mapping function, which maps the low-dimensional feature spaces into the higher-dimensional RKHS, linear models can be found. Suppose that $\mathcal{H}$ is a Hilbert space and that $\langle\cdot,\cdot\rangle_{\mathcal{H}}$ is the inner product in the Hilbert space. A mapping function $\varphi(\cdot)$ will transfer the input feature space $\mathcal{U}$ to a higher-dimensional feature space $\mathcal{F}$. The evaluation of the function $\varphi(\boldsymbol{u}(i))$ on the point $\boldsymbol{u}(j)$ becomes (Aronszajn, 1951)

$$(\varphi(\boldsymbol{u}(i)))(\boldsymbol{u}(j)) = \langle\varphi(\boldsymbol{u}(i)), \varphi(\boldsymbol{u}(j))\rangle_{\mathcal{H}} = k(\boldsymbol{u}(i), \boldsymbol{u}(j)), \tag{1}$$

where $k(\cdot,\cdot)$ is a positive definite and symmetric kernel function. The inner product of two feature vectors in the higher-dimensional feature space can be easily computed by (1) without knowing the exact function of $\varphi(\cdot)$, which is usually called the *kernel trick*. The commonly used kernels include the Gaussian kernel $\kappa(\boldsymbol{u}(i), \boldsymbol{u}(j)) = \exp(-\|\boldsymbol{u}(i) - \boldsymbol{u}(j)\|^2/2\sigma^2)$, the Laplacian kernel $\kappa(\boldsymbol{u}(i), \boldsymbol{u}(j)) = \exp(-\|\boldsymbol{u}(i) - \boldsymbol{u}(j)\|/\sigma)$ with σ being the kernel width, and the polynomial kernel $\kappa(\boldsymbol{u}(i), \boldsymbol{u}(j)) = (\eta + \boldsymbol{u}(i)^\top \boldsymbol{u}(j))^q$, with $\eta \geq 0$ and $q \in \mathbb{N}$.

2.1. Kernel methods

The Mercer kernel is widely used to compute the inner product of high-dimensional features in the RKHS in kernel methods. Let $\kappa(\cdot,\cdot) : \mathcal{U} \times \mathcal{U} \rightarrow R$ denote the kernel mapping in the corresponding RKHS $\mathcal{H}$. Given the feature input-desired output sequence $\{\boldsymbol{u}(j), d(j)\}_{j=1}^t$, the problem is to find a function $f(\cdot)$ to reconstruct the corresponding output $f(\boldsymbol{u}(t)) = \langle f(\cdot), k(\cdot, \boldsymbol{u}(t))\rangle_{\mathcal{H}}$. By virtue of the representer theorem (Aronszajn, 1951), the function $f(\cdot)$ can be expressed as linear form in the RKHS

$$f(\cdot) = \boldsymbol{\omega}^T(t)\varphi(\cdot), \tag{2}$$

where $\boldsymbol{\omega}(t)$ is the weight coefficient, and it can be expressed as a linear combination of the feature vectors in the RKHS obtained till the tth training iteration,

$$\boldsymbol{\omega}(t) = \sum_{j=1}^{t} \alpha_j \varphi(\boldsymbol{u}(j)). \tag{3}$$

Using the kernel trick, we have

$$f(\cdot) = \sum_{j=1}^{t} \alpha_j \kappa(\cdot, \boldsymbol{u}(j)), \tag{4}$$

where $\kappa(\cdot, \boldsymbol{u}(j))$ is a kernel function with its center being the input feature vector $\boldsymbol{u}(j)$, and α_j is the kernel weight. As a result, the function can be estimated implicitly by the feature input vectors with a Mercer kernel function. If sparsification methods (Engel et al., 2004; Kivinen et al., 2004; Liu et al., 2009; Richard et al., 2009) are applied to reduce the kernel function number, and supposing that a sparse dictionary $\mathcal{D}(t) = \{\boldsymbol{u}(\mathcal{D}_1), \ldots, \boldsymbol{u}(\mathcal{D}_m)\}$ with m members is obtained, the estimated function becomes

$$f(\cdot) = \sum_{j=1}^{m} \alpha_j \kappa(\cdot, \boldsymbol{u}(\mathcal{D}_j)), \tag{5}$$

where the number of kernel functions is limited to the size of the dictionary.

3. Linear recurrent kernel online learning algorithm

Different from the *feedforward* kernel algorithm, the output of the proposed recurrent kernel algorithm depends not only on the current feature input but also on the one-step previous output. The explicit network structure is shown in Fig. 1. Consider the current input $\boldsymbol{u}(t)$; the estimated output $y(t)$ is determined by the feature input together with a linear feedback of the previous output $y(t-1)$, which can be formulated as

$$\begin{aligned} y(t) &= \sum_{i=1}^{m} (\kappa(\boldsymbol{u}(t), \boldsymbol{u}(\mathcal{D}_i)) + \lambda_i y(t-1))\alpha_i \\ &= [\boldsymbol{K}(t) + y(t-1)\boldsymbol{\lambda}(t)]^T \boldsymbol{\alpha}(t), \end{aligned} \tag{6}$$

where $\boldsymbol{K}(t) \in \mathcal{R}^{m\times 1}$ is the kernel evaluation vector of the input with the existing dictionary $\mathcal{D}(t) = \{\boldsymbol{u}(\mathcal{D}_1), \ldots, \boldsymbol{u}(\mathcal{D}_m)\}$, and it is defined as

$$\boldsymbol{K}(t) = [\kappa(\boldsymbol{u}(t), \boldsymbol{u}(\mathcal{D}_1)), \ldots, \kappa(\boldsymbol{u}(t), \boldsymbol{u}(\mathcal{D}_m))]^T. \tag{7}$$