CHAPTER III

PROPOSED METHODOLOGY

The aim of this work is to develop the incremental learning algorithm to handle the streaming chunk of data under one-pass-thrown-away concept. In real-world application, the data stream has some special characteristics including the data distribution that can be changed as time passes, the number of data increases with time pass. The number of classes presented in each chunk is random in real and it cannot be predicted at the time that the new class appears. An example of data stream in two-dimensional space is shown in Figure 2. For the first stage t_1, there is only two classes identified by triangles and stars. After many stages pass, the data chunk with new class expressed in the circles is presented in stage t_i.

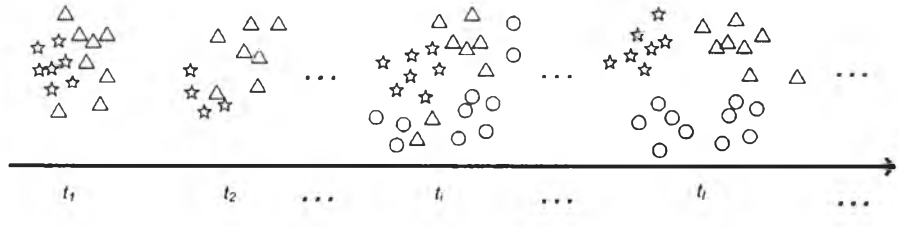


Figure 2: Example of streaming chunk data classification in two-dimensional space.

Furthermore, the proposed incremental learning algorithm is also dealing with a large size of data chunk. The proposed learning algorithm such that the number of repetitions of all learning n data of a chunk is bounded at $O(n^2)$. To reach the bound of $O(n^2)$ and eliminate the sensitivity of learning data sequence, the learning process is learned by one class at a time. Once the data in any class are learned, they are thrown away and never learned again forever. Furthermore, to reach the minimum number of required neurons for any class, it is essential to estimate the number of distributed sub-clusters first and capture these sub-clusters by a set of

radial activation functions to reduce the effect of misclassification. To manage the most concerned factors which are (1) the number of uncontrollable iterations, (2) the unpredictable number of hidden neurons, and (3) the unknown prior data distribution, the following problems must be addressed.

(1) How can capture the data of each sub-cluster in any class within the bounded learning time complexity of $O(n_i^2)$, where n_i is the number of data in sub-cluster i ?

(2) Given a set of data in all classes, what is the minimum number of neurons to be deployed so that the learning time complexity is bounded by $O(n^2)$, where n is the number of a data chunk, regardless of classes ?

Since the size and direction of the radial activation function proposed in [4] can be algorithmically adjusted, it is adopted as a part of neuron operation. However the learning process cannot be directly adapted to resolve the studied problems. In [4], an incremental learning algorithm for dealing with one datum at a time was proposed. Although multiple data points were available, the learning algorithm still used only one datum for parameter update. This may cause the problem of creating too many VEBF and increase unnecessary computational time during the learning process. Furthermore, the performance of their proposed algorithm is dramatically affected from the order of presented training data points and the case of streaming data chunk is also not considered. To improve this disadvantage, our algorithm learns more than one datum of any class at a time and threw them away afterwards.

In this work, Data-throwaway Learning for streaming chunk data classification (DLSC) for Versatile Elliptic Basis Function Neural Network (VEBFNN) is proposed. Based on the class-wise learning, the proposed method can be dealing with the burst of new class. For each chunk, the pattern of presented data to the proposed learning process is class-wise as shown in Figure 3. t_{il} stands for data with the same class obtained from data chunk at stage i^{th} with the l^{th} class label.

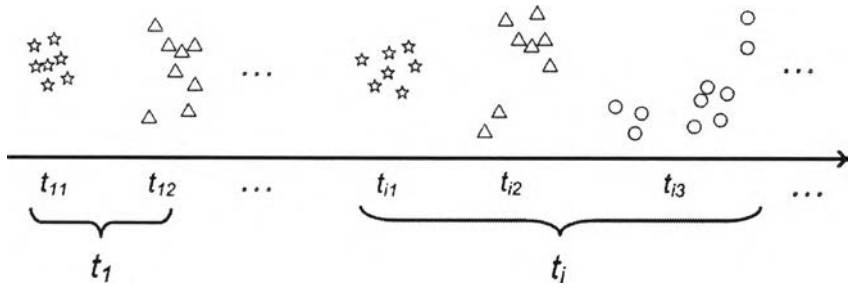


Figure 3: Example of class-wise streaming chunk in two-dimensional space.

3.1 Parameters Update

Let $\mathbb{D} = \{\{\mathbf{X}^t, t^t\} | \mathbf{X}^t \subset \mathbb{R}^n$ and $t^k \in \mathbb{I}^+$ for $i = 1,2,...\}$ be a streaming chunk data chunk $\mathbf{X}^t$ with class label t^t. The parameters update of a VEBF neuron for handling streaming chunk is concerned instead of feeding only one datum to VEBFNN at a time. Considering a chunk of data presented to the network at a time, each data chunk may consist of several sub-clusters distributed at different locations. A set of VEBF neurons is incrementally introduced during the learning process to cover each sub-cluster and the relevant VEBF parameters are updated according to the data in the sub-cluster. The learning process focuses on how to a VEBF neuron updating the relevant parameters of network to the incoming data chunk without keeping all previous learnt data chunk.

3.1.1 Center vector and covariance matrix update

Assume $\bar{\mathbf{x}}_c$ and $\mathbf{S}_c$ are the current center vector and covariance matrix computed from the previous m data points, respectively. Let $\mathbf{X} = \{\mathbf{x}_i \in \mathbb{R}^n \mid i = 1,2,...,q\}$ be a new data chunk with q samples and $\bar{\mathbf{x}}_l$ refers to the local center vector computed from the new data chunk $\mathbf{X}$ by $\bar{\mathbf{x}}_l = \frac{1}{q}\sum_{i=1}^{q}\mathbf{x}_i$. The updated center vector $\bar{\mathbf{x}}_u$ with respect to the current and local center vectors, $\bar{\mathbf{x}}_c$ and $\bar{\mathbf{x}}_u$, can be expressed by

$$\bar{\mathbf{x}}_u = \frac{1}{m+q}(m\bar{\mathbf{x}}_c + q\bar{\mathbf{x}}_l). \quad (21)$$

Theoretically, a covariance matrix $\mathbf{S}_u$ of both previous and current data sets is computed by

$$\mathbf{S}_u = \frac{1}{m+q}\sum_{i=1}^{m+q}\mathbf{x}_i\mathbf{x}_i^T - \bar{\mathbf{x}}_u\bar{\mathbf{x}}_u^T, \tag{22}$$

where the superscript T stands for transpose vector or matrix. However, this covariance matrix derived by this equation is not suitable for incremental manner. For incremental learning concept, the updated covariance matrix $\mathbf{S}_u$ can be computed by

$$\mathbf{S}_u = \frac{m}{m+q}\left(\mathbf{S}_c + \bar{\mathbf{x}}_c\bar{\mathbf{x}}_c^T\right) + \frac{1}{m+q}\sum_{j=1}^{q}\mathbf{x}_j\mathbf{x}_j^T - \bar{\mathbf{x}}_u\bar{\mathbf{x}}_u^T \tag{23}$$

3.1.2 Width vector update

The concept of Data-throwaway Learning for streaming chunk data classification (DLSC) requires adjusting the width vector $\mathbf{w}_j^k$ of the j^{th} VEBF $\psi_j^k(\mathbf{x}:\bar{\mathbf{x}}_j^k,\mathbf{w}_j^k,\mathbf{U}_j^k)$ to cover any new unlearned data. The unlearned data will be covered by the nearest VEBF $\psi_j^k(\mathbf{x}:\bar{\mathbf{x}}_j^k,\mathbf{w}_j^k,\mathbf{U}_j^k)$. Because all learned data are already thrown away, by the similarity to the new center and covariance matrix computation, a recurrence function for computing the new width must be formed. Let $\mathbf{w}_j^k = [w_{j1}^k\ w_{j2}^k\ \cdots\ w_{jn}^k]^T$ be the width vector of the j^{th} VEBF neuron. Suppose that $\bar{\mathbf{x}}_{j_{old}}^k$ and $\bar{\mathbf{x}}_{j_{new}}^k$ are the centers of the neuron Ω_j^k before and after covering the unlearned data, respectively. The value of each w_{jl}^k is adjusted with respect to the orthonormal basis vectors $\mathbf{U}_l^k = [\mathbf{u}_{j1}^k\ \mathbf{u}_{j2}^k\ \cdots\ \mathbf{u}_{jn}^k]$ by the following equation.

$$w_{jl}^k = w_{jl}^k + |\,(\bar{\mathbf{x}}_{j_{new}}^k - \bar{\mathbf{x}}_{j_{old}}^k)^T\mathbf{u}_{jl}^k\,|, \text{ for } l = 1,2,\ldots,n. \tag{24}$$

However it is not guaranteed that the width vector adjusted by Equation (24) can cover all unlearnt data points $\mathbf{y}_t \in \mathbf{Y}$. So, every data point $\mathbf{y}_t$ is checked first by the following condition to see whether it is covered by the closest j^{th} VEBF.

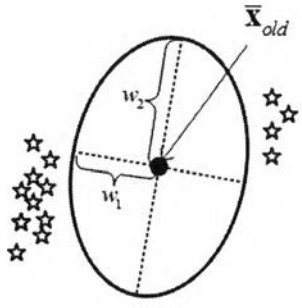
$$max_{val} = \max_{\mathbf{y}_t \in \mathbf{Y}} (\psi_j^k(\mathbf{y}_t, \bar{\mathbf{x}}_{j_{new}}^k, \mathbf{w}_j^k, \mathbf{U}_j^k)) \tag{25}$$

By definition 1, if $max_{val} > 0$, i.e. there exists a set of uncovered data, then adjusting the width vector by the following equation,

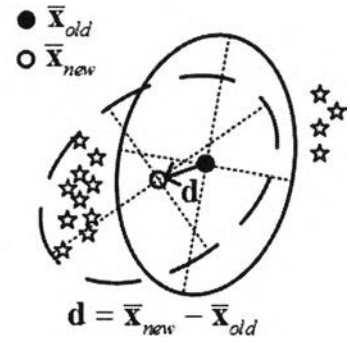
$$\mathbf{w}_j^k = \varepsilon \mathbf{w}_j^k \tag{26}$$

where $\varepsilon = \sqrt{1 + \eta * max_{val}}$ and $\eta \geq 1$. Based on Equation (26), there is no uncovered data as shown in Theorem 1 in Appendix.

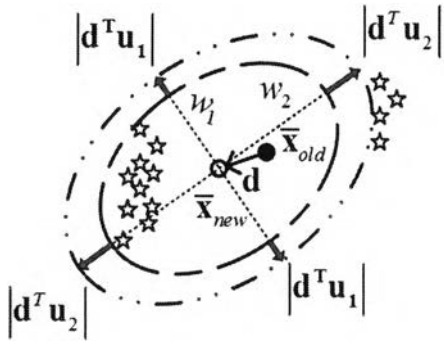
The geometric interpretation of parameter update in two-dimensional space is illustrated in Figure 4(a) - Figure 4(e). All new incoming data of the same class are represented by stars. Suppose that those already learned data are covered by the VEBF shown in a form of elliptic shape in Figure 4(a). The center of VEBF is at $\bar{\mathbf{x}}_{old}$ and the widths are $\mathbf{w}_1$ and $\mathbf{w}_2$. To cover the new incoming data, the following steps are performed. The center of VEBF is moved to the new center $\bar{\mathbf{x}}_{new}$ of data as shown in Figure 4(b). The vector $\mathbf{d}$ represents the center movement vector. The eigenvector $\mathbf{u}_t$ of data is used to rotate and adjust the widths of VEBF, based on the absolute of scalar projection of the center movement vector $\mathbf{d}$ on each eigenvector $\mathbf{u}_t$, all new incoming data as shown in Figure 4(c) to Figure 4(e), respectively.



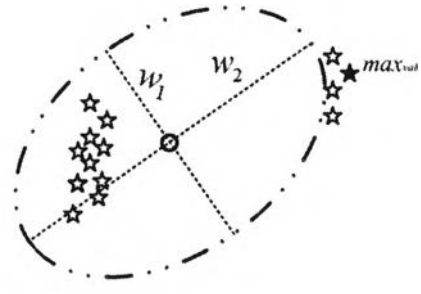
(a) A VEBF neuron with unlearned data



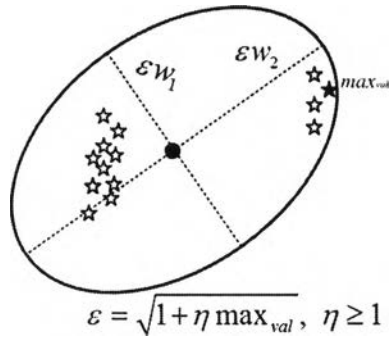
(b) Comparison of before and after update of center and covariance matrix



(c) Update the width vector by $|\mathbf{d}^T\mathbf{u}_l|$ for l = 1,2



(d) Four uncovered data



(e) Update width vector

Figure 4: An example of how to cover new incoming data by updating VEBF parameters

3.1.3 Merge Criterion

For the sub-hidden layer Λ^k, hidden neurons adjust parameters every time when data chunk is presented. There are some redundant hidden neurons having common data points. Two hidden neurons $\Omega_i^k = (\bar{\mathbf{c}}_i^k, \mathbf{w}_i^k, \mathbf{S}_i^k, N_i^k)$ and $\Omega_j^k = (\bar{\mathbf{c}}_j^k, \mathbf{w}_j^k, \mathbf{S}_j^k, N_j^k)$ are merged, if at least one hidden neuron cover another center identified by $\psi_i^k(\bar{\mathbf{c}}_j^k : \bar{\mathbf{c}}_i^k, \mathbf{w}_i^k, \mathbf{U}_i^k) < 0$ or $\psi_j^k(\bar{\mathbf{c}}_i^k : \bar{\mathbf{c}}_j^k, \mathbf{w}_j^k, \mathbf{U}_j^k) < 0$. The new parameters of the merged hidden neuron $\Omega_m = (\bar{\mathbf{c}}_m, \mathbf{w}_m, \mathbf{S}_m, N_m)$ can be computed as follows [4]:

$$N_m = N_i^k + N_j^k, \tag{27}$$

$$\bar{\mathbf{c}}_m = \frac{1}{N_m^k}(N_i^k \bar{\mathbf{c}}_i^k + N_j^k \bar{\mathbf{c}}_j^k), \tag{28}$$

$$\mathbf{S}_m = \frac{N_i^k}{N_m^k}\mathbf{S}_i^k + \frac{N_j^k}{N_m^k}\mathbf{S}_j^k + \frac{N_i^k N_j^k}{N_m^k}(\bar{\mathbf{c}}_i^k - \bar{\mathbf{c}}_j^k)(\bar{\mathbf{c}}_i^k - \bar{\mathbf{c}}_j^k)^T, \tag{29}$$

$$\mathbf{w}_m = \sqrt{2\pi \mid \lambda_l \mid}, \; l = 1, 2, ..., n \tag{30}$$

where λ_l is the l^{th} eigenvalue of the new covariance matrix $\mathbf{S}_m$. The merge process for any two neurons is performed if and only if at least one neuron covers the center of another neuron. Two conditions of merge process are illustrated in Figure 5.

(a) One center is covered (b) Two centers are covered

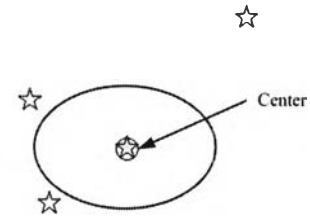
Figure 5: Two cases of merge process.

3.2 The Proposed Learning Algorithm

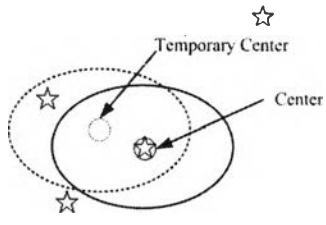
An example of how to apply the DLSC algorithm to 2-dimensional data in the same class is illustrated in Figure 6. Suppose there are four data samples denoted by four star signs as shown in Figure 6(a). First, a new ellipsoidal hidden neuron with initial width is created and a data point is randomly selected as the center of this new hidden neuron and covered by the neuron as shown in Figure 6(b). Next, without adjusting the size of the ellipsoidal neuron, the neuron is temporarily shifted towards the other new data point to cover it so that the new center is located in the middle between the old center and the new data point as shown in Figure 6(c) - Figure 6(e). During this process, any new data satisfying this condition is marked. Figure 6(c) - Figure 6(d) show two new marked data points. But some new data points in Figure 6(e) do not satisfy the condition because their locations are beyond the space possibly covered by the ellipsoidal neuron. The size of previously created ellipsoidal is adjusted to cover these two new marked data point as shown in Figure 6(f). All covered data points are thrown away afterwards and only the center is kept as shown in Figure 6(g). In Figure 6(h), the uncovered data point is covered by the second newly created ellipsoidal neuron.

In this work, Data-throwaway Learning for Streaming Chunk (DLSC) algorithm is proposed to handle a continuous learning scenario. The streaming chunk data with the same class label of training samples is successively presented to a learning system called a stream of training data chunks. Assume $\mathbb{D}$ be a stream of training data chunks defined by $\mathbb{D} = \{\wp^i \mid \wp^i = \left\{\{\mathbf{X}^{ij}, t^{ij}\}_{j=1}^{p_i}\right\},\ \mathbf{X}^{ij} \subset \mathbb{R}^n, t^{ij} \in \mathbb{I}^+ \text{ for } i = 1,2,...\}$, where t^{ij} and p_i is the class label of a data chunk $\mathbf{X}^{ij}$ and the number of class label of a data chunk $\wp^i$. The details of DLSC algorithm are summarized as follows:

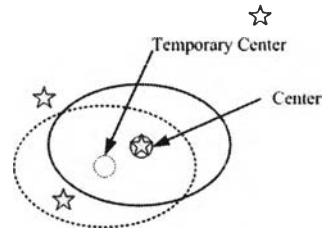
(a) Four uncovered data point



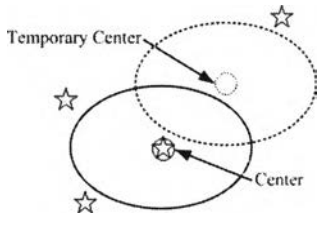
(b) Creating a new hidden neuron



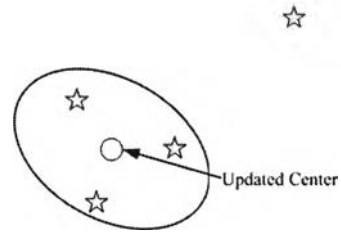
(c) Parameter update for the left most point



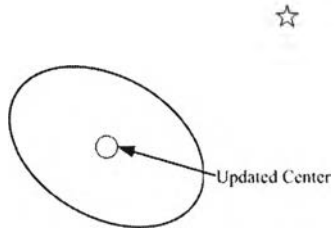
(d) Parameter update for the lowest point



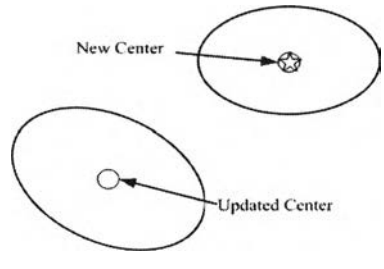
(e) Avoid the outside point for parameter update



(f) Updating parameter to cover three point



(g)Throwaway the covered point



(h) Creating another new neuron

Figure 6: Example of the proposed method in two-dimensional space.

3.2.1 Data-throwaway Learning Streaming Chunk (DLSC) algorithm

Input: A data chunk $\wp^i = \left\{\{\mathbf{X}^{ij}, t^{ij}\}_{j=1}^{p}\right\}$ and the VEBFNN with K hidden neurons where $K \in \mathbb{I}^+$.

Output: The VEBFNN $\Gamma = \{\Lambda^1, \Lambda^2, \ldots, \Lambda^r\}$ learned by DLSC algorithm.

Line 1: Let $\Gamma = \{\Lambda^1, \Lambda^2, \ldots, \Lambda^r\}$ and $r = 0$.

Line 2: Initialize the width vector as $\mathbf{w}^0 = [w_1^0 \; w_2^0 \; \cdots \; w_n^0]^T$

Line 3: Present a data chunk $\wp^i = \left\{\{\mathbf{X}^{ij}, t^{ij}\}_{j=1}^{p}\right\}$ to the network Γ.

Line 4: For j=1,2,...,p

Line 5: Present $\{\mathbf{X}^{ij}, t^{ij}\}$

Line 6: If t^{ij} is a new class label **then**

Line 7: Set $r = r+1$, $d_r = 0$ and $\Lambda^r = \{\}$.

Line 8: **Do**

Line 9: Set $d_r = d_r + 1$.

Line 10: Create new hidden neuron by $[\mathbf{X}^{ij}, \Omega_{d_r}^r] = CreateNewNeuron(\mathbf{X}^i)$

Line 11: Update parameters of $\Omega_{d_r}^r$ by

$$[\mathbf{X}^{ij}, \Omega_{d_r}^r, update] = UpdateParameter(\eta, \mathbf{X}^{ij}, \Omega_{d_r}^r, \mathbf{Y}) \; .$$

Line 12: Add $\Omega_{d_r}^r$ to Λ^r by $\Lambda^r = \Lambda^r \cup \{\Omega_{d_r}^r\}$.

Line 13: **While** $\mathbf{X}^{ij} \neq \varnothing$

Line 12: Add the r^{th} sub-hidden layer Λ^r by $\Gamma = \Gamma \cup \{\Lambda^r\}$.

Line 14: **Else**

Line 15: **Do**

Line 16: Compute the center vector $\bar{\mathbf{x}}$ for data chunk $\mathbf{X}^{ij}$ by

$$\bar{\mathbf{x}} = \frac{\sum_{b=1}^{|\mathbf{X}^{ij}|} \mathbf{x}_b^{ij}}{|\mathbf{X}^{ij}|} .$$

Line 17: Find the k^{th} sub-hidden layer Λ^k corresponding to $k = t^{ij}$.

Line 18: Assign the active VEBF neuron by

$$active = \arg\min_{1 \le j \le d_k} \{\psi_j^k(\bar{\mathbf{x}} : \mathbf{c}_j^k, \mathbf{w}_j^k, \mathbf{U}_j^k)\}$$

Line 19: $[\mathbf{X}^{ij}, \Omega_{active}^k, update] = UpdateParameter(\eta, \mathbf{X}^{ij}, \Omega_{active}^k, \mathbf{Y})$

Line 20: If $update \neq 0$ then

Line 21: Perform merging process by $\Lambda^k = MergeProcess(active, \Lambda^k)$

Line 22: Else

Line 23: Go to Line 26

Line 24: EndIf

Line 25: While $update \neq 0$ and $\mathbf{X}^{ij} \neq \varnothing$.

Line 26: If $\mathbf{X}^{ij} \neq \varnothing$ then

Line 27: Do

Line 28: Set $d_k = d_k + 1$.

Line 29: $[\mathbf{X}^{ij}, \Omega_{d_k}^k] = CreateNewNeuron(\mathbf{X}^{ij})$.

Line 30: $[\mathbf{X}^{ij}, \Omega_{d_k}^k, update] = UpdateParameter(\eta, \mathbf{X}^{ij}, \Omega_{d_k}^k, \mathbf{Y})$

Line 31: Add $\Omega_{d_k}^k$ to $\Lambda^k = \Lambda^k \cup \{\Omega_{d_k}^k\}$.

Line 32: While $\mathbf{X}^{ij} \neq \varnothing$

Line 33: EndIf

Line 34: EndIf

Line 35: EndFor

Line 36: Stop training

Create the new hidden neuron $\Omega = (\bar{\mathbf{c}}, \mathbf{w}, \mathbf{S}, N)$.

function $[\mathbf{X}, \Omega] = CreateNewNeuron(\mathbf{X})$

Input: The data chunk $\mathbf{X}$.

Output: The data chunk $\mathbf{X}$ reduced by one and the new hidden neuron Ω.

Line 1: Select randomly a data vector $\mathbf{x} \in \mathbf{X}$.

Line 2: Set the center vector $\bar{\mathbf{c}}$ by $\bar{\mathbf{c}} = \mathbf{x}$.

Line 3: Set the covariance matrix by $\mathbf{S} = \bar{\mathbf{0}}$ where $\bar{\mathbf{0}}$ is the null matrix.

Line 4: Set the orthonormal basis by $\mathbf{U} = \mathbf{I}_{n \times n}$, where $\mathbf{I}_{n \times n}$ is identity matrix.

Line 5: Set the number of covered data by $N = 1$.

Line 6: Set $\Omega = (\bar{\mathbf{c}}, \mathbf{w}, \mathbf{S}, N)$

Line 7: Discard $\mathbf{x}$ from $\mathbf{X}$, $\mathbf{X} = \mathbf{X} - \{\mathbf{x}\}$.

Line 8: **Return** $\mathbf{X}$ and Ω.

Update Parameters

function $[\mathbf{X}, \Omega, update] = UpdateParameter(\eta, \mathbf{X}, \Omega_o, \mathbf{Y})$

Input: The chunk data set $\mathbf{X}$, the current hidden neuron $\Omega_o = (\bar{\mathbf{c}}_o, \mathbf{w}_o, \mathbf{S}_o, N_o)$, the parameter η, and the covered data set $\mathbf{Y}$.

Output: The remain data set $\mathbf{X}$, the updated neuron $\Omega = (\bar{\mathbf{c}}, \mathbf{w}, \mathbf{S}, N)$ and *update* variable.

Line 1: Set $update = 0$.

Line 2: $\mathbf{Y} = FormCoveredData(\mathbf{X}, \Omega)$

Line 3: If $\mathbf{Y} \neq \varnothing$ then

Line 4: $update = 1$

Line 5: Do

Line 6: Set $Z = |\mathbf{Y}|, \mathbf{c}_o = \bar{\mathbf{c}}_a^k$ and $N_o = N_a^k$.

Line 7: Update N by $N = N_o + Z$.

Line 8: Compute the center vector $\bar{\mathbf{y}}$ of $\mathbf{Y}$ by $\bar{\mathbf{y}} = \frac{\sum_{\mathbf{y}_q \in \mathbf{Y}} \mathbf{y}_q}{Z}$.

Line 9: Update the center vector $\bar{\mathbf{c}}$ by $\bar{\mathbf{c}} = \frac{1}{N}(N_o \bar{\mathbf{c}}_o + Z\bar{\mathbf{y}})$.

Line 10: Update the covariance matrix $\mathbf{S}$ by

$$\mathbf{S} = \frac{N_o}{N}(\mathbf{S}_o + \bar{\mathbf{c}}_o \bar{\mathbf{c}}_o^T) + \frac{1}{N}\sum_{q=1}^{Z} \mathbf{y}_q \mathbf{y}_q^T - \bar{\mathbf{c}}(\bar{\mathbf{c}})^T.$$

Line 11: Compute new orthonormal basis vectors $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n]$ from the updated $\mathbf{S}$.

Line 12: Update the width vector $\mathbf{w} = [w_1, w_2, \ldots, w_n]^T$ by $w_i = w_{oi} + |(\bar{\mathbf{c}} - \mathbf{c}_o)^T \mathbf{u}_i|$ for $i = 1, 2, \ldots, n$

Line 13: Compute $max_{val} = \max_{\mathbf{y}_q \in \mathbf{Y}}(\psi(\mathbf{y}_q : \bar{\mathbf{c}}, \mathbf{w}, \mathbf{U}))$

Line 14: If $max_{val} > 0$ then

Line 15: $\mathbf{w} = \mathbf{w}\sqrt{1 + \eta * max_{val}}$

Line 16: EndIf

Line 17: Discard the covered data $\mathbf{Y}$ from $\mathbf{X}$ by $\mathbf{X} = \mathbf{X} - \mathbf{Y}$.

Line 18: If $\mathbf{X} = \varnothing$ then

Line 19: $\mathbf{Y} = \varnothing$.

Line 20: Else

Line 21: $\mathbf{Y} = FormCoveredData(\mathbf{X}, \Omega)$

Line 20: EndIf

Line 22: While $\mathbf{Y} \neq \varnothing$

Line 23: EndIf

Line 24: Return $\mathbf{X}$, $\Omega = (\bar{\mathbf{c}}, \mathbf{w}, \mathbf{S}, N)$ and *update*

Form Covered Data Set $\mathbf{Y}$

function $\mathbf{Y} = FormCoveredData(\mathbf{X}, \Omega)$

Input: The data chunk $\mathbf{X}$ and a active neuron $\Omega = (\bar{\mathbf{c}}, \mathbf{w}, \mathbf{S}, N)$.

Output: The covered data set $\mathbf{Y}$.

Line 1: Set $\mathbf{Y} = \varnothing$.

Line 2: For each vector $\mathbf{x} \in \mathbf{X}$.

Line 3: Compute $\mathbf{c}^{tmp} = \frac{1}{N+1}(N\overline{\mathbf{c}} + \mathbf{x})$

Line 4: Compute $\psi(\mathbf{x} : \mathbf{c}^{tmp}, \mathbf{w}, \mathbf{U}) = \sum_{l=1}^{n} \frac{((\mathbf{x} - \mathbf{c}^{tmp})^T \mathbf{u}_l^k)^2}{(w_l^k)^2} - 1$

Line 5: If $\psi(\mathbf{x} : \mathbf{c}^{tmp}, \mathbf{w}, \mathbf{U}) \leq 0$ then

Line 6: $\mathbf{Y} = \mathbf{Y} \cup \{\mathbf{x}\}$.

Line 7: EndIf

Line 8: EndFor

Line 9: Return $\mathbf{Y}$

Merging Process within the k^{th} sub-hidden layer Λ^k

function $\Lambda^k = MergeProcess(i, \Lambda^k)$

Input: The index i of checked neuron and the k^{th} sub-hidden layer, $\Lambda_{d_k}^k$.

Output: The k^{th} sub-hidden layer Λ^k, after merging process.

Line 1: Apply PCA method with the covariance matrix $\mathbf{S}_i^k$ to obtain the orthonormal basis $\mathbf{U}_i^k = [\mathbf{u}_{il}^k]_{l=1}^n$ with corresponding $\lambda_{i1}^k > \lambda_{i2}^k > ... > \lambda_{in}^k$.

Line 2: For $j = 1, 2, ..., i-1, i+1, ..., d_k$

Line 3: Apply PCA method with the covariance matrix $\mathbf{S}_j^k$ to obtain the orthonormal basis matrix $\mathbf{U}_j^k = [\mathbf{u}_{jl}^k]_{l=1}^n$ with corresponding $\lambda_{j1}^k > \lambda_{j2}^k > ... > \lambda_{jn}^k$.

Line 4: Compute $\psi_j^k(\overline{\mathbf{c}}_i^k : \overline{\mathbf{c}}_j^k, \mathbf{w}_j^k, \mathbf{U}_j^k)$ and $\psi_i^k(\overline{\mathbf{c}}_j^k : \overline{\mathbf{c}}_i^k, \mathbf{w}_i^k, \mathbf{U}_i^k)$.

Line 5: If $\psi_j^k(\overline{\mathbf{c}}_i^k : \overline{\mathbf{c}}_j^k, \mathbf{w}_j^k, \mathbf{U}_j^k) < 0$ or $\psi_i^k(\overline{\mathbf{c}}_j^k : \overline{\mathbf{c}}_i^k, \mathbf{w}_i^k, \mathbf{U}_i^k) < 0$ then

Line 6: Compute $N_m = N_i^k + N_j^k$.

Line 7: Compute $\bar{\mathbf{c}}_m = \frac{1}{N_m}(N_i^k \bar{\mathbf{c}}_i^k + N_j^k \bar{\mathbf{c}}_j^k)$.

Line 8: Compute $\mathbf{S}_m = \frac{N_i^k}{N_m^k}\mathbf{S}_i^k + \frac{N_j^k}{N_m^k}\mathbf{S}_j^k + \frac{N_i^k N_j^k}{N_m^k}(\bar{\mathbf{c}}_i^k - \bar{\mathbf{c}}_j^k)(\bar{\mathbf{c}}_i^k - \bar{\mathbf{c}}_j^k)^T$.

Line 9: Compute $\mathbf{w}_m = \sqrt{2\pi \mid \lambda_l \mid}, \quad l = 1,2,...,n$ where λ_l is the l^{th} eigenvalue of the new covariance matrix $\mathbf{S}_m$.

Line 10: Set $\Omega_i^k = (\bar{\mathbf{c}}_m, \mathbf{w}_m, \mathbf{S}_m, N_m)$.

Line 11: Delete Ω_j^k from the k^{th} sub-hidden layer, $\Lambda^k = \Lambda^k - \{\Omega_j^k\}$.

Line 12: Set $d_k = d_k - 1$.

Line 13: **Go to** line 17

Line 14: **EndIf**

Line 15: **EndFor**

Line 16: **Return** Λ^k.

From DLSC algorithm, the VEBFNN starts with the *K* hidden neurons. Two cases of a class data chunk $\{\mathbf{X}^{ij}, t^{ij}\}$ are considered i.e., t^{ij} is new class label or t^{ij} is an old class label. For the new class label, two main operations are successively performed called *CreateNewNeuron* and *UpdateParameter*. The new sub-hidden layer responsible for the class t^{i} and new hidden neuron are created for *CreateNewNeuron* process. Then, the created hidden neuron adapts itself through parameter update to cover as many data points in $\mathbf{X}^{ij}$ as possible for *UpdateParameter* process. Within *UpdateParameter* process, there is an important sub-process called *FormCoveredData* sub-process. This sub-process selects the set of data points in $\mathbf{X}^{ij}$ called covered data set $\mathbf{Y}$. The parameters are updated only once for the covered data set $\mathbf{Y}$. The covered data points are thrown away from the data chunk $\mathbf{X}^{ij}$. The parameter update for a hidden neuron is repeated until there is no covered data point. These two main operations, *CreateNewNeuron* and *UpdateParameter*, are repeatedly performed until all data points in $\mathbf{X}^{ij}$ are discarded entirely. For the old class label t^{ij}, the old hidden neurons constructed from previous data chunks handle the incoming data chunk $\mathbf{X}^{ij}$. The learning process starts computing the mean vector of the data chunk $\mathbf{X}^{ij}$. After assigning the closest hidden neuron with respect to the mean vector, the *UpdateParameter* process is performed. The

distribution of the data with the class label t^{ij} always obtains the stream of data chunks. Therefore, the merging process called MergeProcess is needed for the old hidden neuron dealing with an incoming data chunk. Both *UpdateParameter* and MergeProcess are performed repeatedly until there is also no covered data. The rest data points in data chunk $\mathbf{X}^{ij}$ are viewed like case of new class label. The *CreateNewNeuron* and *UpdateParameter* processes are performed until the data chunk $\mathbf{X}^{ij}$ is empty. The time complexity T_{alg} of the Data-throwaway Learning Streaming Chunk (DLSC) algorithm is $O(K+n^2)$, where K is the number of hidden neurons before presenting a data chunk $\wp$. The proof of time complexity is given in Theorem 2 in Appendix.