FP-ELM: An Online Sequential Learning Algorithm for Dealing with Concept Drift

Dong Liu[a,b,c], YouXi Wu[b,c,*], He Jiang[a]

[a]School of Software, Dalian University of Technology, Dalian 116621, China
[b]School of Computer Science and Software, Hebei University of Technology, Tianjin 300130, China
[c]Hebei province key laboratory of big data calculation, Tianjin 300401, China

Abstract

The online sequential extreme learning machine (OS-ELM) algorithm is an on-line and incremental learning method, which can learn data one-by-one or chunk-by-chunk with a fixed or varying chunk size. And OS-ELM achieves the same learning performance as ELM trained by the complete set of data. However, in on-line learning environments, the concepts to be learned may change with time, a feature referred to as concept drift. To use ELMs in such non-stationary environments, a forgetting parameters extreme learning machine (FP-ELM) is proposed in this paper. The proposed FP-ELM can achieve incremental and on-line learning, just like OS-ELM. Furthermore, FP-ELM will assign a forgetting parameter to the previous training data according to the current performance to adapt to possible changes after a new chunk comes. The regularized optimization method is used to avoid estimator windup. Performance comparisons between FP-ELM and two frequently used ensemble approaches are carried out on several regression and classification problems with concept drift. The experimental results show that FP-ELM produces comparable or better performance with lower training time.

Keywords: extreme learning machine, online/incremental learning, concept drift, regularized optimization method

1. Introduction

Single layer feed-forward networks (SLFNs) have been intensively studied in both theoretical and applied fields [1, 2] due to their concise structures and approximation capability [3, 4, 5]. However, most of the learning algorithms of SLFNs, such as gradient based and computational intelligence based algorithms, tend to be time consuming, as many parameters need to be determined during training. Different from the conventional methods, a new learning scheme, which is referred to as extreme learning machine (ELM), has been proposed by Huang *et al.* [6] for training SLFNs. In ELM, all the hidden layer related parameters are randomly assigned, and the output weights between the hidden layer and the output layer can be expressed in a compact form or calculated in an efficient way (e.g., by using the pseudo-inverse matrix). Compared to gradient based learning algorithms, ELM usually achieves better generalization performance in a much shorter training time [7, 8]. Apart from applying ELM directly, many variants were developed to solve different problems, such as functional data learning [9], imbalance classification [10], semi-supervised and unsupervised learning [11], etc.

In recent years, online and incremental learning has been used in a growing number of applications [12, 13, 14, 15]. To incorporate online learning with ELM, Liang *et al.* [16] proposed the online sequential extreme learning machine (OS-ELM), which can learn data one-by-one or chunk-by-chunk with a fixed

*Corresponding author
Email addresses: dongliu05@gmail.com (Dong Liu), wuc@scse.hebut.edu.cn (YouXi Wu), jianghe@dlut.edu.cn (He Jiang)

or varying chunk size. Then, Lan *et al.* [17] proposed EOS-ELM, which predicts by integrating several individual OS-ELMs. EOS-ELM is more stable than the original OS-ELM.

However, in many practical applications, such as customer behavior analysis, stock index prediction, network security protection, bug triage [18], etc., the online environment is non-stationary. That is, the pattern to be modelled by the learner may change with time. This phenomenon is called concept drift [19]. Non-stationary environment learning algorithms should be able to detect changes quickly and update the learner accordingly.

There are various kinds of methods to deal with concept drift problems, and they are mainly divided into two classes. The first one is the single learner method. This class of algorithms constructs only one learner during learning and usually works by actively selecting training samples [20], dynamically tuning the structure of the learner [21], or by assigning weights to the samples to discount old data [22, 23]. The second one is referred to as the ensemble method. These algorithms usually focus on how to integrate different learners to adapt to the possible changes. In the main ensemble strategies, a learner constructed on the latest data is added periodically to the ensemble, and some outdated or inaccurate learners may be pruned [24, 25, 26]. The ensemble weights are updated constantly according to the performance of each base learner and the ensemble based on recent data [27, 28, 29, 30]. Each base learner in the ensemble may keep its structure unchange [24], execute ordinary online learning [28] or even evolve with the change of the environment [31]. Some researchers use ensembles with different diversity levels [32, 33]. Compared with single learner methods, ensemble methods achieve higher accuracy. But as the information of multiple learners needs to be stored and processed, they usually have a high time and space cost, which puts up barriers to their use in fast changing environments.

In this paper, we propose a new learning algorithm for concept drift problems called forgetting parameters extreme learning machine (FP-ELM). The same as OS-ELM, FP-ELM can learn data arriving one-by-one or chunk-by-chunk without the need for storing all the training samples accumulated so far. To adapt to non-stationary environments, FP-ELM works by incorporating online learning with a forgetting mechanism in which the weights of previous training samples are reduced gradually. Just like other variants of ELM, FP-ELM has a high learning speed and a good generalization performance.

The rest of this paper is organized as follows. Section 2 gives a brief review of concept drift, ELM and OS-ELM. Section 3 represents the main ideas and properties of the proposed algorithm, FP-ELM. Performance comparisons between FP-ELM and two ensemble approaches for concept drift are shown in Section 4. And the conclusions are highlighted in Section 5.

2. Preliminaries

2.1. Concept drift

Generally speaking, there is not a universal definition of concept drift, and it roughly refers to a type of time-changing behavior in data streams. This non-stationary behavior may appear in both classification and regression problems. Most of the relevant studies are about concept drift in classification problems.

Specially, we give a formal description of concept drift in classification case. Assume $\mathbf{x}$ represents the attributes (inputs) of data, and t represents a possible class label. Concept drift means that the joint distribution $p(\mathbf{x}, t)$ changes over time [19, 34]. Deduced from formula

$$p(\mathbf{x}, t) = p(t|\mathbf{x})p(\mathbf{x}), \tag{1}$$

concept drift can be attributed to a change in the feature probability $p(\mathbf{x})$ and a change in the posterior probability $p(t|\mathbf{x})$. A change in $p(t|\mathbf{x})$ is also called real drift as the true class boundaries are altered. The learner should be rebuilt to adapt to the new concept in this case. We mainly focus on this case in the following discussion.

In fact, the arguments above can be extended to the regression case directly. Then, t represents the target (output) value, which is a continuous variable rather than one chosen from a discrete set. For instance, if the outputs are all normalized into [-1,1], then t can be any value in [-1,1].

2.2. ELM and OS-ELM

ELM is a training algorithm for generalized SLFNs. A hidden node of such SLFNs can be represented as not only a neuron in a neural network, but also a non-neuron-like term such as a fuzzy rule, a wavelet or even an independent learner. Different from conventional learning algorithms, the parameters of hidden nodes need not be tuned during training in ELM. Suppose that a SLFN has d input nodes, L hidden nodes and m output nodes. Its output function (a vector) can be represented by

$$\mathbf{f}_L(\mathbf{x}) = \sum_{i=1}^{L} \boldsymbol{\beta}_i G(\mathbf{a}_i, b_i, \mathbf{x}), \tag{2}$$

where $\mathbf{x} \in \mathbf{R}^d$ are the inputs, $\mathbf{a}_i \in \mathbf{R}^d$ and $b_i \in R$ are hidden parameters, which are assigned randomly, $G(\mathbf{a}_i, b_i, \mathbf{x})$ is the output of the ith hidden node and $\boldsymbol{\beta}_i \in \mathbf{R}^m$ are the output weights of the connections between the ith hidden node and the output layer. Various types of hidden nodes can used by SLFNs; the most common ones include the additive hidden nodes:

$$G(\mathbf{a}, b, \mathbf{x}) = g(\mathbf{x} \cdot \mathbf{a} + b) \tag{3}$$

and the RBF hidden nodes:

$$G(\mathbf{a}, b, \mathbf{x}) = g(b\|\mathbf{x} - \mathbf{a}\|)(b > 0), \tag{4}$$

where the activation $g : R \to R$ can be many continuous or piecewise continuous functions.

In ELM, the only parameters, which need to be determined, are the output weights $\boldsymbol{\beta} = (\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, ..., \boldsymbol{\beta}_L)^{\mathbf{T}}$. Concretely speaking, for a given training set $\{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=1}^{N} \in \mathbf{R}^d \times \mathbf{R}^m$, ELM is to minimize the training error $\|\mathbf{H}\boldsymbol{\beta} - \mathbf{T}\|$ and the norm of the output weight matrix $\|\boldsymbol{\beta}\|$[1], where

$$\mathbf{H} = \begin{pmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_1) & \dots & G(\mathbf{a}_L, b_L, \mathbf{x}_1) \\ \vdots & \dots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_N) & \dots & G(\mathbf{a}_L, b_L, \mathbf{x}_N) \end{pmatrix}_{N \times L} \tag{5}$$

and

$$\mathbf{T} = \begin{pmatrix} \mathbf{t}_1^{\mathbf{T}} \\ \vdots \\ \mathbf{t}_N^{\mathbf{T}} \end{pmatrix}_{N \times m}. \tag{6}$$

$\mathbf{H}$ is called the hidden layer output matrix.

To calculate $\boldsymbol{\beta}$, the minimal norm least square solution is used in the original ELM, that is

$$\boldsymbol{\beta} = \mathbf{H}^{\dagger}\mathbf{T}, \tag{7}$$

where $\mathbf{H}^{\dagger}$ is the Moore-Penrose generalized inverse of matrix $\mathbf{H}$. $\mathbf{H}^{\dagger}$ can be efficiently calculated using $\mathbf{H}^{\dagger} = (\mathbf{H}^{\mathbf{T}}\mathbf{H})^{-1}\mathbf{H}^{\mathbf{T}}$ when $\mathbf{H}^{\mathbf{T}}\mathbf{H}$ is nonsingular or $\mathbf{H}^{\dagger} = \mathbf{H}^{\mathbf{T}}(\mathbf{H}\mathbf{H}^{\mathbf{T}})^{-1}$ when $\mathbf{H}\mathbf{H}^{\mathbf{T}}$ is nonsingular.

In analogy with support vector machine (SVM), another way of calculating the output weights is proposed based on the regularized optimization method [35]:

$$\text{Minimize:} \quad L(\boldsymbol{\beta}) = \frac{\lambda}{2}\|\boldsymbol{\beta}\|^2 + \frac{1}{2}\|\mathbf{H}\boldsymbol{\beta} - \mathbf{T}\|^2 \tag{8}$$

where λ is the regularization parameter. Then we have

$$\boldsymbol{\beta} = (\lambda\mathbf{I} + \mathbf{H}^{\mathrm{T}}\mathbf{H})^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{T} \quad \text{or} \quad \mathbf{H}^{\mathrm{T}}(\lambda\mathbf{I} + \mathbf{H}\mathbf{H}^{\mathrm{T}})^{-1}\mathbf{T}. \tag{9}$$

[1]The norm $\|\cdot\|$ appearing in this paper denotes the Frobenius norm of "$\cdot$". And if the network output dimension is $m = 1$, the norm will be equal to the vector 2-norm.

Compared with the original form, the new form adds a positive value λ to the diagonal of $\mathbf{H}^{\mathrm{T}}\mathbf{H}$ or $\mathbf{H}\mathbf{H}^{\mathrm{T}}$, and it is more stable.

OS-ELM is a variant of ELM, which can learn data one by one or chunk by chunk. To run OS-ELM, a small chunk of initial training data is needed, and we assume it is $\aleph_0 = \{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=1}^{N_0}$. Then, initialize $\mathbf{P}_0 = (\mathbf{H}_0^{\mathbf{T}}\mathbf{H}_0)^{-1}$ and the output weight matrix $\boldsymbol{\beta}^{(0)} = \mathbf{H}_0^{\dagger}\mathbf{T}_0 = \mathbf{P}_0\mathbf{H}_0^{\mathbf{T}}\mathbf{T}_0$ where

$$\mathbf{H}_0 = \begin{pmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_1) & \dots & G(\mathbf{a}_L, b_L, \mathbf{x}_1) \\ \vdots & \dots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_{N_0}) & \dots & G(\mathbf{a}_L, b_L, \mathbf{x}_{N_0}) \end{pmatrix}_{N_0 \times L} \tag{10}$$

and

$$\mathbf{T}_0 = \begin{pmatrix} \mathbf{t}_1^{\mathbf{T}} \\ \vdots \\ \mathbf{t}_{N_0}^{\mathbf{T}} \end{pmatrix}_{N_0 \times m}. \tag{11}$$

In order to make $\mathbf{H}_0^{\mathrm{T}}\mathbf{H}_0$ nonsingular (rank($\mathbf{H}_0$)=L), $N_0 \geq L$ is required.

After this, when the kth ($k \geq 1$) data chunk $\aleph_k = \{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=(\sum_{j=0}^{k-1} N_j)+1}^{\sum_{j=0}^{k} N_j}$ arrives, the output weight matrix $\boldsymbol{\beta}$ is updated by formulas

$$\mathbf{P}_k = \mathbf{P}_{k-1} - \mathbf{P}_{k-1}\mathbf{H}_k^{\mathrm{T}}(\mathbf{I} + \mathbf{H}_k\mathbf{P}_{k-1}\mathbf{H}_k^{\mathrm{T}})^{-1}\mathbf{H}_k\mathbf{P}_{k-1} \tag{12}$$

and

$$\boldsymbol{\beta}^{(k)} = \boldsymbol{\beta}^{(k-1)} + \mathbf{P}_k\mathbf{H}_k^{\mathrm{T}}(\mathbf{T}_k - \mathbf{H}_k\boldsymbol{\beta}^{(k-1)}), \tag{13}$$

where

$$\mathbf{H}_k = \begin{pmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_{(\sum_{j=0}^{k-1} N_j)+1}) & \dots & G(\mathbf{a}_L, b_L, \mathbf{x}_{(\sum_{j=0}^{k-1} N_j)+1}) \\ \vdots & \dots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_{\sum_{j=0}^{k} N_j}) & \dots & G(\mathbf{a}_L, b_L, \mathbf{x}_{\sum_{j=0}^{k} N_j}) \end{pmatrix}_{N_k \times L} \tag{14}$$

and

$$\mathbf{T}_k = \begin{pmatrix} \mathbf{t}_{(\sum_{j=0}^{k-1} N_j)+1}^{\mathbf{T}} \\ \vdots \\ \mathbf{t}_{\sum_{j=0}^{k} N_j}^{\mathbf{T}} \end{pmatrix}_{N_k \times m}. \tag{15}$$

3. Proposed FP-ELM

As an online sequential learning method, OS-ELM can handle data arriving one-by-one or chunk-by-chunk. In OS-ELM, every chunk plays an equal role during the training phase. Concretely, for any newly arrived chunk, we assume its order number is n. The current output weights are determined to minimize

$$L_n(\boldsymbol{\beta}) = \frac{1}{2}\sum_{k=0}^{n} \|\mathbf{H}_k\boldsymbol{\beta} - \mathbf{T}_k\|^2. \tag{16}$$

However, in many real applications, especially in non-stationary or concept drifting environments, the data in stream may have obvious timeliness. That is, the newly arrived data tend to reflect the current patterns in the data, and this is pivotal in learner training. The earlier data may play a minor role, and some data that are too old or are even totally out-dated only bring errors to the learner. Under these conditions, data chunks arriving at different periods have a different importance in training, but OS-ELM fails to reflect this. To deal with this problem, we propose the forgetting parameters extreme learning machine (FP-ELM).

Let us begin with a special case. For an initial data chunk $\aleph_0 = \{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=1}^{N_0}$, according to the regularized ELM theory, the network output weight matrix is

$$\boldsymbol{\beta}^{(0)} = (\lambda \mathbf{I} + \mathbf{H}_0^{\mathrm{T}} \mathbf{H}_0)^{-1} \mathbf{H}_0^{\mathrm{T}} \mathbf{T}_0, \tag{17}$$

where λ is the regularization parameter. When a new data chunk $\aleph_1 = \{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=N_0+1}^{N_0+N_1}$ arrives, we can easily obtain its hidden-layer output matrix $\mathbf{H}_1$ and target output $\mathbf{T}_1$. Now, the output weights should be updated by using the new chunk. To pay more attention to the new data chunk, we give a forgetting parameter α_1 $(0 < \alpha_1 < 1)$ to the old data chunk $\aleph_0$. Formally, the new output weight matrix $\boldsymbol{\beta}^{(1)}$ is the solution to minimize

$$L_1^*(\boldsymbol{\beta}) = \frac{\lambda}{2}\|\boldsymbol{\beta}\|^2 + \frac{1}{2}(\|\alpha_1(\mathbf{H}_0\boldsymbol{\beta} - \mathbf{T}_0)\|^2 + \|\mathbf{H}_1\boldsymbol{\beta} - \mathbf{T}_1\|^2), \tag{18}$$

where λ is the regularization parameter. In fact, the explicit expression is

$$\boldsymbol{\beta}^{(1)} = (\lambda \mathbf{I} + \alpha_1^2 \mathbf{H}_0^{\mathrm{T}} \mathbf{H}_0 + \mathbf{H}_1^{\mathrm{T}} \mathbf{H}_1)^{-1} (\alpha_1^2 \mathbf{H}_0^{\mathrm{T}} \mathbf{T}_0 + \mathbf{H}_1^{\mathrm{T}} \mathbf{T}_1). \tag{19}$$

Notice $(\lambda \mathbf{I} + \mathbf{K}_0)\boldsymbol{\beta}^{(0)} = \mathbf{H}_0^{\mathrm{T}} \mathbf{T}_0$ where $\mathbf{K}_0 = \mathbf{H}_0^{\mathrm{T}} \mathbf{H}_0$; then, we have

$$\begin{aligned}
\boldsymbol{\beta}^{(1)} =& (\lambda \mathbf{I} + \alpha_1^2 \mathbf{K}_0 + \mathbf{H}_1^{\mathrm{T}} \mathbf{H}_1)^{-1} (\alpha_1^2 (\lambda \mathbf{I} + \mathbf{K}_0)\boldsymbol{\beta}^{(0)} + \mathbf{H}_1^{\mathrm{T}} \mathbf{T}_1) \\
=& (\lambda \mathbf{I} + \alpha_1^2 \mathbf{K}_0 + \mathbf{H}_1^{\mathrm{T}} \mathbf{H}_1)^{-1} ((\lambda \mathbf{I} + \alpha_1^2 \mathbf{K}_0 + \mathbf{H}_1^{\mathrm{T}} \mathbf{H}_1)\boldsymbol{\beta}^{(0)} \\
& - \lambda(1 - \alpha_1^2)\boldsymbol{\beta}^{(0)} + \mathbf{H}_1^{\mathrm{T}} \mathbf{T}_1 - \mathbf{H}_1^{\mathrm{T}} \mathbf{H}_1 \boldsymbol{\beta}^{(0)}) \\
=& \boldsymbol{\beta}^{(0)} + (\lambda \mathbf{I} + \alpha_1^2 \mathbf{K}_0 + \mathbf{H}_1^{\mathrm{T}} \mathbf{H}_1)^{-1} (\mathbf{H}_1^{\mathrm{T}} (\mathbf{T}_1 - \mathbf{H}_1 \boldsymbol{\beta}^{(0)}) \\
& - \lambda(1 - \alpha_1^2)\boldsymbol{\beta}^{(0)}) \\
=& \boldsymbol{\beta}^{(0)} + (\lambda \mathbf{I} + \mathbf{K}_1)^{-1} (\mathbf{H}_1^{\mathrm{T}} (\mathbf{T}_1 - \mathbf{H}_1 \boldsymbol{\beta}^{(0)}) - \lambda(1 - \alpha_1^2)\boldsymbol{\beta}^{(0)}),
\end{aligned} \tag{20}$$

where $\mathbf{K}_1$ is given by

$$\mathbf{K}_1 = \alpha_1^2 \mathbf{K}_0 + \mathbf{H}_1^{\mathrm{T}} \mathbf{H}_1. \tag{21}$$

Generalizing this process, when the kth $(k \geq 1)$ data chunk $\aleph_k$ arrives, the weights (importance) in the training of all previous chunks will shrink by the forgetting parameter α_k $(0 < \alpha_k < 1)$. Then, the recursive way of updating the output weights can be written as

$$\mathbf{K}_k = \alpha_k^2 \mathbf{K}_{k-1} + \mathbf{H}_k^{\mathrm{T}} \mathbf{H}_k, \tag{22}$$

$$\boldsymbol{\beta}^{(k)} = \boldsymbol{\beta}^{(k-1)} + (\lambda \mathbf{I} + \mathbf{K}_k)^{-1} (\mathbf{H}_k^{\mathrm{T}} (\mathbf{T}_k - \mathbf{H}_k \boldsymbol{\beta}^{(k-1)}) - \lambda(1 - \alpha_k^2)\boldsymbol{\beta}^{(k-1)}), \tag{23}$$

where $\mathbf{H}_k$ and $\mathbf{T}_k$ are the hidden-layer output matrix and the target output of $\aleph_k$, respectively.

Once the forgetting parameters $\{\alpha_k\}_{k \in N_+}$ are determined, FP-ELM can learn by using recursive formulas (22) and (23).

We have the following observations.

Proposition 1. *For any integer n $(n \geq 1)$, given the forgetting parameters $\{\alpha_k\}_{k=1,2,\ldots,n}$, the output weights are recursively calculated using formulas (22)(23) and initial values $\mathbf{K}_0 = \mathbf{H}_0^{\mathrm{T}} \mathbf{H}_0$, $\boldsymbol{\beta}^{(0)} = (\lambda \mathbf{I} + \mathbf{K}_0)^{-1} \mathbf{H}_0^{\mathrm{T}} \mathbf{T}_0$; then $\boldsymbol{\beta}^{(n)}$, the final output weight matrix after n steps, is the solution to minimize*

$$L_n^*(\boldsymbol{\beta}) = \frac{\lambda}{2}\|\boldsymbol{\beta}\|^2 + \frac{1}{2}(\sum_{k=0}^{n-1} \| \prod_{j=k+1}^{n} \alpha_j (\mathbf{H}_k \boldsymbol{\beta} - \mathbf{T}_k)\|^2 + \|\mathbf{H}_n \boldsymbol{\beta} - \mathbf{T}_n\|^2). \tag{24}$$

Proof. From $\mathbf{K}_0 = \mathbf{H}_0^{\mathrm{T}} \mathbf{H}_0$ and formula (22), we have

$$\mathbf{K}_n = \sum_{k=0}^{n-1} (\prod_{j=k+1}^{n} \alpha_j^2 \mathbf{H}_k^{\mathrm{T}} \mathbf{H}_k) + \mathbf{H}_n^{\mathrm{T}} \mathbf{H}_n \quad (n \geq 1). \tag{25}$$

Multiply both sides of equation (23) by $\lambda\mathbf{I}+\mathbf{K}_k$, we have

$$\begin{aligned}(\lambda\mathbf{I}+\mathbf{K}_k)\boldsymbol{\beta}^{(k)} =&(\lambda\mathbf{I}+\mathbf{K}_k)\boldsymbol{\beta}^{(k-1)}+\mathbf{H}_k^{\mathrm{T}}(\mathbf{T}_k-\mathbf{H}_k\boldsymbol{\beta}^{(k-1)})-\lambda(1-\alpha_k^2)\boldsymbol{\beta}^{(k-1)}\\ =&(\lambda\mathbf{I}+\alpha_k^2\mathbf{K}_{k-1}+\mathbf{H}_k^{\mathrm{T}}\mathbf{H}_k)\boldsymbol{\beta}^{(k-1)}+\mathbf{H}_k^{\mathrm{T}}(\mathbf{T}_k-\mathbf{H}_k\boldsymbol{\beta}^{(k-1)})\\ &-\lambda(1-\alpha_k^2)\boldsymbol{\beta}^{(k-1)}\\ =&\alpha_k^2(\lambda\mathbf{I}+\mathbf{K}_{k-1})\boldsymbol{\beta}^{(k-1)}+\mathbf{H}_k^{\mathrm{T}}\mathbf{T}_k\quad(k\geq 1).\end{aligned}\tag{26}$$

Combine with $(\lambda\mathbf{I}+\mathbf{K}_0)\boldsymbol{\beta}^{(0)}=\mathbf{H}_0^{\mathrm{T}}\mathbf{T}_0$, we get

$$(\lambda\mathbf{I}+\mathbf{K}_n)\boldsymbol{\beta}^{(n)}=\sum_{k=0}^{n-1}(\prod_{j=k+1}^{n}\alpha_j^2\mathbf{H}_k^{\mathrm{T}}\mathbf{T}_k)+\mathbf{H}_n^{\mathrm{T}}\mathbf{T}_n\quad(n\geq 1).\tag{27}$$

From equations (25) and (27), we have

$$\begin{aligned}\boldsymbol{\beta}^{(n)}=&\Big(\lambda\mathbf{I}+\sum_{k=0}^{n-1}(\prod_{j=k+1}^{n}\alpha_j^2\mathbf{H}_k^{\mathrm{T}}\mathbf{H}_k)+\mathbf{H}_n^{\mathrm{T}}\mathbf{H}_n\Big)^{-1}\\ &\Big(\sum_{k=0}^{n-1}(\prod_{j=k+1}^{n}\alpha_j^2\mathbf{H}_k^{\mathrm{T}}\mathbf{T}_k)+\mathbf{H}_n^{\mathrm{T}}\mathbf{T}_n\Big)\quad(n\geq 1).\end{aligned}\tag{28}$$

Let $\mathbf{H}_n^*=(\prod_{j=1}^{n}\alpha_j\mathbf{H}_0^{\mathrm{T}},...,\alpha_n\mathbf{H}_{n-1}^{\mathrm{T}},\mathbf{H}_n^{\mathrm{T}})^{\mathrm{T}}$ and $\mathbf{T}_n^*=(\prod_{j=1}^{n}\alpha_j\mathbf{T}_0^{\mathrm{T}},...,\alpha_n\mathbf{T}_{n-1}^{\mathrm{T}},\mathbf{T}_n^{\mathrm{T}})^{\mathrm{T}}$. Then

$$\boldsymbol{\beta}^{(n)}=(\lambda\mathbf{I}+\mathbf{H}_n^{*\mathrm{T}}\mathbf{H}_n^*)^{-1}\mathbf{H}_n^{*\mathrm{T}}\mathbf{T}_n^*.\tag{29}$$

This means $\boldsymbol{\beta}^{(n)}$ are the minima of function

$$\begin{aligned}L_n^{**}(\boldsymbol{\beta})=&\frac{\lambda}{2}\|\boldsymbol{\beta}\|^2+\frac{1}{2}\|\mathbf{H}_n^*\boldsymbol{\beta}-\mathbf{T}_n^*\|^2\\ =&\frac{\lambda}{2}\|\boldsymbol{\beta}\|^2+\frac{1}{2}\left\|\begin{pmatrix}\prod_{j=1}^{n}\alpha_j\mathbf{H}_0\\ \vdots\\ \mathbf{H}_n\end{pmatrix}\boldsymbol{\beta}-\begin{pmatrix}\prod_{j=1}^{n}\alpha_j\mathbf{T}_0\\ \vdots\\ \mathbf{T}_n\end{pmatrix}\right\|^2\\ =&\frac{\lambda}{2}\|\boldsymbol{\beta}\|^2+\frac{1}{2}\left\|\begin{matrix}\prod_{j=1}^{n}\alpha_j(\mathbf{H}_0\boldsymbol{\beta}-\mathbf{T}_0)\\ \vdots\\ \mathbf{H}_n\boldsymbol{\beta}-\mathbf{T}_n\end{matrix}\right\|^2\\ =&\frac{\lambda}{2}\|\boldsymbol{\beta}\|^2+\frac{1}{2}(\sum_{k=0}^{n-1}\|\prod_{j=k+1}^{n}\alpha_j(\mathbf{H}_k\boldsymbol{\beta}-\mathbf{T}_k)\|^2+\|\mathbf{H}_n\boldsymbol{\beta}-\mathbf{T}_n\|^2)\\ =&L_n^*(\boldsymbol{\beta}).\end{aligned}\tag{30}$$

□

Proposition 2. *For any integer n ($n\geq 1$), if the forgetting parameters $\{\alpha_k\}_{k=1,2,...,n}$ are allowed to be any positive values, then the arrived data chunks $\{\aleph_k\}_{k=0,1,...,n-1}$ can be assigned with any positive weights (importance), and the weight of the new data chunk $\aleph_n$ is 1.*

Proof. According to Proposition 1, the weight of $\aleph_k$ $(k=0,1,...,n)$ can be written as

$$w_k=\begin{cases}\prod_{j=k+1}^{n}\alpha_j, & 0\leq k\leq n-1\\ 1, & k=n\end{cases}.\tag{31}$$

Then, for any arbitrarily given weights w_k ($w_k>0, k=0,1,...,n-1$), the weight assignment can be achieved by setting $\alpha_k=w_{k-1}/w_k$ $(k=1,2,...,n)$. □

Propositions 1 and 2 imply that data chunks arriving at different periods can be assigned with different weights (importance) by using the recursive formulas (22) and (23). In fact, we assume that the new data are more important for learner training; that is, we restrict $0 \leq \alpha_k \leq 1$ ($k \in N_+$). So, the weight assignment mechanism is actually a forgetting mechanism; the effect of the old data will reduce gradually during the training phase. The smaller the new forgetting parameter α_k is, the smaller the weights of all previous data chunks will be and the more attention the new data chunk will be paid to, and vice versa.

Another question is how to determine the forgetting parameter α_k ($k \in N_+$). Generally speaking, if the training samples are already sufficient, then the validation error of OS-ELM keeps stable during the following training phase [16]. And if the accuracy of the learner has an obvious drop on the new data, concept drift is likely to happen. So, we choose a relatively large forgetting parameter α_k if the current learner performs well on the new data; otherwise, we choose a relatively small α_k. In many ensemble methods, the performance on new data is regard as an important standard for tuning the weight of each base learner [25, 27, 28, 29, 30]. From all the above, when a new data chunk $\aleph_n$ ($n \in N_+$) arrives, the new forgetting parameter α_n is given by

$$\alpha_n = reg(\theta - \eta * \boldsymbol{Error}_n), \tag{32}$$

where $\boldsymbol{Error}_n$ is the current learner's mean error rate or mean square error on the new data chunk $\aleph_n$ (we assume the data samples all have been normalized), function $reg(x) = \begin{cases} 0, & x < 0 \\ x, & 0 \leq x \leq 1 \\ 1, & x > 1 \end{cases}$ regulates α_n into the range [0,1], and (θ, η) are control parameters. Parameter θ represents the base forgetting rate for the old data, and η represents α_n's sensitivity to the current error.

Remark 1. The current error $\boldsymbol{Error}_n$ may become meaningless due to its high randomicity if $\aleph_n$ is too small (e.g., $N_n = 1$). Under this condition, we can calculate $\boldsymbol{Error}_n$ based on a few more samples, e.g., the most recent 10 samples. We can even set $\eta = 0$ and only use $\alpha_n = \theta$ to perform the forgetting mechanism.

Remark 2. Different from OS-ELM, the output weights of FP-ELM are recursively calculated based on the regularized optimization method. The regularized optimization method here is vital for FP-ELM and it is introduced to avoid a numerical problem which is referred to as windup. Details will be demonstrated in section 4.1. What's more, $\lambda\mathbf{I} + \mathbf{H}_0^{\mathrm{T}}\mathbf{H}_0$ must be nonsingular whether $N_0 \geq L$ or $N_0 < L$ thanks to the regularization parameter λ. So there are no restrictions on the number of initial training samples N_0 in FP-ELM.

Remark 3. The time and space cost of algorithm FP-ELM is reasonable. Similar to OS-ELM, all the information of the previous data is stored in $\mathbf{K}$ and $\boldsymbol{\beta}$, which are matrices with a fixed size. When the new data chunk $\aleph$ arrives, we need to fill up its hidden-layer output matrix $\mathbf{H}$ and target output $\mathbf{T}$ first and then update $\mathbf{K}$ and $\boldsymbol{\beta}$ by formulas (22) and (23). If the size of the data chunk is fixed during training, the space consumption will remain constant, and the time cost of each step will be almost constant, too.

Remark 4. There are some other ELM based non-stationary learning methods. Zhao *et al.* proposed FOS-ELM and used it in stock price short-term predictions [36]. In FOS-ELM, the learner is always constructed on the latest s data chunks, and s is fixed during training. Rather than this "sliding window of fixed size" mechanism, FP-ELM uses the forgetting mechanism, which is smoother and tends to achieve a balance between stability and plasticity [37]. In [38], a variable forgetting factor (FF) is introduced to OS-ELM to forget old samples gradually. The FF is adjusted iteratively to obtain an appropriate FF value. Recently, a new FF based online ELM model, called λ_{DFF}OS-ELM, is proposed by Soares *et al.* [31]. λ_{DFF}OS-ELM calculates the FF value and avoids windup all by using the Directional Forgetting Factor (DFF) method. But it can only learn samples one-by-one. Quite analogous to these two models, FP-ELM use similar forgetting parameters to discount or forget the old data continuously. However, FP-ELM determines the forgetting parameters simply according to the real-time performance and overcome the windup problem by using the regularized method. Moreover, FP-ELM can learn data not only on a sample by sample basis but also in batches.

The FP-ELM algorithm can be summarized as follows.

Algorithm 1. Select the type of hidden nodes, the corresponding activation function g, the hidden nodes number L and the user-specified parameters of FP-ELM (the control parameters θ, η, and the regularization parameter λ). The data chunks $\{\aleph_k\}_{k=0,1,...}$ arrive sequentially.

(1) Randomly assign the hidden-layer parameters $\{(\mathbf{a}_i, b_i)|i = 1, 2, ..., L\}$;
(2) For the initial data chunk $\aleph_0$, calculate its hidden-layer output matrix $\mathbf{H}_0$;
(3) Initialize $\mathbf{K}_0 = \mathbf{H}_0^{\mathrm{T}}\mathbf{H}_0$ and $\boldsymbol{\beta}^{(0)} = (\lambda\mathbf{I} + \mathbf{K}_0)^{-1}\mathbf{H}_0^{\mathrm{T}}\mathbf{T}_0$;
(4) Set $k = 1$;
(5) For the kth data chunk $\aleph_k$, calculate its hidden-layer output matrix $\mathbf{H}_k$;
(6) Using $\aleph_k$ or some recent samples as a testing set, calculate the current learner's error $\boldsymbol{Error}_k$;
(7) Calculate the forgetting parameter $\alpha_k = reg(\theta - \eta * \boldsymbol{Error}_k)$;
(8) Update $\mathbf{K}_k = \alpha_k^2\mathbf{K}_{k-1}+\mathbf{H}_k^{\mathrm{T}}\mathbf{H}_k$ and $\boldsymbol{\beta}^{(k)} = \boldsymbol{\beta}^{(k-1)}+(\lambda\mathbf{I}+\mathbf{K}_k)^{-1}(\mathbf{H}_k^{\mathrm{T}}(\mathbf{T}_k-\mathbf{H}_k\boldsymbol{\beta}^{(k-1)})-\lambda(1-\alpha_k^2)\boldsymbol{\beta}^{(k-1)})$;
(9) Set $k = k + 1$, go to (5);

4. Performance evaluation of FP-ELM

In this section, the new proposed algorithm is evaluated on four kinds of concept drift data sets; both classification and regression cases are involved. For all the data sets, the input attributes are normalized into the range [-1,1]. The targets in regression problems are normalized into [-1,1], too. All the experiments are carried out in MATLAB 7.0 environment running on a laptop with 2.39GHz CPU and 1.98GB RAM.

To deal with concept drift problems, ensemble methods, such as streaming ensemble algorithm (SEA) [24] and additive expert ensemble (AddExp) [28], are frequently used. In the experiments, we compare FP-ELM with SEA and AddExp on the four kinds of data sets. Both SEA and AddExp use ELMs as base learners. In AddExp, the ensemble members are required to learn incrementally, and the OS-ELM algorithm makes this possible. In addition, we also take the performance of OS-ELM as a comparison, which represents the bad result we get without drift processing strategies.

The user-specified parameters of each algorithm are: ensemble size M for SEA; (τ, β, γ) and ensemble size M for AddExp; (θ, η) for FP-ELM. For all benchmark problems, the optimal parameters of each algorithm are specified based on large numbers of trials with different parameter values. For AddExp, β and γ are chosen from $\{0.01, 0.05, 0.1, 0.2, \cdots, 0.9\}$ and ensure that $\beta + \gamma < 1$; τ is chosen from $\{0.01, 0.05, 0.1, 0.15, 0.2\}$. For FP-ELM, when the chunk size is 1, we simply set $\eta = 0$ and choose θ from the range $\{0.05, 0.10, 0.15, \cdots, 0.95, 0.955, 0.960, 0.965, \cdots, 0.995\}$. Otherwise, let $\theta = \overline{e}/(\overline{e}-\underline{e})$ and $\eta = 1/(\overline{e}-\underline{e})$. That is to say, the current forgetting parameter $\alpha_n \equiv 1$ if the current learner's error is below $\underline{e}$, and $\alpha_n \equiv 0$ if the error is above $\overline{e}$. To determine θ and η, we pick $\underline{e}$ and $\overline{e}$ from $\{0.0, 0.05, 0.1, 0.2, 0.3, \cdots, 1.0\}$ and keep $\underline{e} < \overline{e}$. After the parameters are determined, we take the average results of 50 independent trials as the final results. For each benchmark case, the best testing result (testing error) among the results of all the testing algorithms will be shown in boldface in the tables. But if there exists another testing result which is so similar to the best one that their relative difference is below 2%, this result and the best one will all be shown underlined.

Many types of hidden nodes can be used by ELM. For the sake of simplicity, we choose the additive hidden nodes with sigmoidal activation function for all the experiments. Then the hidden node output function can be written as

$$G(\mathbf{a}, b, \mathbf{x}) = \frac{1}{1 + \exp(-(\mathbf{a} \cdot \mathbf{x} + b))}. \tag{33}$$

Studies show that the generalization performance of ELM is not very sensitive to the number of hidden nodes [6, 35]. According to some simple trials, the hidden node number is uniformly set to be 30 in our experiments.

4.1. The impact of regularization parameter λ

In the training phase, the matrix $\mathbf{K}_k$ is constantly updated and some of its eigenvalues may become unacceptable large. As a result, the condition number of $\mathbf{K}_k$ may be too large to calculate the inverse $(\mathbf{K}_k)^{-1}$ accurately. This phenomenon is called windup in automatic control theory. In fact, this case rarely happens in OS-ELM for real applications and the inverse is calculated using singular value decomposition (SVD) even if the relevant matrix is singular or near singular. But in FP-ELM, $\mathbf{K}_k$ usually becomes near singular, especially after a large number of iterations, due to the use of forgetting parameters. Thus, a regularization parameter λ is introduced to improve the condition number in matrix inversion and then $(\lambda\mathbf{I} + \mathbf{K}_k)^{-1}$ is calculated instead of $(\mathbf{K}_k)^{-1}$. We demonstrate this by an example and observe how λ impacts the generalization performance of FP-ELM.

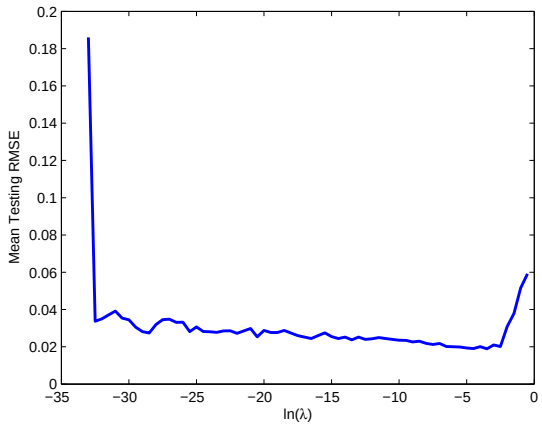


Figure 1: The relationship between $\ln(\lambda)$ and the testing error of FP-ELM.

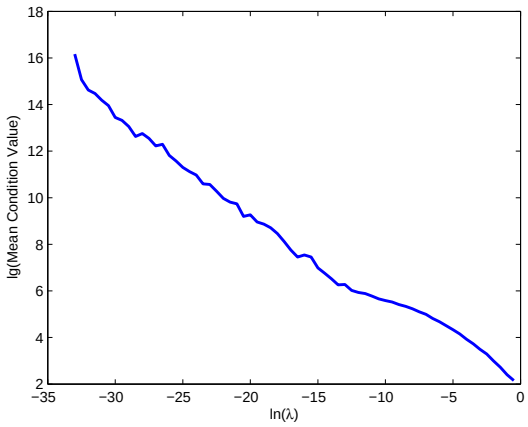


Figure 2: The relationship between $\ln(\lambda)$ and the average condition number.

The example is based on a concept drifting regression data set which is introduced in section 4.4. We first choose a relatively large initial hidden node number N_0 so that $\text{rank}(\mathbf{K}_0) = \text{rank}(\mathbf{H}_0) = L$ and $\mathbf{K}_0$ has a reasonable condition number. Then $\mathbf{K}_k$ $(k \geq 0)$ keeps positive definite during training, and $\lambda\mathbf{I} + \mathbf{K}_k$ is always nonsingular no matter how small λ is. Fig. 1 shows the testing root mean square error (RMSE) curve of FP-ELM with different values of $\ln(\lambda)$. It can be seen that the generalization performance is bad when λ is too small, and it varies inconspicuously over a wide range of λ.

Assume $\lambda_{\min}$ and $\lambda_{\max}$ are the minimal and the maximal eigenvalues of $\mathbf{K}_k$, respectively. Then, the condition number of $\lambda\mathbf{I} + \mathbf{K}_k$ is

$$\text{cond}(\lambda\mathbf{I} + \mathbf{K}_k) = (\lambda_{\max} + \lambda)/(\lambda_{\min} + \lambda). \tag{34}$$

The smaller λ is, the larger $\text{cond}(\lambda\mathbf{I}+\mathbf{K}_k)$ is and the closer $\text{cond}(\lambda\mathbf{I}+\mathbf{K}_k)$ is to $\text{cond}(\mathbf{K}_k)$. Fig. 2 also shows the negative correlation between λ and the average condition number $\text{avg}\{\text{cond}(\lambda\mathbf{I}+\mathbf{K}_i)\}_{i=0,1,\ldots,400}$. When λ is small, $\lambda\mathbf{I} + \mathbf{K}_k$ could be near singular for some large k. This is the reason why the algorithm performs poorly. In this example, if we further decrease regularization parameter λ ($\ln(\lambda)$<-33.0), an error that the matrix is singular to working precision will be reported by MATLAB.

In fact, similar conclusions can be drawn from other data sets. In FP-ELM, $\mathbf{K}_k$ may become ill-conditioned during training. So regularization parameter λ is necessary for FP-ELM, and it should not be too small. On the other hand, the generalization performance of FP-ELM is not sensitive to λ over a wide range of λ. We simply choose $\lambda = 0.02$ for all the experiments below and its reasonability will be verified by the experimental results.

4.2. SEA concept problem

The SEA concept describes a kind of data set with sudden drift. That is, the pattern of the data in a data stream may change totally at some time. It was proposed by Street *et al.* [24] and has been used by other researchers as a standard test for non-stationary learning algorithms. The SEA concept was originally designed for classification problems, and Kolter *et al.* [28] used it in regression problems.

The data set we use is created following the same way as in [28]. There are four concepts and 2,000 training samples per concept. The data chunk size is set to be 20. So there are 400 time steps in total, and each concept lasts for 100 steps. And 200 samples from the current concept are used for testing at each time step. The samples are to simulate a 10-dimensional drifting hyperplane, and the input attributes $\mathbf{x} = (x_1, x_2, ..., x_{10})$ are randomly generated from $[-1, 1]^{10}$. For classification case, the class label is $t = \text{sign}(x_i + x_{i+1} + x_{i+2})$ with i=1, 2, 4, and 7 for each concept. 10% random class noise is introduced into the training set; that is, 10% of the training data reverse their labels. For regression case, the target output is $t = (x_i + x_{i+1} + x_{i+2})/3$ with i=1, 2, 4, and 7 for each concept. We also introduce ± 0.1 random noise to t.

Moreover, we also use real-world data sets to construct SEA concepts. Three classification applications, Contraceptive Method Choice (1,460 instances), Iris (150 instances) and Car Evaluation (1,720 instances), and three regression applications, Abalone (4,160 instances), AutoMPG (380 instances) and Concrete Compressive Strength (1,020 instances), from UCI Machine Learning Repository [39] are chosen in our experiments. A concept is constructed by altering the class labels or targets of the samples in a raw data set. For all the cases, there're four concepts, too. To obtain four different concepts, four ways are used to change the original class labels into new labels (e.g., 1, 2, 3) for classification cases [32]. And four functions are used to transform the original target values into new values for regression cases. The original class label types or target value and the new label types or target value of each concept for all the data sets are shown in Table 1. For each data set, all the samples are chosen to form the training set of each concept. The chunk size is 10 for Iris and 20 for other data sets. At each time step, a number of samples (150 samples for Iris and 200 samples for other data sets) are randomly chosen to form the testing set according to the current concept.

Table 1: Concepts (C_i) used to create the drifting data sets.

Data sets	Original class/target	C_1	C_2	C_3	C_4
Contraceptive.	No-use	2	3	1	2
	Long-term	3	1	2	3
	Short-term	1	2	3	1
Iris	Setosa	2	3	1	2
	Versicolour	3	1	2	3
	Virginica	1	2	3	1
Car.	Unacc	-1	1	-1	1
	Acc	1	-1	1	-1
	Good	1	-1	-1	1
	Very Good	1	-1	-1	1
Abalone AutoMPG Concrete.	t	t	$\frac{t+1}{2}$	$\frac{t-1}{2}$	$-t$

In the experiments, another strategy for comparison is presented as the best result of a single learner. The learner is trained and updated by OS-ELM when there is no concept change, and it will be retrained if drift occurs. For both SEA(ELM) and AddExp(ELM), the ensemble size M is set to be 10. Other parameter settings of each algorithm on each data set are shown in Table 2. Fig. 3 displays the behavior of the testing error during learning for some cases. The average testing error and the total training time over all the time steps of each algorithm for each case are given in Table 3.

Seen from Fig. 3, when the concept changes from one to another (e.g., at the 100th, 200th or 300th time step in Fig. 3(a)), AddExp(ELM) and FP-ELM can adapt to the new concept quickly. And seen from the mean results, AddExp(ELM) and FP-ELM perform better than SEA(ELM) for most cases. It is worth noticing that AddExp(ELM) obviously performs better than FP-ELM and the perfect method of single ELM for some cases (e.g., hyperplane (regression), Car.). And for hyperplane(regression) case, SEA(ELM)

Table 2: Parameters of AddExp(ELM) and FP-ELM on SEA concept problems.

Problems	AddExp(ELM)	FP-ELM
Classification	(β, γ)	(θ, η)
Hyperplane	(0.8,0.01)	(2.0,3.333)
Contraceptive.	(0.9,0.05)	(4.5,5.0)
Iris	(0.6,0.05)	(1.5,2.5)
Car.	(0.8,0.01)	(2.0,3.333)
Regression	(τ, β, γ)	(θ, η)
Hyperplane	(0.05,0.5,0.1)	(1.5,5.0)
Abalone	(0.15,0.7,0.05)	(1.5,2.5)
AutoMPG	(0.1,0.7,0.1)	(1.5,2.5)
Concrete.	(0.2,0.7,0.1)	(1.75,2.5)

Table 3: Comparison between FP-ELM and other algorithms on SEA concept problems.

Problems	OS-ELM		OS-ELM on each concept	SEA(ELM)		AddExp(ELM)		FP-ELM	
Classification	Error Rate (%)	Time(s)	Error Rate (%)	Error Rate(%)	Time(s)	Error Rate(%)	Time(s)	Error Rate(%)	Time(s)
Hyperplane	27.24	0.0903	7.48	9.90	0.5072	**6.21**	8.9869	8.18	0.1119
Contraceptive.	62.50	0.0775	<u>46.99</u>	52.40	0.4144	48.44	7.2813	<u>47.88</u>	0,0878
Iris	48.23	0.0122	5.63	19.57	0.0288	**5.07**	0.5378	5.76	0.0147
Car.	49.70	0.0822	11.98	17.87	0.3216	**9.98**	6.0531	11.99	0.0972
Regression	RMSE	Time(s)	RMSE	RMSE	Time(s)	RMSE	Time(s)	RMSE	Time(s)
Hyperplane	0.2291	0.0922	0.0400	0.0415	0.3672	**0.0245**	5.1953	0.0402	0.1044
Abalone	0.4426	0.1894	<u>0.1258</u>	0.1716	0.8369	0.1334	17.9300	<u>0.1283</u>	0.2203
AutoMPG	0.4626	0.0191	<u>0.1136</u>	0.1835	0.0397	<u>0.1127</u>	1.4131	<u>0.1138</u>	0.0213
Concrete.	0.4561	0.0494	0.1746	0.2250	0.1331	**0.1706**	3.6334	0.1775	0.0500

and AddExp(ELM) have a lower testing error, compared with FP-ELM, after all the algorithms adapt to a new concept, as shown in Fig. 3(b). This is probably because SEA(ELM) and AddExp(ELM) benefit from their ensemble strategies but FP-ELM uses the single learner method. However, FP-ELM achieves a similar performance to the perfect method of single ELM on all the data sets. This means that FP-ELM can cope with this kind of drift well. What's more, FP-ELM has a much shorter training time than that of SEA(ELM) and AddExp(ELM) for all the cases. And FP-ELM is only slightly more time-consuming than the original OS-ELM. Time consumption may be not a primary concern in many studies of concept drift, but these facts imply FP-ELM is a simple and efficient algorithm.

4.3. Moving hyperplane problem

A moving hyperplane can be used to create a gradual changing concept, and it is originally from [40]. Many researchers employ it as a generator of drifting data streams [20, 41].

The synthetic data are created as follows. We generate random input attributes $\mathbf{x} = (x_1, x_2, ..., x_{10})$ uniformly distributed in $[-1, 1]^{10}$, and the class label $t = \text{sign}(\sum_{i=1}^{10} a_i x_i - a_0)$. Coefficients a_i (i=1,2,...,10) associated with the input attributes are initialized randomly in [0,2], and $a_0 = 0.07 \sum_{i=1}^{10} a_i$ so that the positive class and the negative class almost have the same size. The method can be easily extended to regression case by setting $t = \sum_{i=1}^{10} a_i x_i$. We simulate concept drifts by gradually changing the parameters a_i (i=1,2,...,10). After the generation of each sample, a_i is adjusted by $0.05 s_i$, where $s_i \in \{-1, 1\}$ is the adjustment direction of a_i. And then, s_i would invert its direction with a probability of 0.2. The total number of the samples is 4,010. After the generation process is complete, we introduce noise by randomly reversing the class labels of 2% of the data or adding ±0,02 random variates to the outputs.

We also consider another simpler 4-dimensional hyperplane [23]. Coefficients a_i (i=1,2,3,4) are initialized by $a_1(0) = 0.5$, $a_2(0) = 0.2$, $a_3(0) = 0.7$ and $a_4(0) = 0.8$. After each sample is generated, the coefficients evolve according to

$$a_i(n) = a_i(n-1) + \frac{n}{10^{4.7}} \text{logsig}(a_i(n-1)), i = 1, 2, 3, 4, n = 1, 2, ..., N-1, \tag{35}$$

where the total sample number is $N = 500$.

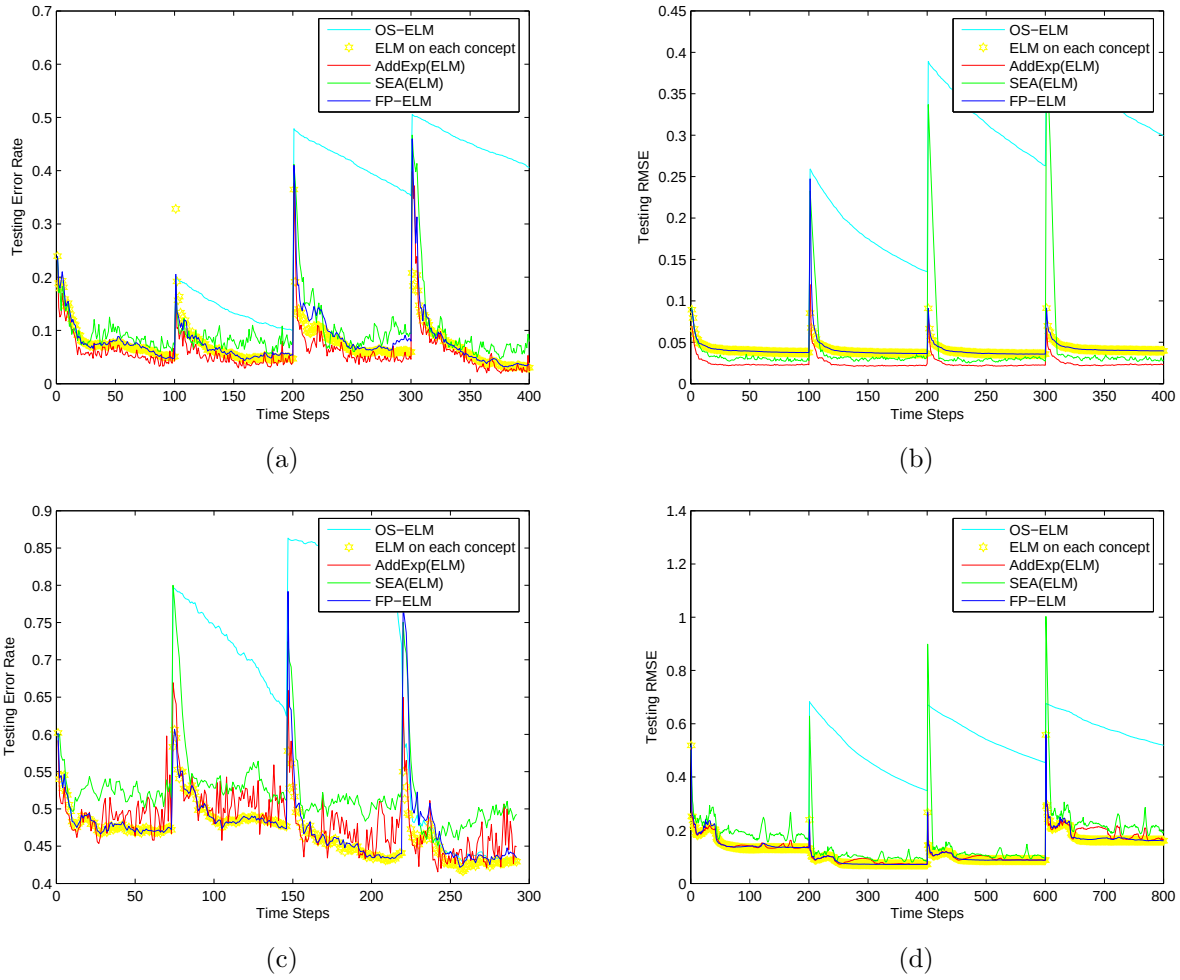


Figure 3: Testing error curves on SEA concept problems. (a)Hyperplane (classification) (b)Hyperplane (regression) (c)Contraceptive. (d)Abalone

For the moving hyperplane problems, the concept changes constantly and it is unrealistic to construct a sufficient testing set that reflects the exact concept of the training set. In this part of the experiments, the algorithms are tested in this way: the current learner is frequently tested by the next chunk of data to be learnt. To choose a reasonable ensemble size for SEA(ELM) or AddExp(ELM), we try to add base learners one by one from zero until the performance of the ensemble is not notably improved. The chunk size is set to be 10, and we also try to shrink it to 1 so that the proposed method and the forgetting mechanism can be tested in online learning. In the SEA algorithm, a base learner is trained based on a data chunk; the base learners will be inaccurate if the chunk is too small. Hence, SEA(ELM) is inapplicable and eliminated in the experiments when the chunk size is 1.

Table 4: Parameters of SEA(ELM), AddExp(ELM) and FP-ELM on moving hyperplane problems.

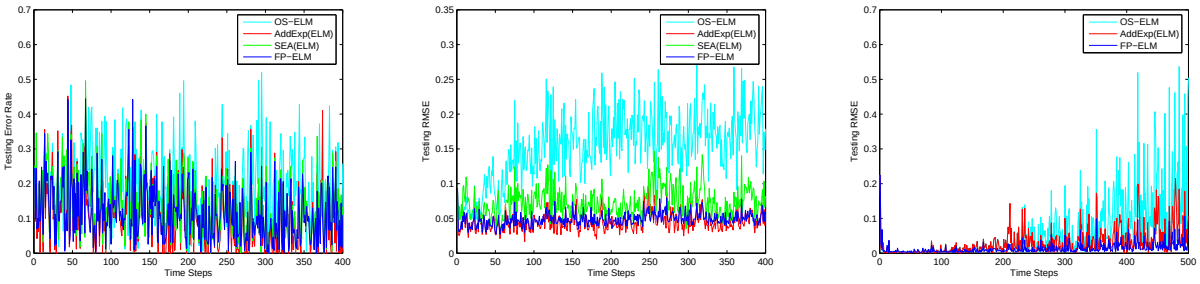
Problems	SEA(ELM)	AddExp(ELM)	FP-ELM
Classification	M	(β, γ, M)	(θ, η)
Hyperplane1(10)	15	(0.8,0.05,15)	(1.111,1.111)
Hyperplane1(1)	-	(0.6,0.05,15)	(0.995,0.0)
Hyperplane2(10)	10	(0.5,0.05,10)	(10.0,10.0)
Hyperplane2(1)	-	(0.5,0.05,15)	(0.995,0.0)
Regression	M	(τ, β, γ, M)	(θ, η)
Hyperplane1(10)	10	(0.05,0.01,0.3,10)	(1.0,3.333)
Hyperplane1(1)	-	(0.1,0.05,0.5,15)	(0.98,0.0)
Hyperplane2(10)	2	(0.01,0.01,0.9,20)	(1.0,20.0)
Hyperplane2(1)	-	(0.05,0.01,0.5,25)	(0.95,0.0)

Table 5: Comparison between FP-ELM and other algorithms on moving hyperplane problems.

Problems	OS-ELM		SEA(ELM)		AddExp(ELM)		FP-ELM	
Classification	Error Rate (%)	Time(s)	Error Rate(%)	Time(s)	Error Rate(%)	Time(s)	Error Rate(%)	Time(s)
Hyperplane1(10)	19.53	0.0819	13.69	0.4409	**11.72**	6.3794	11.97	0.0988
Hyperplane1(1)	23.62	0.6206	-	-	12.50	6.7106	12.68	0.7009
Hyperplane2(10)	7.01	0.0113	6.80	0.0341	6.16	0.5369	6.28	0.0125
Hyperplane2(1)	6.72	0.0794	-	-	5.96	0.8694	6.04	0.0906
Regression	RMSE	Time(s)	RMSE	Time(s)	RMSE	Time(s)	RMSE	Time(s)
Hyperplane1(10)	0.1534	0.0800	0.0737	0.3272	**0.0456**	4.5687	0.0489	0.1019
Hyperplane1(1)	0.1262	0.6291	-	-	0.0456	6.5341	**0.0374**	0.7056
Hyperplane2(10)	0.0850	0.0122	0.0237	0.0184	0.0225	1.1128	**0.0216**	0.0122
Hyperplane2(1)	0.0644	0.0800	-	-	0.0304	1.3241	**0.0139**	0.0891

The parameter settings are given in Table 4. Table 5 displays the mean results over all the time steps. In these experiments, the concept changes all the time, and the testing error may not have a clear trend. Therefore, the mean performance is a relatively reasonable standard of evaluating the algorithms. We also give the curves of the testing error with time steps on some data sets. The error curves for the first kind of moving hyperplane with chunk size 10 in classification case and regression case are shown in Fig. 4(a) and Fig. 4(b), respectively. Fig. 4(c) shows the curves for the second kind of moving hyperplane with chunk size 1 in regression case.

Seen from the mean performance, FP-ELM outperforms SEA(ELM) and achieves a similar performance to AddExp(ELM), for both the chunk size is 10 and is 1. In Figs. 4(b)-(c), the error curve of FP-ELM is smoother than those of other algorithms. This implies that FP-ELM can adapt to the changes better. For classification case, the testing error usually has a intensive vibration during learning, just as we see in Fig. 4(a). The situation is more serious when the chunk size shrinks to 1. This makes the figure unreadable. This may be due to the rough 0-1 error in classification problems. The absolute "1" error is produced once the testing sample is misclassified by the learner. Similar phenomena can also be seen from Fig. 3. Meanwhile, FP-ELM has the lowest time cost among the three non-stationary environment algorithms.



(a) (b) (c)

Figure 4: Testing error curves on moving hyperplane problems. (a)Hyperplane1($ChunkSize$=10,classification) (b)Hyperplane1($ChunkSize$=10,regression) (c)Hyperplane2($ChunkSize$=1,regression)

4.4. Regression problem with gradual drift

Many studies about concept drift focus on classification problems. However, from the experiments above, we find that the traces of the error curves are clearer in regression case. Hence, we design a kind of drifting data set for regression problems and test the proposed method on it.

Usually, we regard conventional regression data sets as stationary data sets; that is, there is a unique mapping relation between the inputs and the output in a data set. We can make a drift by adding a changing bias to the target. Concretely speaking, let t_i ($t_i \in [-1, 1]$) denote the target of the ith sample in a data set. Then the new output after drift is introduced can be written as

$$t_i^{'} = \frac{1}{2}(t_i + \sin(\frac{2\pi(i-1)}{N-1})), i = 1, 2, ..., N, \tag{36}$$

where N is the total number of the samples. In this way, the mapping relation between the inputs and the new output changes gradually with the sample sequence number. Four ordinary data sets are chosen to construct four drifting data sets. The first one is a synthetic set. The inputs $\mathbf{x} = (x_1, x_2, ..., x_{10})$ are randomly generated from $[-1, 1]^{10}$, the output $t = \sum_{i=1}^{10} x_i$ and there are 4,010 samples generated in total. The other three are real world data sets: Abalone (4,160 instances), AutoMPG (380 instances) and Space-ga (3,000 instances). Abalone and AutoMPG are from UCI Machine Learning Repository, and Space-ga is from StaLib [42].

Table 6: Parameters of SEA(ELM), AddExp(ELM) and FP-ELM on regression problems with gradual drift.

Problems	SEA(ELM)	AddExp(ELM)	FP-ELM
	M	(τ, β, γ, M)	(θ, η)
Linear(10)	5	(0.01,0.01,0.9,25)	(1.0,20.0)
Linear(1)	-	(0.01,0.01,0.9,25)	(0.95,0.0)
Abalone(10)	5	(0.05,0.6,0.3,20)	(1.0,2.0)
Abalone(1)	-	(0.05,0.1,0.3,20)	(0.98,0.0)
AutoMPG(10)	5	(0.1,0.05,0.4,15)	(1.0,5.0)
AutoMPG(1)	-	(0.1,0.1,0.4,10)	(0.9,0.0)
Space-ga(10)	3	(0.05,0.01,0.9,15)	(1.0,5.0)
Space-ga(1)	-	(0.05,0.05,0.8,15)	(0.96,0.0)

As in section 4.3, the chunk size is set to be 10 or 1, and we always use the next chunk to be learnt to test the algorithms. A reasonable ensemble size is selected for SEA(ELM) and AddExp(ELM). For different data sets, the parameter settings are shown in Table 6, and the mean testing error and the total training time of each algorithm for each case are shown in Table 7. Fig. 5 show the error curves of each algorithm on some data sets.

Generally, we can draw the conclusion that FP-ELM and AddExp(ELM) have a similar performance, which is better than that of SEA(ELM) for both the chunk size is 10 and is 1. In the synthetic data case,

Table 7: Comparison between FP-ELM and other algorithms on regression problems with gradual drift.

Problems	OS-ELM		SEA(ELM)		AddExp(ELM)		FP-ELM	
	RMSE	Time(s)	RMSE	Time(s)	RMSE	Time(s)	RMSE	Time(s)
Linear(10)	0.2943	0.0844	0.0305	0.2053	0.0186	10.9172	0.0186	0.0875
Linear(1)	0.2907	0.6166	-	-	0.0191	11.6578	**0.0144**	0.7059
Abalone(10)	0.3082	0.0853	0.0866	0.2172	0.0821	9.5906	0.0826	0.0984
Abalone(1)	0.2965	0.6409	-	-	0.0634	9.8484	0.0631	0.7200
AutoMPG(10)	0.3276	0.0097	0.1644	0.0213	0.1203	0.6244	0.1210	0.0072
AutoMPG(1)	0.2856	0.0612	-	-	0.0725	0.4594	0.0725	0.0672
Space-ga(10)	0.2890	0.0587	0.0432	0.1150	0.0412	5.3847	0.0412	0.0650
Space-ga(1)	0.2738	0.4641	-	-	0.0290	5.3866	0.0291	0.5291

there is a simple linear relation between the inputs and the output, and it can be learnt by the learner quickly. So the testing error varies smoothly during learning as shown in Figs. 5(a)-(b). FP-ELM's error curve is smoother than those of other algorithms. The mapping relation in Abalone case is more complex and harder to learn, then the testing error has a more obvious vibration during learning, as we can see from Figs. 5(c)-(d). Moreover, the error curve of FP-ELM almost coincides with that of AddExp(ELM). This means that FP-ELM and AddExp(ELM) achieve almost the same real-time performance. The same observation is true for the other two real world data sets. Ensemble methods are usually considered to be more accurate than single learner methods in online and incremental learning. But for this kind of non-stationary regression problem, FP-ELM, as a single learner method, can achieve a comparable performance to the ensemble methods SEA(ELM) and AddExp(ELM) with a shorter learning time. FP-ELM's effectiveness is verified.

4.5. Time-series prediction problem

In this section, we consider two time-series prediction problems. The first time-series is from the numerical solution of the chaotic Mackey-Glass differential delay equation, and it is used to test the OS-ELM algorithm in [16]. We follow the settings in [16] and generate the series $s(t)$ ($t \in N_+$) according to

$$s(t+\Delta t) = \frac{2-b\Delta t}{2+b\Delta t}s(t) + \frac{a\Delta t}{2+b\Delta t}[\frac{s(t-\tau)}{1+s^{10}(t-\tau)} + \frac{s(t+\Delta t-\tau)}{1+s^{10}(t+\Delta t-\tau)}], \tag{37}$$

where $\Delta t = 1$, $\tau = 17$, $a = 0.2$ and $b = 0.1$. The initial condition is that $s(t) = 0.3$ for $t = 1, 2, ..., \tau + 1$. For prediction, the goal is to estimate the value of s $d = 50$ steps ahead using four past points. That is, the inputs and the output of the ith sample are $\mathbf{x}_i = (s(i), s(i+6), s(i+12), s(i+18))$ and $t_i = s(i+d+18)$, respectively. The total number of the samples is 4,010.

The second one is to simulate the vibrational behavior of a bearing, and it arises in [23] as non-stationary data. There are two signals in the time-series: revolutions per minute of the bearing (RPM) and the root mean square of the vibration (RMS). RPM is a real number generated randomly from 1,200 to 1,600 and changes every 30 steps. RMS is modelled by an exponential process:

$$RMS(t) = \frac{\exp(t/80)}{(t/10+100)^{2.4}} + \frac{RPM(t)}{2000} + 0.2\varepsilon, \tag{38}$$

where the noise $\varepsilon \sim N(0,1)$. In this case, for each step i, the target to be predicted is $t_i = RMS(i+d)$, and the two inputs are $\mathbf{x}_i = (RMS(i), RPM(i+d))$, where the time delay $d = 60$. The total sample number is 1,140.

As seen from the empirical study in [16], although OS-ELM can learn incrementally, it is not tested every time after the learner is updated. Rather, the entire testing phase occurs after the training phase. In [23], the researchers used online performance to evaluate their methods. In our experiments, the algorithms are tested by every chunk to be learnt, and the chunk size is 10 or 1, just as the settings in previous experiments. Table 8 displays the parameter settings of each algorithm, and Table 9 shows the mean testing error and training time comparisons between FP-ELM and other algorithms. The error curves of each algorithm for all benchmark problems can be seen from Fig. 6.

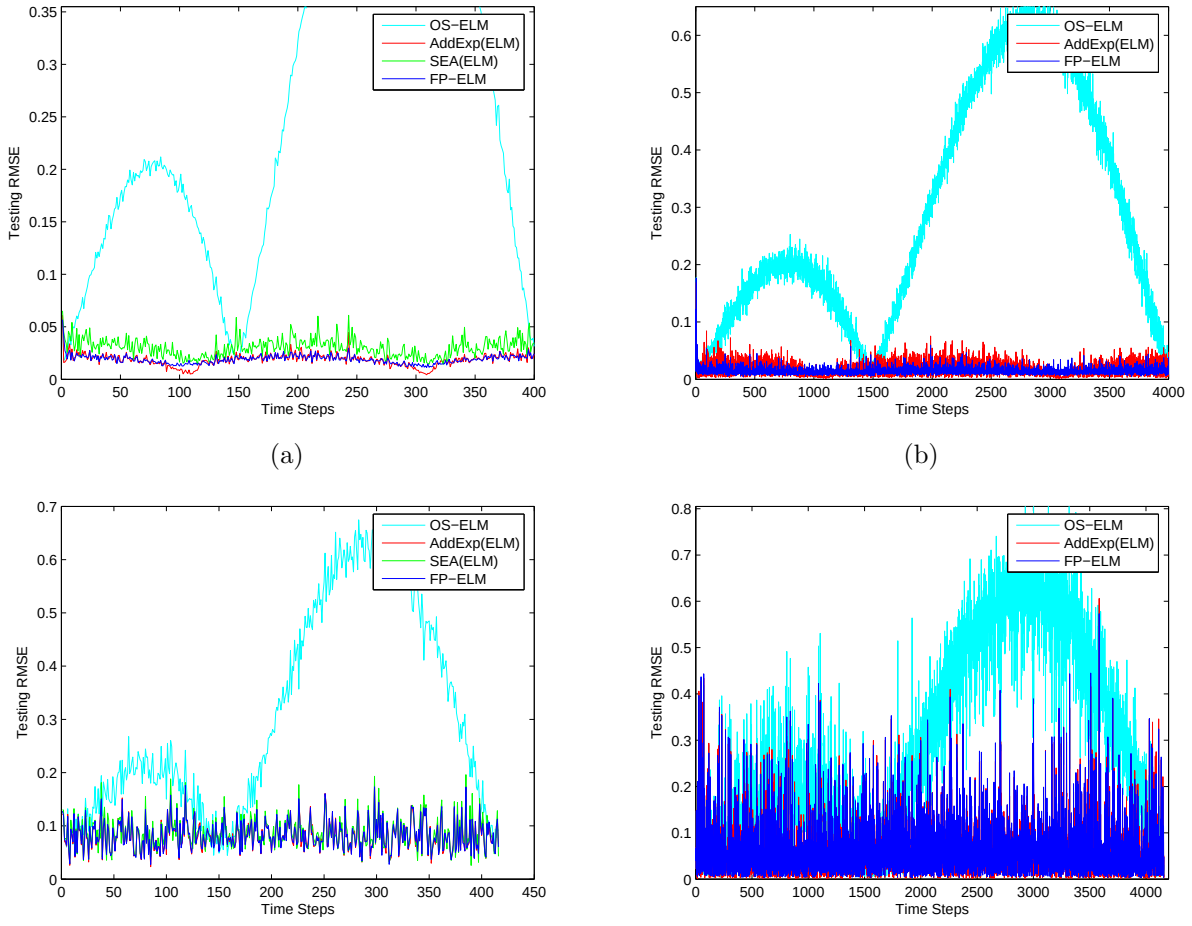


Figure 5: Testing error curves on regression problems with gradual drift. (a)Linear($ChunkSize$=10) (b)Linear($ChunkSize$=1) (c)Abalone($ChunkSize$=10) (d)Abalone($ChunkSize$=1)

Table 8: Parameters of SEA(ELM), AddExp(ELM) and FP-ELM on time-series prediction problems.

Problems	SEA(ELM)	AddExp(ELM)	FP-ELM
	M	(τ, β, γ, M)	(θ, η)
Time-series1(10)	10	(0.01,0.05,0.9,10)	(10.0,10.0)
Time-series1(1)	-	(0.01,0.05,0.9,25)	(0.05,0.0)
Time-series2(10)	2	(0.01,0.01,0.9,15)	(1.0,20.0)
Time-series2(1)	-	(0.01,0.01,0.9,20)	(0.85,0.0)

Table 9: Comparison between FP-ELM and other algorithms on time-series prediction problems.

Problems	OS-ELM		SEA(ELM)		AddExp(ELM)		FP-ELM	
	RMSE	Time(s)	RMSE	Time(s)	RMSE	Time(s)	RMSE	Time(s)
Time-series1(10)	0.2485	0.0791	0.3025	0.2975	0.2392	5.2234	0.2393	0.0900
Time-series1(1)	0.2150	0.6203	-	-	0.0602	11.4213	**0.0545**	0.6625
Time-series2(10)	0.0357	0.0213	0.0268	0.0325	0.0250	1.8700	0.0249	0.0275
Time-series2(1)	0.0322	0.1762	-	-	0.0184	2.7088	0.0181	0.1953

For the first time-series, FP-ELM and AddExp(ELM) achieve a similar performance when the chunk size is 10. But the non-stationary environment algorithms do not have a remarkable effect in this case. And SEA(ELM) performs even worse than those of OS-ELM, which has no strategies to deal with concept drift. Maybe the data chunk is too large to reflect the frequent changes in data. Due to the chaotic properties of this time-series, the testing error of each algorithm undulates greatly during learning as is shown in Fig. 6(a). When the chunk size is reduced to 1, the superiority of the non-stationary environment algorithms is significant and FP-ELM outperforms AddExp(ELM). This means the problem is obviously non-stationary, and FP-ELM can handle it well. Seen from Figs. 6(c)-(d), drifts mainly appear in the end part of the data for the second problem. FP-ELM and AddExp(ELM), or all the three non-stationary environment algorithms, have a quite similar effect in respect to both online performance and the average results. Meanwhile, FP-ELM works more efficiently.

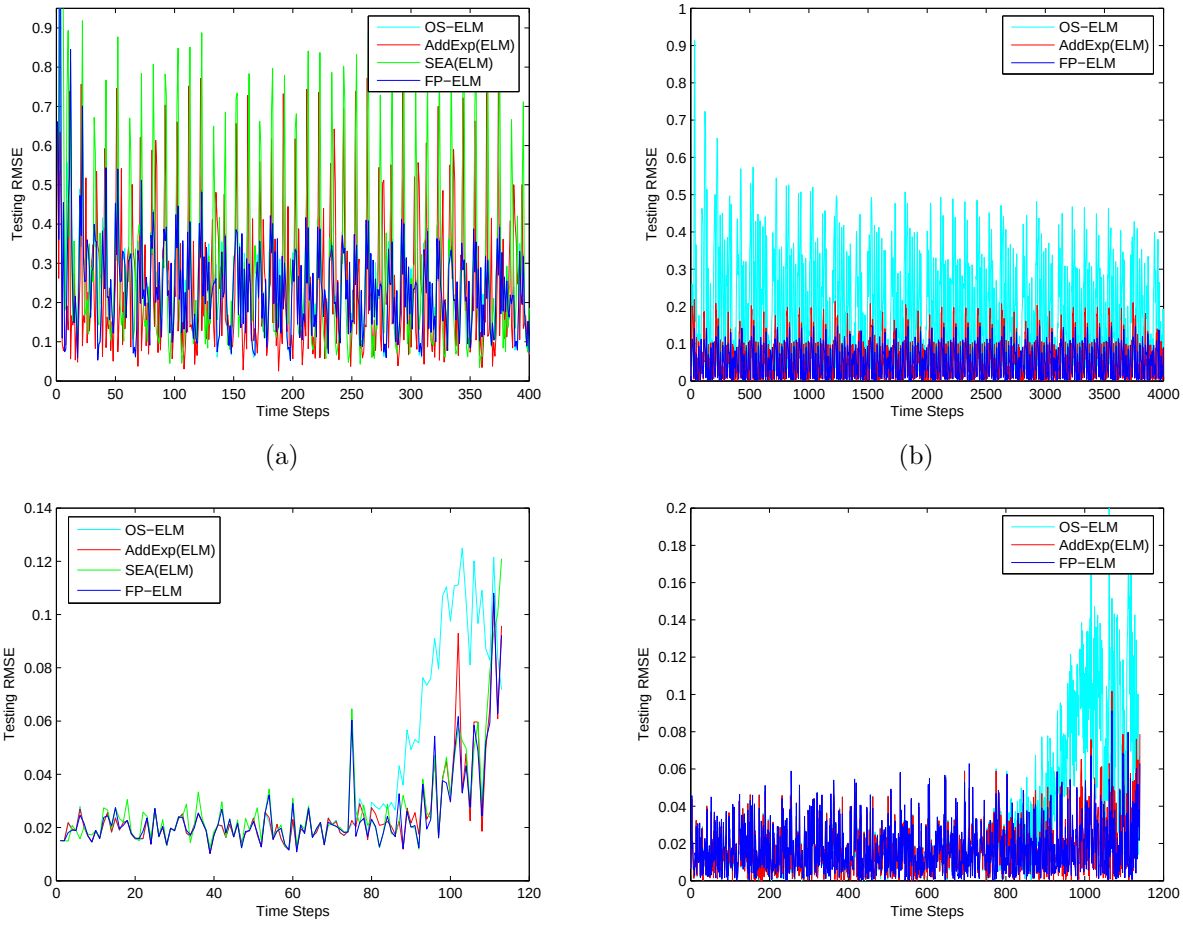


Figure 6: Testing error curves on time-series prediction problems. (a)Time-series1($ChunkSize$=10) (b)Time-series1($ChunkSize$=1) (c)Time-series2($ChunkSize$=10) (d)Time-series2($ChunkSize$=1)

5. Conclusion

In this paper, the forgetting parameters extreme learning machine (FP-ELM) has been developed to deal with concept drift problems. FP-ELM can learn data arriving one-by-one or chunk-by-chunk for both

classification and regression tasks in non-stationary environments. The main mechanism of FP-ELM is to assign a forgetting parameter to old data when new data arrives during learning. Compared to other forgetting mechanism or forgetting factor based ELM algorithms, the forgetting parameters in FP-ELM are simply calculated according to the real-time performance of the learner, and the regularized optimization method is applied to avoid the windup phenomenon.

Empirical studies are carried out on four kinds of concept drift data sets, and we compare the performance of FP-ELM with two ensemble approaches, SEA and AddExp. All use ELMs as ensemble elements. Generally speaking, we can conclude that FP-ELM achieves a similar performance to AddExp(ELM) and outperforms SEA(ELM). What's more, as a single learner approach, FP-ELM realizes a faster training speed than do the ensemble approaches. The experimental results also show that the generalization performance of FP-ELM is not very sensitive to the regularization parameter.

Acknowledgements

This research is supported by the Natural Science Foundation of China under grants Nos. 61370144 and 61571180, the Natural Science Foundation of Hebei Province of China under grant No. G2014202031, the Key Project of the Educational Commission of Hebei Province under grant No. ZD2014009, the Youth Foundation of Education Commission of Hebei Province under grant No. QN2014192, and the Science and Technology Planning Project of Hebei Province of China under grant No. 15210325.

References

[1] P. Taweewat, C. Wutiwiwatchai, Musical pitch estimation using a supervised single hidden layer feed-forward neural network, Expert Systems with Applications 40 (2) (2013) 575-589.
[2] Q. Liu, C. Dang, T. Huang, A one-layer recurrent neural network for real-time portfolio optimization with probability criterion, IEEE Transactions on Cybernetics 43 (1) (2013) 14-23.
[3] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural Networks 4 (1991) 251-257.
[4] M. Leshno, V.Y. Lin, A. Pinkus, S. Schocken, Multilayer feedforward networks with a nonpolynomial activation function can approximate any function, Neural Networks 6 (1993) 861-867.
[5] J. Park, I. W. Sandberg, Universal approximation using radialbasis-function networks, Neural Comput. 3 (1991) 246-257.
[6] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (2006) 489-501.
[7] Y. Wang, F. Cao, and Y. Yuan, A study on effectiveness of extreme learning machine, Neurocomputing 74 (16) (2011) 2483-2490.
[8] L.-C. Shi and B.-L. Lu, EEG-based vigilance estimation using extreme learning machines, Neurocomputing 102 (2013) 135-143.
[9] J.W. Zhao, D.S. Park, J.W. Lee, F.L. Cao, Generalized extreme learning machine acting on a metric space, Soft Computing 16 (9) (2012) 1503-1514.
[10] W. Zong, G.-B. Huang, Y. Chen, Weighted extreme learning machine for imbalance learning, Neurocomputing 101 (2013) 229-242.
[11] G. Huang, S. Song, J. Gupta, C. Wu, Semi-supervised and unsupervised extreme learning machines, IEEE Transactions on Cybernetics 44 (12) (2014) 2405-2417.
[12] O. Fontenla-Romero, B. Guijarro-Berdinãs, B. Pérez-Sánchez, A. Alonso-Betanzos, A new convex objective function for the supervised learning of single-layer neural networks, Pattern Recognition 43 (2010) 1984-1992.
[13] H. Chen, Y. Gong, X. Hong, Online modeling with tunable RBF network, IEEE Transactions on Cybernetics 43 (3) (2013) 935-947.
[14] H. Jiang, Z. Ren, J. Xuan, X. Wu, Extracting Elite Pairwise Constraints for Clustering, Neurocomputing 99 (1) (2013) 124-133.
[15] L. Nie, H. Jiang, Z. Ren, Z. Sun, X. Li, QECK: Query Expansion Based on Crowd Knowledge for Code Search. IEEE Transactions on Services Computing (2016). doi: 10.1109/TSC.2016.2560165.
[16] N.Y. Liang, G.-B. Huang, P. Saratchandran, N. Sundararajan, A fast and accurate online sequential learning algorithm for feedforward networks, IEEE Transactions on Neural Networks 17 (6) (2006) 1411-1423.
[17] Y. Lan, Y.C. Soh, G.-B. Huang, Ensemble of online sequential extreme learning machine, Neurocomputing 72 (2009) 3391-3395.
[18] J. Xuan, H. Jiang, Y. Hu, Z. Ren, W. Zou, Z. Luo, X. Wu, Towards Effective Bug Triage with Software Data Reduction Techniques, IEEE Transactions on Knowledge and Data Engineering 27 (1) 2015 264-280.
[19] I. Žliobaitė, Learning under concept drift: an overview, 2010. ⟨http: //zliobaite.googlepages.com/Zliobaite_CDoverview.pdf⟩.
[20] W. Fan, Systematic data selection to mine concept-drifting data streams, in: Proceedings of the tenth International Conference on Knowledge Discovery and Data Mining, 2004, pp. 128-137.

[21] H. Yang, S. Fong, Countering the concept-drift problems in big data by an incrementally optimized stream mining model, Journal of Systems and Software 102 (2015) 158-166.
[22] S. Sen, Using instance-weighted naive Bayes for adapting concept drift in masquerade detection, International Journal of Information Security 13 (6) (2014) 583-590.
[23] D. Martínez-Rego, B. Pérez-Sánchez, O. Fontenla-Romero, A. Alonso-Betanzos, A robust incremental learning method for non-stationary environments, Neurocomputing 74 (2011) 1800-1808.
[24] W.N. Street, Y.S. Kim, A streaming ensemble algorithm (SEA) for large-scale classification, in: Proceedings of the seventh International Conference on Knowledge Discovery and Data Mining, 2001, pp. 377-382.
[25] D. Brzezinski, J. Stefanowski, Combining block-based and online methods in learning ensembles from concept drifting data streams, Information Sciences 265 (2014) 50-67.
[26] S.G. Soares, R. Araújo, A dynamic and on-line ensemble regression for changing environments, Expert Systems with Applications 42 (6) (2015) 2935-2948.
[27] S.G. Soares, R. Araújo, An on-line weighted ensemble of regressor models to handle concept drifts, Engineering Applications of Artificial Intelligence 37 (2015) 392-406.
[28] J.Z. Kolter, M.A. Maloof, Using additive expert ensembles to cope with concept drift, in: Proceedings of the 22nd International Conference on Machine Learning, 2005, pp. 449-456.
[29] R. Elwell, R. Polikar, Incremental learning of concept drift in nonstationary environments, IEEE Transactions on Neural Networks 22 (10) (2011) 1517-1531.
[30] D. Han, C. Giraud-Carrier, S. Li, Efficient mining of high-speed uncertain data streams, Applied Intelligence (2015).
[31] S.G. Soares, R. Araújo, An adaptive ensemble of on-line extreme learning machines with variable forgetting factor for dynamic system prediction, Neurocomputing 171 (2016) 693-707.
[32] L.L. Minku, A.P. White, X. Yao, The impact of diversity on online ensemble learning in the presence of concept drift, IEEE Transactions on Knowledge and Data Engineering 22 (5) (2010) 730-742.
[33] L.L. Minku, X. Yao, DDD: a new ensemble approach for dealing with concept drift, IEEE Transactions on Knowledge and Data Engineering 24 (4) (2012) 619-633.
[34] J. Gao, W. Fan, J. Han, P.S. Yu, A general framework for mining concept-drifting data streams with skewed distributions, in: Proceedings of the 2007 SIAM International Conference on Data Mining, 2007, pp. 3-14.
[35] G.-B. Huang, H. Zhou, X. Ding, R. Zhang, Extreme learning machine for regression and multi-class classification, IEEE Trans. Syst. Man Cybern. 42 (2) (2012) 513-529.
[36] J. Zhao, Z. Wang, D.S. Park, Online sequential extreme learning machine with forgetting mechanism, Neurocomputing 87 (2012) 79-89.
[37] R. Elwell, R. Polikar, Incremental learning in nonstationary environments with controlled forgetting, in: Proceedings of the International Joint Conference on Neural Networks, 2009, pp. 771-778.
[38] J.S. Lim, S. Lee, H.S. Pang, Low Complexity adaptive forgetting factor for online sequential extreme learning machine (OS-ELM) for application to nonstationary system estimations, Neural Computing and Applications 22 (3-4) (2013) 569-576.
[39] A. Frank, A. Asuncion, UCI Machine Learning Repository, School of Information and Computer Sciences, University of California, 2010. ⟨http: //archive.ics.uci.edu/ml⟩.
[40] G. Hulten, L. Spencer, P. Domingos, Mining time-changing data streams, in: Proceedings of the seventh International Conference on Knowledge Discovery and Data Mining, 2001, pp. 97-106.
[41] P. Gonçalves, S. Santos, R. Barros, D. Vieira, A comparative study on concept drift detectors, Expert Systems with Applications 41 (18) (2014) 8144-8156.
[42] M. Mike, Statistical Datasets, Department of Statistics, University of Carnegie Mellon, 1989. ⟨http://lib.stat.cmu.edu/datasets/⟩.

Dong Liu, born in 1990, got his master degree in computer science and technology at Hebei University of Technology in 2016 and is a Ph. D. candidate of software engineering at Dalian University of Technology. His research interests include machine learning and its applications in software engineering.

Youxi Wu, born in 1974, Ph. D., is a professor of computer science at Hebei University of Technology. His research interests include data mining and intelligent computation.

He Jiang, born in 1980, Ph.D., is a Ph. D. supervisor and a professor of software engineering at Dalian University of Technology. His research interests include intelligent computation and software engineering.