Kobe University Repository : Kernel

PDF issue: 2026-11-18

A fast online learning algorithm of radial basis function network with locality sensitive hashing

Ali, Siti Hajar Aminah

Fukase, Kiminori

Ozawa, Seiichi

(Citation)
Evolving Systems,7(3):173-186

(Issue Date)
2016-09

(Resource Type)
journal article

(Version)
Accepted Manuscript

(Rights)
©Springer-Verlag Berlin Heidelberg 2016. The final publication is available at Springer via http://dx.doi.org/10.1007/s12530-015-9141-5

(URL)
https://hdl.handle.net/20.500.14094/90003875

Evolving Systems manuscript No.
(will be inserted by the editor)

A Fast Online Learning Algorithm of Radial Basis Function Network with Locality Sensitive Hashing

Received: date / Accepted: date

Abstract In this paper, we propose a new incremental learning algorithm of Radial Basis Function (RBF) Network to accelerate the learning for large-scale data sequence. Along with the development of the internet and sensor technologies, a time series of large data chunk are continuously generated in our daily life. Thus it is usually difficult to learn all the data within a short period. A remedy for this is to select only essential data from a given data chunk and provide them to a classifier model to learn. In the proposed method, only data in untrained regions, which correspond to a region with a low output margin, are selected. The regions are formed by grouping the data based on their near neighbor using Locality Sensitive Hashing (LSH), in which LSH has been developed to search neighbors quickly in an approximated way. As the proposed method does not use all training data to calculate the output margins, the time of the data selection is expected to be shortened. In the incremental learning phase, in order to suppress catastrophic forgetting, we also exploit LSH to select neighbor RBF units quickly. In addition, we propose a method to update the hash table in LSH so that the data selection can be adaptive during the learning. From the performance of nine datasets, we confirm that the proposed method can learn large-scale data sequences fast without sacrificing the classification accuracies. This fact implies that the data selection and the incremental learning work effectively in the proposed method.

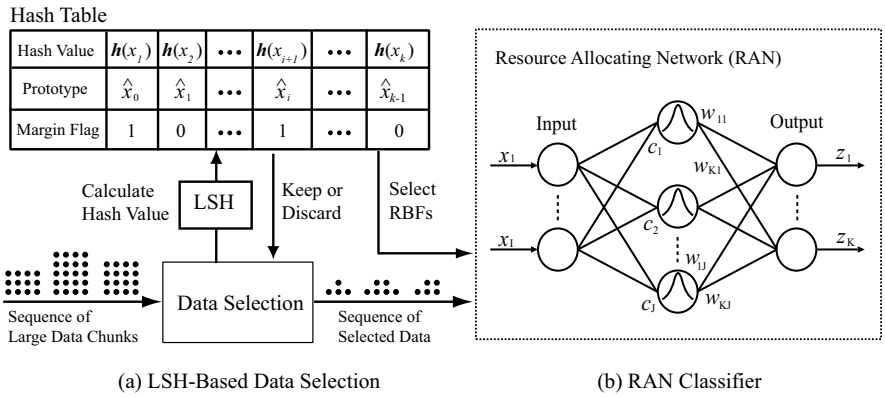
Keywords resource allocating network · locality sensitive hashing · incremental learning · large-scale data sequence

1 Introduction

Due to the recent development of computer networks and sensor technologies, many application software and devices (e.g. www, twitter, surveillance camera) keep generating large-scale and high-dimensional data continuously as a stream. Large-scale data sequences are characterized as data which are given continuously in a short time and whose data distribution is changed over time. There are at least two major issues to be solved: (1) how to handle the growing number of data while maintaining high-performance, and (2) how to adapt to dynamic environment quickly. The first issue can be solved by learning only important information incrementally (Attar et al. 2010). On the other hand, for the second issue, the computational time needs to be reduced by removing unimportant calculations in the learning.

For the incremental learning algorithm, we have proposed an extended version of modified Resource Allocating Network (RAN) classifier (Platt 1991) in which LSH is used to select data and active RBF centers. RAN with local learning (i.e. learning only active RBF centers) can effectively suppress the so-called catastrophic forgetting (Polikar et al. 2001) by learning some RBF centers with a given training data. However, the learning could be problematic if many data are provided to learn at the same time. In order to ensure real-time incremental learning, a mechanism to select and learn only essential data is solicited in modified RAN. For this purpose, a margin-based data selection has often been adopted in neural networks. In this approach, a training data is given to a neural classifier and the margins of the two largest outputs are calculated, and then only data with small margins are selected to learn. However, the computational costs to calculate output margins are generally expensive, especially when a large number of data are given as a continuous data sequence.

One way to alleviate the computational burden in the data selection is that the margin calculation is restricted only for training data in 'untrained' regions. To specify such regions with less computation, we adopt LSH (Datar et al. 2004, Andoni and Indyk 2008, Gu et al. 2013, Lee and Lee 2012, Shen et al. 2008). With LSH, the region of a given training data is quickly specified by transforming into a hash value and the learning status for the specified region is checked in a hash table. If the region is judged as 'untrained', the training data would be selected to learn; otherwise, it would be discarded. Since the calculation of hash values in LSH is generally much smaller than that of output margin, the data selection is expected to be fast by using LSH. The idea of using LSH can also be applied to the selection of active RBF units. Since the learning time strongly depends on the number of RBF units (Ozawa et al. 2005) in modified RAN, if only RBF units with high activations are selected, the learning of connection weights would be significantly faster. For this purpose, we adopt LSH again to find active RBF units quickly; that is, instead of calculating RBF activations, we can find active RBF units by calculating the LSH distances between the hash values of an input and all RBF centers.



(a) LSH-Based Data Selection (b) RAN Classifier

Fig. 1 Network Structure of RAN-LSH.

In this paper, we propose a fast learning algorithm of modified RAN which consists of the following two mechanisms: the LSH-based data selection and the LSH-based RBF selection. With the former mechanism, even if a large chunk of training data is given, only essential data are trained in modified RAN. With the latter mechanism, even if the number of RBF units becomes large, the learning time is expected to be fairly constant because the number of selected RBF units are limited to the near RBF. This paper is organized as follows. Section 2 explains the modified RAN and Section 3 presents its extended model, in which the data selection and the RBF selection are implemented by using LSH. In Section 4, the performance of the proposed model is evaluated for several benchmark datasets in the UCI Machine Learning Repository. Finally, Section 5 gives the conclusions of this work.

2 Modified Resource Allocating Network (RAN)

Figure 1(b) illustrates the structure of the RAN classifier. Let I, J, and K be the numbers of inputs, RBF outputs, and network outputs, respectively. When inputs $\boldsymbol{x} = \{x_1, \cdots, x_I\}^T$ are given to RAN, the RBF outputs $\boldsymbol{y}(\boldsymbol{x}) = \{y_1(\boldsymbol{x}), \cdots, y_J(\boldsymbol{x})\}^T$ and the network outputs $\boldsymbol{z}(\boldsymbol{x}) = \{z_1(\boldsymbol{x}), \cdots, z_K(\boldsymbol{x})\}^T$ are calculated as follows:

$$y_j(\boldsymbol{x}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{c}_j\|^2}{\sigma_j^2}\right) \quad (j = 1, \cdots, J) \tag{1}$$

$$z_k(\boldsymbol{x}) = \sum_{j=1}^{J} w_{kj} y_j(\boldsymbol{x}) + \xi_k \quad (k = 1, \cdots, K) \tag{2}$$

where $\boldsymbol{c}_j = \{c_{j1}, \cdots, c_{jI}\}^T$ and σ_j^2 are the center and the variance of the jth RBF unit, respectively; w_{kj} and ξ_k are the connection weight from the jth RBF unit to the kth output unit and its bias, respectively.

Algorithm 1 Modified RAN Learning

Input: Data $\boldsymbol{X}' = \{(\boldsymbol{x}_i, \boldsymbol{d}_i, \boldsymbol{h}(\boldsymbol{x}_i))\}_{i=1}^{N}$, RAN, hash table $\mathcal{H}$ and parameter θ_p.
Output: RAN.

```
1: for all (x_i, d_i, h(x_i)) ∈ X' do
2:    Calculate the outputs z for x_i and find the nearest center c* to x_i.
3:    Calculate the error: E = ||z(x) − 1_c||/K
4:    if E > ε & ||x − c*|| > δ then
5:       Add an RBF unit (i.e., J ← J + 1) and set an RBF center, and the connection
         weights: c_J = x_i, and w_J = d_i − z.
6:       Get an index set C of RBFs position in hash table as such x_i is the kth entry in
         hash table H_k: C_J = k.
7:    else
8:       Do LSH-based RBF Selection for training data (x_i, d_i, h(x_i)), an index set C of
         RBFs position in hash table and an index set R of selected RBFs.
9:       for all {(c_j, w_j)}_{j∈R} do
10:         Calculate the outputs Φ for selected RBF centers c_j (j ∈ R) and a training
            data x_i by Eq. (1).
11:         Decompose Φ using SVD and obtain the matrices V, H, U.
12:         Update connection weights W' = V H^{-1} U^T D.
13:      end for
14:      Calculate the outputs z for x_i.
15:      Calculate the error: E = ||z(x) − 1_c||/K
16:      if E > ε then
17:         Add a hidden unit and the connection weights: c_J = x_i, and w_J = d_i − z.
18:         Get an index set C: C_J = k.
19:      end if
20:   end if
21: end for
```

Algorithm 1 shows the learning algorithm of a modified RAN. The learning of RAN is divided into two phases: the addition of RBF units (Line 4-5 and Line 16-17) and the update of connection weights between hidden and output units (Line 9-13). Once RBF units are allocated, the centers are fixed afterwards. The connection weights $\boldsymbol{W} = \{\{w_{jk}\}_{j=1}^{J}\}_{k=1}^{K}$ of the modified RAN are updated by solving the following equation (Haykin 1999):

$$\boldsymbol{\Phi W} = \boldsymbol{D} \tag{3}$$

where $\boldsymbol{\Phi}$ is a matrix of RBF outputs and $\boldsymbol{D}$ is a target matrix. Suppose that a training data $(\boldsymbol{x}, \boldsymbol{d})$ is given and J RBF centers $(\boldsymbol{c}_j, \boldsymbol{w}_j)$ $(j = 1, \cdots, J)$ have already been allocated. In order to suppress the interference caused by the incremental learning, the modified RAN is trained with some RBF centers as well as a new data. Therefore, a target matrix $\boldsymbol{D}$ in Eq. (3) is formed as follows: $\boldsymbol{D} = \{\boldsymbol{d}, \boldsymbol{w}_1, \cdots, \boldsymbol{w}_J\}^T$, while a matrix of RBF outputs $\boldsymbol{\Phi}$ is given by $\boldsymbol{\Phi} = \{\boldsymbol{y}(\boldsymbol{x}), \boldsymbol{y}(\boldsymbol{c}_1), \cdots, \boldsymbol{y}(\boldsymbol{c}_J)\}$. To obtain $\boldsymbol{W}$ in Eq. (3), Singular Value Decomposition (SVD) is often used, and the solution is given by $\boldsymbol{W} = \boldsymbol{V}\boldsymbol{H}^{-1}\boldsymbol{U}^T\boldsymbol{D}$ where $\boldsymbol{V}$ and $\boldsymbol{U}$ are orthogonal matrices and $\boldsymbol{H}$ is a diagonal matrix.

Since the computational complexity of SVD is $O(J^3)$, the number of RBFs is a key to reduce the learning time. Let us introduce the selection of active RBF in the modified RAN. Algorithm 1 shows a pseudo-code of the RAN learning with RBF selection. As seen at Line 8 in Algorithm 1, when the

Algorithm 2 RAN-LSH Learning Algorithm

Input: A set of N_0 initial training data $\boldsymbol{X}_0 = \{(\boldsymbol{x}_i, \boldsymbol{d}_i)\}_{r=1}^{N_0}$, parameters θ_a and θ_m.
1: Do Initialization. Get proper parameters σ^2, P and an initial hash table $\mathcal{H}$.
2: loop
3: Input: a chunk of N training data $\boldsymbol{X} = \{(\boldsymbol{x}_i, \boldsymbol{d}_i)\}_{i=1}^{N}$ and parameter θ_d.
4: Calculate the hash values $\boldsymbol{h}(\boldsymbol{x}_i)$ for all $\boldsymbol{x}_i \in \boldsymbol{X}$ by Eqs. (5) and (6).
5: Set $\hat{\boldsymbol{X}} = \{(\boldsymbol{x}_i, \boldsymbol{d}_i, \boldsymbol{h}(\boldsymbol{x}_i))\}_{i=1}^{N}$.
6: Do Update Hash Table and get a new hash table $\mathcal{H}$.
7: Do LSH-Based Data Selection and get a set of selected data $\boldsymbol{X}'$.
8: Do Modified RAN Learning for $\boldsymbol{X}'$.
9: end loop

connection weights are updated, only a part of RBFs are selected and used to define the matrix of RBF outputs $\boldsymbol{\Phi}'$. The matrix $\boldsymbol{\Phi}'$ is decomposed into the orthogonal matrices $\boldsymbol{V}'$, $\boldsymbol{U}'$ and a diagonal matrix $\boldsymbol{H}'$ by using SVD. Then, the connections weights $\boldsymbol{W}'$ are updated for the selected RBF units, while the connections for the other RBF units are not changed.

In order to suppress the catastrophic interference, it is empirically known that some RBF units should be learned along with a given training data, so that the previous memories are not forgotten when learning new data. In conventional RAN, the problem is solved by learning local region which is approximated using C^1 continuous polynomial without calculating the RBF outputs (Platt 1991). Whereas, the same goal is achieved by another variant of RAN, RAN with Long Term Memory (RAN-LTM) where it carried out the learning with memory items that are stored in LTM (Kobayashi et al. 2001, Okamoto et al. 2003, Ozawa et al. 2010). Nevertheless, we use different approach which can be easily implemented by using LSH where we are able to select active RBF units. This will be described in the later section.

3 RAN with LSH-based Data Selection and LSH-based RBF Selection

3.1 Learning Algorithm

We assume that only a limited number of training data are available in the initial learning phase. These initial training data are used to decide the structure of a learning model and to set up its appropriate parameters. On the other hand, in the incremental learning phase, it is assumed that a large set of training data are given sequentially and it is too large to learn all data at every time step.

Under such learning environments, training data to be learned in the incremental learning phase should be restricted to only essential ones such as those existing in untrained regions. For this purpose, the margin-based data selection has often been used (Dan and Kevin 2006). However, to introduce this margin-based method in modified RAN, the output margins must be calculated for all training data. Obviously, it requires expensive computations for

Algorithm 3 Initialization

Input: Initial data set $\boldsymbol{X}_0 = \{(\boldsymbol{x}_i, \boldsymbol{d}_i)\}_{i=1}^{N_0}$, parameters θ_a and θ_m.
Output: Hash table $\mathcal{H}$, parameters σ^2 and P.

1: Do the cross validation for $\boldsymbol{X}_0$ to determine the parameters: the RBF width σ^2 and the number of partitions P.
2: Do PCA for $\boldsymbol{X}_0$, and obtain l eigenvectors $\boldsymbol{U}_l$ such that the accumulation ratio is larger than θ_a.
3: for j=1 to l do
4: Calculate the jth projection values v_{ij} for all $(\boldsymbol{x}_i, \boldsymbol{d}_i) \in \boldsymbol{X}_0$ by Eq. (4).
5: Get the maximum value $v_j^{\max}$ and the minimum value $v_j^{\min}$.
6: Calculate the decimal hash function $H(v_{ij})$ for v_{ij} by Eq. (5).
7: end for
8: Get the hash value $\boldsymbol{h}(\boldsymbol{x}_i)$ for all $\boldsymbol{x}_i \in \boldsymbol{X}_0$ by Eq. (6).
9: Sort the hash values in the ascending order: $\{\boldsymbol{h}_1, \cdots, \boldsymbol{h}_k. \cdots\}$.
10: for all hash values $\boldsymbol{h}_k$ do
11: Find all $\boldsymbol{x}_i$ with $\boldsymbol{h}(\boldsymbol{x}_i) = \boldsymbol{h}_k$ and calculate the mean vector $\hat{\boldsymbol{x}}_k$.
12: Calculate the output margin for $\hat{\boldsymbol{x}}_k$ by Eq. (8) and set the margin flag F_k by Eq. (9).
13: Register $\{\boldsymbol{h}_k, \hat{\boldsymbol{x}}_k, F_k\}$ in the hash table $\mathcal{H}$.
14: end for

the data selection, and it makes difficult for modified RAN to learn a large data chunk in real time.

To select untrained data from a chunk of given data quickly, we introduce LSH in the data selection. Figure 1 illustrates the proposed model called RAN with LSH-based Data Selection (RAN-LSH) and Algorithm 2 shows the main flows of the RAN-LSH learning. As seen in Fig. 1, RAN-LSH is composed of two parts: (a) LSH-based data selection and (b) RAN classifier. The learning is divided into the following two phases: initial learning phase and incremental learning phase. As mentioned above, in the initial learning phase, only a limited number of data are assumed to be given. They are used to get proper values of the RBF width σ^2 and the number of partitions P, and to create an initial hash table $\mathcal{H}$ (see Step 1 in both Algorithms 2 and 3).

After the initialization, the learning mode is changed to incremental learning. Whenever a new chunk of data is given, it is first forwarded to LSH to calculate a hash value. After updating the hash table, the chunk of data is sent to the data selection part to select only essential data from the given chunk $\boldsymbol{X}$. In order to make the learning of RAN-LSH faster, we adopt a local learning algorithm in which only a limited number of active RBF units are selected by using LSH and their connection weights are trained incrementally (see Fig. 1). Since the time complexity of learning connection weights in modified RAN is the order $O(J^3)$ (J: the number of RBF units) (Ozawa et al. 2005), it is expected that the proposed local learning algorithm would also contribute to reducing the learning time in RAN-LSH. These steps are summarized in Algorithm 2 (RAN-LSH learning algorithm).

In the following, we explain the detailed procedures in the LSH-based data selection and the whole learning algorithm of RAN-LSH.

3.2 LSH-based Data Selection

The LSH-based data selection is carried out based on a hash table $\mathcal{H}$ which consists of the following three items: hash value $\boldsymbol{h}(\boldsymbol{x})$, prototype vector $\hat{\boldsymbol{x}}$ and margin flag F. A hash function is used to encode an input data into a hash value as an index in the hash table. In LSH, input data that exist within a local region are encoded into the same hash value (Chellapilla et al. 2007). A prototype vector corresponds to a typical input data (e.g., mean vector) which represents a local region, and the margin flag means the learning status of the local region: trained or untrained.

Since the proposed LSH-based data selection is carried out under an incremental learning environment, the hash table $\mathcal{H}$ in the proposed method should be updated on an online basis. For this purpose, the LSH method used here has the following differences from the conventional projection-based implementations of LSH proposed by Andoni et. al. (Andoni and Indyk 2008):

1) A hash function H_i is defined as the projection to an eigenvector obtained by Principal Component Analysis (PCA), instead of that to a random vector.
2) Only one hash table $\mathcal{H}$ is used and $\mathcal{H}$ is evolved in both size and contents to adapt to time-varying data distributions; that is, the entries in $\mathcal{H}$ is increased in number and updated online. Therefore, it is unlikely that $\mathcal{H}$ covers all areas of an input space, although it may end up covering the whole space for a long run.

In the next subsections, let us give detailed explanations on the hash encoding, how to update a hash table and the propose data selection algorithm.

3.2.1 Hash Encoding

The basic idea of LSH is to hash high-dimensional input data so that similar data are mapped to the same buckets (subregions of inputs) with high probability. Here, we adopt a projection-based LSH, in which the projection in a hash encoding is conducted for eigenvectors obtained by PCA, instead of random vectors. It is well known that an eigen-subspace spanned by such eigenvectors gives a good approximation to an input space and its approximation error can be easily controlled by the accumulation ratio, which is defined by the ratio of input components in the approximated subspace over those in the whole input space. Therefore, by introducing eigenvectors instead of random vectors in the projection-based LSH, we can easily determine a suitable number of hash functions from a given data distribution.

Let l be the number of eigenvectors which are obtained by setting the threshold θ_a for the accumulation ratio; that is, l is given as the minimum number of eigenvectors whose accumulation ratio is larger than or equal to the threshold θ_a. In addition, let the obtained eigenvectors be $\boldsymbol{U}_l = \{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_l\}$. Then, a projection vector $\boldsymbol{V} = \{v_1, \cdots, v_l\}^T$ of an input data $\boldsymbol{x}$ is obtained by

$$\boldsymbol{V} = \boldsymbol{U}_l^T \boldsymbol{x}. \tag{4}$$

Algorithm 4 Update Hash Table

Input: RAN, $\hat{\boldsymbol{X}} = \{(\boldsymbol{x}_i, \boldsymbol{d}_i, \boldsymbol{h}(\boldsymbol{x}_i))\}_{i=1}^{N}$, and hash table $\mathcal{H}$.
Output: Hash table $\mathcal{H}$.

```
for all x_i ∈ X̂ do
    Find h(x_i) in H.
    if h(x_i) and h(x̂_k) of kth entry H_k = {h(x̂_k), x̂_k, F_k} is matched then
        Update x̂_k with x_i by Eq. (7).
        if F_k = 0 then
            Calculate the output margin for x̂_k by Eq. (8).
            Update margin flag F_k by Eq. (9).
        end if
    else
        Calculate the output margin for x_i by Eq. (8).
        Calculate margin flag F_{K+1} by Eq. (9).
        Register a (K+1)th entry in H: H_{K+1} = {h(x_i), x_i, F_{K+1}}.
    end if
end for
```

In the proposed hash encoding, each projection v_i $(i = 1, \cdots, l)$ is divided into P partitions along the one-dimensional space spanned by the eigenvector $\boldsymbol{u}_i$. Let $H(v_i)$ be a hash function that transforms v_i into an l-series of hash value: $H(v_i) \in \{1, \ldots, P\}_{i=1}^{l}$, and let v_i^+ and v_i^- be the lower and upper values of typical projections v_i on the ith eigenvector $\boldsymbol{u}_i$, respectively. In the proposed RAN-LSH, v_i^+ and v_i^- are determined in the initial learning phase (see Algorithm 3 Line 5), and they are defined as the minimum and maximum values of the projections for initial training data $\boldsymbol{X}_0$. The hash function $H(v_i)$ is given as follow:

$$H(v_i) = \max\{\left\lceil \frac{\min\{\max\{v_i, v_i^-\}, v_i^+\} - v_i^-}{v_i^+ - v_i^-} P \right\rceil, 1\}. \tag{5}$$

Therefore, a hash value $\boldsymbol{h}(\boldsymbol{x})$ is defined as the following decimal code, which is given by concatenating all the decimal codes of a hash function $H(v_i)$:

$$\boldsymbol{h}(\boldsymbol{x}) = \{H(v_1), \cdots, H(v_l)\} \tag{6}$$

as such, an input $\boldsymbol{x}$ is encoded into an l-series of decimal hash vector.

3.2.2 Update of Hash Table

As seen in Algorithm 2, the hash values of training data $\boldsymbol{X}$ are first calculated, and $\boldsymbol{X}$ is redefined by adding the hash values $\boldsymbol{h}(\boldsymbol{x}_i)$: $\hat{\boldsymbol{X}} = \{(\boldsymbol{x}_i, \boldsymbol{d}_i, \boldsymbol{h}(\boldsymbol{x}_i))\}_{i=1}^{N}$. Then, a hash table $\mathcal{H}$ is updated with the newly defined training data $\hat{\boldsymbol{X}}$. Algorithm 4 shows how to update a hash table $\mathcal{H}$ with $\hat{\boldsymbol{X}}$. In the following, let us assume that a hash table has K entries (i.e., $\mathcal{H} = \{\mathcal{H}_k\}_{k=1}^{K}$) and the kth entry $\mathcal{H}_k$ is represented by a triplet $\mathcal{H}_k = \{\boldsymbol{h}(\hat{\boldsymbol{x}}_k), \hat{\boldsymbol{x}}_k, F_k\}$ where $\hat{\boldsymbol{x}}_k$, $\boldsymbol{h}(\hat{\boldsymbol{x}}_k)$, and F_k are a prototype vector, its decimal hash encoding, and the margin flag, respectively.

For each training data in $\hat{\boldsymbol{X}}$, the following procedures are carried out. First, a hash value $\boldsymbol{h}(\boldsymbol{x}_i)$ is searched from $\mathcal{H}$. If $\boldsymbol{h}(\boldsymbol{x}_i)$ is matched with $\boldsymbol{h}(\hat{\boldsymbol{x}}_k)$ in $\mathcal{H}$,

Algorithm 5 LSH-Based Data Selection

Input: $\hat{\boldsymbol{X}} = \{(\boldsymbol{x}_i, \boldsymbol{d}_i, \boldsymbol{h}(\boldsymbol{x}_i))\}_{i=1}^{N}$ and hash table $\mathcal{H}$.
Output: $\hat{\boldsymbol{X}} = \{(\boldsymbol{x}_i, \boldsymbol{d}_i, \boldsymbol{h}(\boldsymbol{x}_i))\}_{i=1}^{N}$.

```
for all x_i ∈ X̂ do
    Find h(x_i) in H.
    if F_k = 1 for a matched entry then
        Remove x_i from X̂ and N ← N − 1.
    end if
end for
```

it implies that the neighbor area around $\boldsymbol{x}_i$ has already been learned. In this case, the prototype vector $\hat{\boldsymbol{x}}_k$ is updated with $\boldsymbol{x}_i$ as follow:

$$\hat{\boldsymbol{x}}_k' = \frac{1}{N+1}\left(N\hat{\boldsymbol{x}}_k + \boldsymbol{x}_i\right). \tag{7}$$

If the margin flag $F_k = 0$, the output margin should be checked for the updated prototype $\hat{\boldsymbol{x}}_k'$ as follow:

$$\Delta z(\hat{\boldsymbol{x}}_k') = z^*(\hat{\boldsymbol{x}}_k') - z^{**}(\hat{\boldsymbol{x}}_k') \tag{8}$$

where z^* and z^{**} are the largest and the second largest outputs of modified RAN, respectively. Then, the margin flag F_k is updated as follow:

$$F_k = \begin{cases} 1 & (\Delta z(\hat{\boldsymbol{x}}_k') > \theta_m) \\ 0 & (\text{otherwise}) \end{cases} \tag{9}$$

where θ_m is a positive threshold.

If there is no matched entry in $\mathcal{H}$, it implies that a training data $\boldsymbol{x}_i$ has not been learned so far. Thus, the output margin for $\boldsymbol{x}_i$ need to be calculated, so that a margin flag would be assigned for the new entry. Then, a new entry $\mathcal{H}_{K+1}$ is created in $\mathcal{H}$ as follow: $\mathcal{H}_{K+1} = \{\boldsymbol{h}(\boldsymbol{x}_i), \boldsymbol{x}_i, F_{K+1}\}$.

3.2.3 Data Selection

Algorithm 5 shows the steps to select essential data. In the data selection, the hash value for each data is examined (see also the upper left of Fig. 1(a)). If the margin flag is 0, it implies that a training data exists in an untrained region; thus, such a training data should be selected and learned by RAN. On the other hand, if the margin flag is 1, it implies that the neighbor region of such a training data is well trained. Therefore, such a data is not considered to be essential for RAN, and it can be discarded to reduce the learning time.

3.3 LSH-based RBF Selection

As mentioned before in Section 2, RBF bases in hidden layer are used to suppress the interference effectively. However, it is not efficient to retrieve and train all RBF bases. One of the reason is the calculation of the SVD

Algorithm 6 LSH-based RBF Selection

Input: Training data $(\boldsymbol{x}_i, \boldsymbol{d}_i, \boldsymbol{h}(\boldsymbol{x}_i))$, an index set $\mathcal{C}$ of RBFs position in hash table, an index set $\mathcal{R}$ of selected RBFs, hash table $\mathcal{H}$, and parameter θ_p.
Output: $\mathcal{R}$
1: for $j = 1$ to J do
2: Get hash value of kth entry from hash table $\mathcal{H}_{\{k=\mathcal{C}_j\}}$: $\boldsymbol{h}(\boldsymbol{c}_j) = \boldsymbol{h}(\hat{\boldsymbol{x}}_k)$.
3: Calculate LSH distance d_j^* between hash values of both RBF centers $\boldsymbol{h}(\boldsymbol{c}_j)$ and a given training data $\boldsymbol{h}(\boldsymbol{x}_i)$ using Eq. (10).
4: if $d_j^* \leq \theta_p$ then
5: Define a set $\mathcal{R}$ of selected RBF centers (i.e $\mathcal{R} \leftarrow j$).
6: end if
7: end for

in $\boldsymbol{W} = \boldsymbol{V}\boldsymbol{H}^{-1}\boldsymbol{U}^T\boldsymbol{D}$ would become heavier for a larger number of hidden units as such, if all RBF centers are used to update the weight. Thus, the processing time would require a longer time. Therefore, we restrict the RBF bases to learn only near RBF centers, in which must satisfy a specific condition (i.e. LSH distance $d^* \leq \theta_p$). The algorithm to retrieve only a part of RBF bases is shown in Algorithm 6.

In this study, we adopt LSH distance with decimal hash code, instead of the well-known Hamming distance of binary hash code. This is because, we notice that the Hamming distance of binary values is only appropriate to measure distance of projection vector $\boldsymbol{V}$ which is divided into at most three partitions. If larger number of partitions is used, the Hamming distance value would not represents the distance correctly. For example, in the case of $P = 8$, 3-bits binary would be used to represent each segment/bucket where v_l is located (i.e. segment $\in \{1, \ldots, P\}$). Let say segment = 5 and segment = 6 are represented by binary code 011_b and 100_b respectively. When the Hamming distance between these segments are calculated, it would give 3-bit difference instead of 1-bit difference. Since in this study we use larger than three partitions, it is inappropriate to use binary hash code and Hamming distance to find near RBFs.

To find the RBFs neighbor of a training data using LSH, firstly, the hash values of RBFs need to be retrieved from the constructed hash table $\mathcal{H}$ where the $\mathcal{C}_j$th entries in $\mathcal{H}$ are correspond to the hash values of the jth RBFs. Next, the LSH distance d_j^* for each jth RBFs can be calculated as follows;

$$d_j^* = \sum_{i=1}^{l} (\boldsymbol{h}_i(\boldsymbol{x}) - \boldsymbol{h}_i(\boldsymbol{c}_j)) \tag{10}$$

where we accumulate the distance (using hash values in decimal) between a given training data $\boldsymbol{h}_i(\boldsymbol{x})$ and all RBF centers $\boldsymbol{h}_i(\boldsymbol{c})$ for l-projection vector in $\boldsymbol{V}$. Then the RBF centers with LSH distances that are less or equal to θ_p would be selected to solve the linear equation in Eq. (3).

For instance, let partition P is equal to 16 and the number of eigenvectors i given by $\theta_a = 0.7$ is equal to 4. Thus the projection vector is $\{v_i\}_{i=1}^4 \in \{1, \cdots, 16\}$. Let say, the hash value of a training data $\boldsymbol{h}(\boldsymbol{x})$ is $\{6, 3, 1, 10\}$. While

Table 1 Evaluated UCI Datasets (Bache and Lichman 2013)

Dataset	#Init. Data	#Train. Data	#Test Data	#Attrib.	#Classes
Hiragana	500	8366	8365	13	38
Adult	500	22611	22611	14	2
Shuttle	500	28989	28988	9	5
Bank	500	22606	22605	16	2
Pendigits	500	5496	5496	16	10
Letter	500	10000	10000	16	26
MAGIC	500	9510	9510	10	2
Optical Digits	281	2810	2810	64	10
Semeion	80	797	796	256	10

the hash values of two different RBF centers $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ are $\boldsymbol{h}(\boldsymbol{c}_1) = \{5, 12, 7, 1\}$ and $\boldsymbol{h}(\boldsymbol{c}_2) = \{6, 2, 1, 9\}$, respectively. Therefore, the LSH distance d_j^* between a training data $\boldsymbol{x}$ and RBF centers $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ are $d_1^* = |6-5| + |3-12| + |1-7| + |10-1| = 25$ and $d_2^* = |6-6| + |3-2| + |1-1| + |10-9| = 2$, respectively. Hence for $\theta_p = 3$, RBF center $\boldsymbol{c}_2$ would be selected. This strategy to select RBF bases is expected to be fast because the hash functions are not recalculated but instead are recalled from the hash table which had been constructed during data selection. Besides that, it is assumed that more learning time would be reduced especially for large dimensions dataset, since the number of projection vectors are much smaller than the number of feature vectors.

4 Experimental Results and Discussions

4.1 Experimental Setup

To evaluate the performance of the proposed method, we use the nine benchmark datasets in the University of California at Irvine (UCI) Machine Learning Repository (Bache and Lichman 2013). The dataset information is shown in Table 1. Although training and test data are separately provided in some data sets, they are merged and randomly divided into two subsets to evaluate the average performance through the two-fold cross-validation. Since the performance generally depends on the sequence of training data, 30 sequences in total are trained incrementally and the average performance is evaluated for the test data. For the first seven datasets in Table 1, 500 initial training data are randomly selected from the training data. For the other datasets (Optical Digits and Semeion), 10% of the training data are randomly selected as initial data. After the initial learning, it is assumed that a large chunk of data is given sequentially in the incremental learning phase. The number of data in a chunk is set to 1,000 at every learning stage except for the last stage. Since training data are assumed to be given only once, the data chunk given at the last stage may consist of less than 1,000 data.

In the following experiments, we evaluate the proposed RAN-LSH from the following three points: (1) appropriateness of the granularity in the hash encoding, (2) effectiveness of the data selection, and (3) the effectiveness of

the local learning in modified RAN. For the notational convenience, let us denote modified RAN with the margin-based data selection as RAN-MRG. The experiments are carried out on a personal computer with Intel Core Quad (3.4 GHz) CPU and 32 GB main memory. We estimate the computational costs based on the actual learning time measured by the Matlab stopwatch timer function.

In RAN-LSH, the three parameters on accumulation ratio θ_a, output margin θ_m, and tolerant distance θ_p should be determined empirically. In this experiment, we use the following parameter settings: $\theta_a = 0.7$, $\theta_m = 0.05$, and $\theta_p = 3$. Meanwhile, the error threshold ε is set to 0.5. The other parameters, RBF width σ and the number of partitions P, are determined in the initial learning phase through the cross-validation.

4.2 Granularity in Hash Encoding

To avoid unnecessary calculations of output margins in the data selection, similar data are associated with the same partition to represent the learning status (i.e., well-trained or untrained). As easily expected, if the number of partitions is small, a state space is roughly represented in the hash encoding and the collision probability of dissimilar data tends to be high. On the contrary, if the number of partitions is large, the collision probability would be low (i.e., data are sparsely allocated to each segment/bucket of P partition), while the size of a hash table would be large, resulting in the increase of the memory and computational costs.

To find a suitable granularity in the hash encoding (i.e., a proper number of partitions P), let us introduce the following Davies-Bouldin index I_{DB} (Davies and Bouldin 1979):

$$I_{DB} = \frac{1}{P}\sum_{i=1}^{P}\max_{j:i\neq j}\frac{S_i + S_j}{\|\bar{\boldsymbol{x}}_i - \bar{\boldsymbol{x}}_j\|} \tag{11}$$

where $\bar{\boldsymbol{x}}_i$ and S_i are the centroid and the variance of data allocated to the ith-segment of P partition, respectively. The numerator in Eq. (11) corresponds to the within-cluster scatter for the segments i and j, which should be as small as possible. The denominator corresponds to the between-cluster separation between the centroids $\bar{\boldsymbol{x}}_i$ and $\bar{\boldsymbol{x}}_j$. This Davies-Bouldin index I_{DB} depends on the data as well as the hash encoding method, and ideally has to be as small as possible. However, as mentioned above, too small I_{DB} leads to the sparse data allocation in the hash encoding and results in the increase in computation and memory costs. Therefore, we need to find a proper value of P such that a moderate value of I_{DB} with less memory and computational costs can be obtained.

Figure 2 illustrates I_{DB} and the memory size for different numbers of partitions P. We use Shuttle, Pendigits and MAGIC Gamma Telescope (MAGIC) datasets for the evaluation. As seen in Fig. 2, when P increases, I_{DB} decreases;

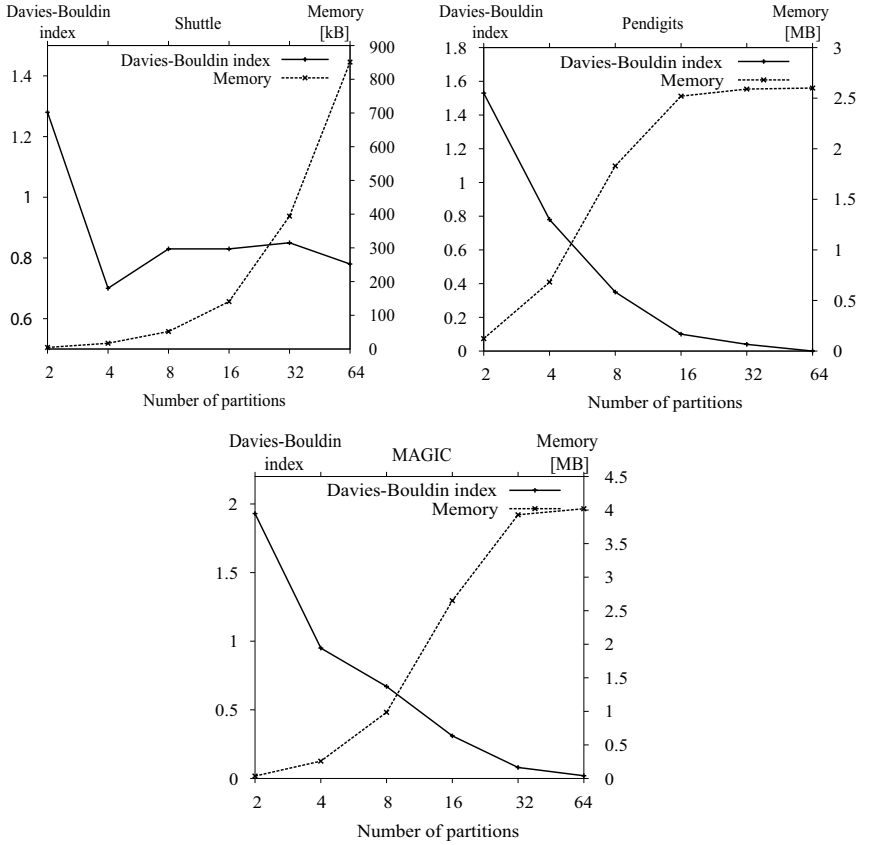


Fig. 2 Effect of the number of partitions P to the Davies-Bouldin index I_{DB} and the memory consumption for (a) Shuttle, (b) Pendigits, and (c) MAGIC datasets.

Table 2 Effect of the number of partitions P to learning time [sec.]

Methods	Shuttle	Pendigits	MAGIC
RAN-MRG	1.6±0.4	1.2±0.1	13.8±1.7
RAN-LSH ($P = 2$)	0.6±0	0.3±0.1	0.3±0.2
RAN-LSH ($P = 4$)	1.0±0.1	0.9±0.1	4.4±0.8
RAN-LSH ($P = 8$)	1.0±0.1	1.4±0.1	10.0±1.6
RAN-LSH ($P = 16$)	1.0±0.1	1.6±0.1	14.4±1.9
RAN-LSH ($P = 32$)	1.3±0.2	1.6±0.1	14.7±1.8
RAN-LSH ($P = 64$)	3.3±0.5	2.2±0.1	16.5±1.7

that is, the collision probability of dissimilar data is getting low. In contrast, the required memory size is increased due to the increase in the number of hash entries. Meanwhile, Table 2 shows the learning time for different numbers of partitions. Obviously, when the number of partitions increases, the time for incremental learning becomes long.

Table 3 Effectiveness of introducing the data selection. The performance is compared among three models; modified RAN, RAN-MRG and RAN-LSH. Note that modified RAN has no data selection mechanism. In the incremental learning phase, a chunk of 1,000 data is given at every learning stage. The performance measures are (a) the final recognition accuracy [%], (b) learning time [sec.], and (c) the total number of selected data, which are averaged over 30 trials with different data sequences. The results are represented in the form of (average) ± (standard deviation). The single asterisk (*) and the double asterisk (**) mean that the average difference against modified RAN (no data selection) is statistically significant under p-value is less than 5% and 1% level, respectively.

(a) Recognition accuracy [%]

Dataset	RAN	RAN-MRG	RAN-LSH
Shuttle	98.4±1.8	98.9 ± 1.1	98.6 ± 0.9
Pendigits	98.9±0.2	$98.3 \pm 0.3^{**}$	$98.3 \pm 0.3^{**}$
MAGIC	76.5±0.8	$74.6 \pm 2.1^{**}$	$75.1 \pm 2.1^{*}$

(b) Learning time [sec.]

Dataset	RAN	RAN-MRG	RAN-LSH
Shuttle	151.2±20.7	$1.6 \pm 0.4^{**}$	$1.3 \pm 0.5^{**}$
Pendigits	134.7±9.8	$1.9 \pm 0.1^{**}$	$1.7 \pm 0.1^{**}$
MAGIC	10,808±809	$18.8 \pm 1.7^{**}$	$16.2 \pm 2.2^{**}$

(c) Total number of selected data

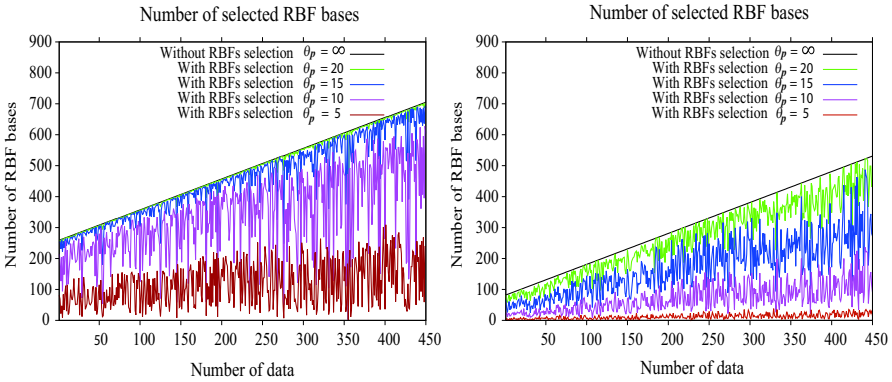
Dataset	RAN	RAN-MRG	RAN-LSH
Shuttle	28,488	$216 \pm 101^{**}$	$95 \pm 28^{**}$
Pendigits	4,996	$67 \pm 8^{**}$	$57 \pm 10^{**}$
MAGIC	9,010	$380 \pm 27^{**}$	$249 \pm 30^{**}$

From the results in Fig. 2 and Table 2, one can say that an optimal value of P exists between 4 and 16. Therefore, let us find a suitable number of partitions P within this range by applying the cross-validation to an initial dataset.

4.3 Effectiveness of Data Selection

To see the effectiveness of the data selection in RAN-LSH, we compare the performance with the other two modified RAN models, in which no data selection mechanism is introduced (RAN model) and a margin-based data selection is introduced (RAN-MRG model). Table 3 shows the results of the performance comparison based on the following measures: (a) the final recognition accuracy [%], (b) learning time [sec.], and (c) the total number of selected data. The final recognition accuracy is examined for the test dataset after the incremental learning phase is completed, and the learning time only includes the computational time for incremental learning.

As seen in Table 3(a), the recognition accuracies of RAN-MRG and RAN-LSH are slightly dropped compared with that of modified RAN because many training data are discarded in the data selection (see Table 3(c)). For Shuttle, only 0.3% of training data are used for learning in RAN-LSH on average. Thus, 99.7% training data are discarded by the proposed LSH-based data selection, although the performance seems to be relatively decreased by 0.1%. On the other hand, although small performance drops are recognized for Pendigits



(a) Optical Digits (b) Semeion

Fig. 3 Effect of RBF bases selection with different θ_p. Best viewed in color.

and MAGIC, the percentages of the data reduction are 98.9% and 97.2%, respectively. Obviously, this data reduction results in significant speed-up in learning (see Table 3(b)). Compared with RAN-MRG, the proposed RAN-LSH can reduce more training data without deteriorating the recognition accuracy. Therefore, we can conclude that the proposed LSH-based data selection works properly.

4.4 The Effect of RBF Bases Selection

To investigate the efficiency of RAN-LSH with RBF selection, we carried out two observations as follows: (1) identify the effect of selecting RBF bases using different values of θ_p, and (2) examine the performance regarding (a) the recognition accuracy and (b) learning time. Figure 3 and Table 4 show the above findings respectively.

Figure 3 illustrates the effects of using different values of θ_p towards number of RBFs selected for two datasets (a) Optical Digits and (b) Semeion. In Fig. 3 (without RBFs selection), it can be seen that the number of RBF bases increases gradually in proportion to the number of training data. As previously stated in Algorithm 1, the RBF center would be added if the current model is not well-trained and thus RAN-LSH model without RBFs selection is not suitable for large-scale data sequences. This is because the learning time would take an eternity to update the weights of a very large RBF outputs matrix $\boldsymbol{\Phi}$. As a result, the model could not catch up with the speed of the next data chunk. However, we can reduce the influence of expansion on hidden units' size by learning only RBF bases which are located near to the training data.

Learning a training sample along with near RBF bases could prevent catastrophic forgetting in the incremental learning. However, learning all RBF bases would cause a new interference due to the distribution of the RBF bases that

Table 4 The effect of selecting RBF centers using different threshold values θ_p namely through; (a) recognition accuracy [%], (b) incremental learning time [s], and (c) the total number of RBF bases selected. The values represented are the mean and standard deviation of each method.

(a) Recognition accuracy [%]

Dataset	RAN-MRG	RAN-LSH $\theta_p = \infty$	RAN-LSH $\theta_p = 10$	RAN-LSH $\theta_p = 3$
Hiragana	96.8±0.4	96.6±0.4	96.8±0.4	96.7±0.4
Adult	78.1±1.2	78.0±1.2	78.1±1.0	77.8±1.3
Shuttle	98.9±1.1	98.6±0.9	98.3±1.4	98.1±1.3
Bank	84.1±1.8	84.4±1.7	83.9±1.9	84.6±1.5
Pendigits	98.3±0.3	98.3±0.3	98.3±0.3	97.8±0.4
Letter	86.6±0.5	86.4±0.5	86.5±0.5	86.4±0.8
MAGIC	74.6±2.1	75.1±2.1	75.7±1.8	73.9±2.0
Optical Digits	97.7±0.5	97.7±0.5	97.7±0.5	97.7±0.5
Semeion	90.4±0.1	90.4±0.1	90.4±0.1	90.4±0.1

(b) Incremental learning time [sec.]

Dataset	RAN-MRG	RAN-LSH $\theta_p = \infty$	RAN-LSH $\theta_p = 10$	RAN-LSH $\theta_p = 3$
Hiragana	48.0±4.5	42.9±4.0	10.5±1.6	7.4±1.1
Adult	102.4±10.8	90.1±9.0	30.0±6.2	14.9±2.6
Shuttle	1.6±0.4	1.3±0.5	1.2±0.5	1.2±0.4
Bank	17.3±1.7	15.7±1.8	15.2±1.9	6.8±1.0
Pendigits	1.9±0.1	1.7±0.1	1.0±0.0	0.9±0.0
Letter	218.1±13.8	179.0±16.4	156.0±8.9	8.5±0.3
MAGIC	18.8±1.7	16.2±2.2	4.5±0.8	2.6±0.3
Optical Digits	65.8±3.2	63.2±2.9	18.2±2.4	9.3±4.3
Semeion	124.5±1.2	121.9±3.1	8.9±1.2	4.1±0.6

(c) Total number of RBF bases

Dataset	RAN-MRG	RAN-LSH $\theta_p = \infty$	RAN-LSH $\theta_p = 10$	RAN-LSH $\theta_p = 3$
Hiragana	559.9±17.5	557.7±16.8	575.0±19.0	572.2±19.1
Adult	670.2±21.0	688.0±23.8	757.3±54.0	608.4±19.6
Shuttle	97.0±5.1	56.5±10.6	60.1±14.7	53.9±10.4
Bank	274.1±15.2	277.2±16.4	273.6±17.3	268.1±40.9
Pendigits	176.6±6.7	176.7±6.7	176.2±6.7	159.4±6.5
Letter	818.6±16.4	817.2±13.9	814.9±15.2	832.2±32.3
MAGIC	352.6±13.4	374.1±13.8	401.7±28.7	333.7±11.9
Optical Digits	727.4±10.4	727.9±10.1	727.7±10.2	728.3±10.3
Semeion	792.1±2.1	792.1±2.1	792.1±2.1	792.1±2.1

is sparsely distributed. One of the RBF bases selection strategy is to choose the active RBFs using Euclidean distance. Conversely, in our approach, the LSH distance is used to choose RBF bases that are near to the given training data. In this present study, we used θ_p to select the maximum LSH distance between a training data and all RBF centers in the hidden layer nodes. When a small value of θ_p is used, only a small area of the near RBF bases to a training data is involved and thus a few RBF centers would be selected and used

to update the weights. Meanwhile, when θ_p approached ∞, RAN-LSH model would act as such there is no RBF bases selection since the area of the nearest RBF centers would cover almost all regions in the input space. The advantage of our approach is evident through this experiment in that; it selects only one nearest RBF bases during learning by using $\theta_p = 3$, thus enabling this model to learn a large-scale data sequence in high-speed.

By selecting near RBF, we assumed that the active RBF bases would be selected to be learned together with a training data. To evaluate the performance of the proposed method, we compared the performance of four models; conventional RAN-MRG (without RBFs selection), RAN-LSH without RBFs selection ($\theta_p = \infty$), RAN-LSH with RBFs selection using $\theta_p = 10$ and RAN-LSH that selects the nearest RBFs ($\theta_p = 3$). Nine UCI datasets are used to evaluate the performance of these models.

Table 4 demonstrates (a) the final recognition accuracy, (b) the incremental learning time and (c) the final number of RBF bases created. In Table 4 (a) and (c), for RAN-LSH with $\theta = \infty$ (without RBF selection) model, it can be seen that the LSH-based data selection is comparable to the conventional margin-based data selection. This can be justified from the findings of recognition accuracy and final number of RBF bases created that are almost similar. Nevertheless, RAN-LSH with $\theta = \infty$ is capable to speed up the learning where all datasets show a time reduction during learning.

To shorten more learning time in RAN-LSH, we adopt the locally learning model (RAN-LSH with RBF selection). Local RAN-LSH model can avoid the high computional of SVD calculation because only a few RBFs in the hidden layer are selected to update the weight. For RAN-LSH with $\theta_p = 10$ model, the recognition accuracy and the final number of RBF bases are able to be maintained as previously models, RAN-MRG and RAN-LSH without RBF selection. Furthermore, the incremental learning time is successfully reduced to 35% of the incremental learning in RAN-MRG on average for five datasets which are Hiragana, Adult, MAGIC, Optical Digits and Semeion datasets using RAN-LSH ($\theta_p = 10$) model.

On the other hand, RAN-LSH with $\theta_p = 3$ is able to achieve at most 0.9% difference of recognition rate for Shuttle dataset compared to RAN-MRG. Whereas for the other datasets, the recognition accuracy shows a difference of only 0.5% or lower. This shows that it is sufficient to use only one nearest RBF center to be learned together with a given training data. In term of incremental learning time (see Table 4 (b)), six datasets, which are Hiragana, Adult, Letter, MAGIC, Optical Digits and Semeion datasets would require only 20% of the incremental learning in RAN-MRG on average, where 84.6%, 85.4%, 96.1%, 81.2%, 85.9% and 96.7% of learning times are speeden up, respectively. Whereas, for Shuttle, Bank and Pendigits datasets, at least 25% of the incremental learning time is shortened compare to RAN-MRG. Another advantage of RAN-LSH with $\theta_p = 3$ model is the final number of RBF bases shown to be lower than RAN-MRG for all datasets except Hiragana, Letter and Optical Digits where an extra 20 RBF bases are created at most. This result indicates that for especially large scale dataset, RAN-LSH using $\theta_p = 3$

model can effectively be used to reduce the learning time without sacrificing the recognition accuracy.

5 Conclusions

In this paper, we propose a fast incremental learning algorithm of modified RAN namely RAN-LSH which can be used to select essential data and near RBF bases. By utilizing this, the high cost for calculating high dimensional data can be lowered, as we can avoid the repetition of output margins and weights update calculations. In the present study, we introduced the LSH-based data selection and LSH-based RBF selection in the modified RAN classifier to solve two major issues in large-scale data sequences. Through our experiment, we managed to show that our proposed method of RAN-LSH with $\theta_p = 3$ is capable to learn a large chunk of data within a short period of time. This can be seen when only 20% of RAN-MRG's learning time on average is used by the proposed method, while keeping a similar accuracy rate to the conventional method.

Furthermore, the proposed learning scheme provides desirable characteristics for large-scale data sequences application. First, the learning is conducted in one-pass; which is only new training data would be used to be learned by the classifier, whereas the previous training data were discarded after the learning. This property is very important for learning large-scale data sequences where the data are growing in size, thus it is nearly impossible to learn and store a massive data without using one-pass learning. The other property is that the learning time is fast although large data chunk are given in every learning stage. From several experiments using nine datasets in the UCI Machine Learning Repository, we verify that the proposed scheme has the above desired characteristics.

There still remains several open problems in the proposed RAN-LSH. We can see that the fastest learning time is achieved by using LSH with two partitions. However, the recognition accuracy is slightly reduced if a small number of partitions is used. The reason is because the matrix of eigenvectors U_l are only determined during the initial training and does not updated incrementally. An alternative solution is to update eigenvectors U_l incrementally and adapt with the most sufficient number of partition to assign the hash value. This can be attained by introducing Incremental PCA (IPCA) (Ozawa et al. 2008) into the proposed RAN-LSH with two partitions. However, there is one problem for this. After updating hash functions, the hash table should also be updated properly without unexpected forgetting of the previous knowledge. If this problem is solved, it is expected that the idea of fast processing for large-scale data sequences using LSH would work faster.

The above issues are left as our future work.

Acknowledgements This work was financially supported by the University of Tun Hussein Onn Malaysia (UTHM) and the Ministry of Education (KPM).

References

Attar V, Sinha P, Wankhade K (2010) A fast and light classifier for data streams. Evolving Systems 1(3):199-207

Platt J (1991) A resource-allocating network for function interpolation. Neural Computation 3(2):213-225

Polikar R, Udpa L, Udpa SS, Honavar V (2001) Learn++: An incremental learning algorithms for supervised neural networks. IEEE Transactions on Systems, Man, and Cybernetics 31:497-508

Datar M, Immorlica N, Indyk P, Mirrokni VS (2004) Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of Symposium on Computational Geometry (SoCG'04):253-262

Andoni A, Indyk P (2008) Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Communications of the ACM 51(1):117-122

Gu X, Zhang Y, Zhang L, Zhang D, Li J (2013) An improved method of locality sensitive hashing for indexing large-scale and high-dimensional features. Signal Processing 93(8):2244-2255

Lee KM, Lee KM (2012) Similar pair identification using locality-sensitive hashing technique. In: Proceedings of Joint 6th International Conference on Soft Computing and Intelligent Systems (SCIS) and 13th International Symposium on Advanced Intelligent Systems (ISIS):2117-2119

Shen H, Li T, Li Z, Ching F (2008) Locality sensitive hashing based searching scheme for a massive database. In: Proceedings of IEEE SoutheastCon'08:123-128

Ozawa S, Toh SL, Abe S, Pang S, Kasabov N (2005) Incremental learning of feature space and classifier for face recognition. Neural Networks 18(5-6):575-584

Haykin S (1999) Neural networks: a comprehensive foundation. Prentice Hall

Kobayashi M, Zamani A, Ozawa S, Abe S (2001) Reducing computational in incremental learning for feedforward neural network with long-term memory. In: Proceedings International Joint Conference on Neural Networks:1989-1994

Okamoto K, Ozawa S, Abe S (2003) A fast incremental learning algorithm of RBF networks with long-term memory. In: Proceedings of International Joint Conference on Neural Networks:102-107

Ozawa S, Tabuchi T, Nakasaka S, Roy A (2010) An autonomous incremental learning algorithm for radial basis function networks. Journal of Intelligent Learning Systems and Applications 2:179-189

Dan R, Kevin S (2006) Margin-based active learning for structured output spaces. In: Proceedings ECML'06:413-424

Chellapilla K, Mityagin A, Charles DX (2007) GigaHash: scalable minimal perfect hashing for billions of urls. In: Proceedings of the 16th International Conference on World Wide Web

Bache K, Lichman M (2013) UCI machine learning repository [http://archive.ics.uci.edu/ml]. Accessed 01 January 2013

Davies DL, Bouldin DW (1979) A Cluster Separation Measure. IEEE Trans. on Pattern Analysis and Machine Intelligence 1(2):224-227

Ozawa S, Pang S, and Kasabov N (2008) Incremental learning of chunk data for on-line pattern classification systems. IEEE Trans. on Neural Networks 19(6):1061-1074