Received May 31, 2020, accepted June 15, 2020, date of publication June 18, 2020, date of current version July 1, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.3003531*

# Neural Learning With Recoil Behavior in Hyperellipsoidal Structure

**KANOKSILP JINDADOUNGRUT, SUPHAKANT PHIMOLTARES, (Member, IEEE), AND CHIDCHANOK LURSINSAP, (Member, IEEE)**

Advanced Virtual and Intelligent Computing (AVIC) Research Center, Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University, Bangkok 10330, Thailand

Corresponding author: Suphakant Phimoltares (suphakant.p@chula.ac.th)

This work was supported by the Thailand Research Fund (TRF) under Grant RTA6080013.

**ABSTRACT** In recent years, the quantity of digital data being generated has increased considerably and is overwhelming the storage capacity. To overcome this problem, acquiring more and larger data storage is the simplest solution. But this solution is rather costly and may produce poisonous electronic garbage. A new fast and memory-efficient algorithm for learning and classifying these data without increasing the space and time complexities more than those of current learning and classifying algorithms is desirable. Although many one-pass online or incremental learning algorithms based on hyperellipsoidal functions for streaming data without retaining any learned data in fixed storage have been successfully developed for training streaming data, achieving high accuracy of any testing dataset is unstable and uncontrollable, depending on the experimental datasets. This paper proposes an improvement to these one-pass and fixed-storage learning algorithms so that the high accuracy of testing data can be significantly improved and stabilized, regardless of the experimental datasets. The concept is based on animal recoil behavior, which occurs when an animal moves away suddenly from something it dislikes. The behavior is mathematically modeled in forms of shrinking and shifting the hyperellipsoidal function during the training period to improve testing accuracy. The experimental results on 15 datasets improved the accuracy up to 8.16 % and also provided the highest or near-highest accuracy results in 10 datasets when compared to other algorithms.

**INDEX TERMS** Discard-after-learn, hyperellipsoidal function, incremental learning, recoil behavior.

## I. INTRODUCTION

The quantity of digital data generated in recent years has been increasing rapidly due to the advancement of information technology. The term ''big data'', which refers to the massive datasets that often have a complex structure and are created at a rapid pace, is currently being widely used. Some examples frequently seen in the literature include retail business data, online social network user interaction data, measurements from scientific sensor networks, and hospital patient profiles [1]. Businesses and organizations can gain knowledge and possibly competitive advantage if these data can be analyzed rapidly and accurately, although the quantity of data makes it impractical for manual processing by human staff.

One of the methods for analyzing and processing information in big datasets is the artificial neural network. Many types of neural networks have been developed with different concepts, performances, strengths, and weaknesses. For example, multilayer perceptron (MLP) is one of the traditional neural networks that works by adjusting the synaptic weights to reduce classification or prediction error based on the training data. The training data are fed into the network repeatedly until the network achieves satisfactory results. In addition, many datasets currently being generated are streaming data. New data samples are being created and need to be learned incrementally, rather than having the whole dataset available at once [2], which can cause the memory overflow condition. Very large datasets that cannot be fit in the memory all at once can also be handled as data streams [3]. A fast algorithm that can learn streaming data without increasing the space complexity of memory size and the time complexity of the learning algorithm is thus desirable for handling modern datasets. Many attempts for online learning algorithms have been reported over the past years. Some of these methods are briefly summarized as follows.

The approximate large margin algorithm (ALMA) proposed by [4] is an incremental learning algorithm that attempts to approximate a separating hyperplane between

The associate editor coordinating the review of this manuscript and approving it for publication was Mohammad Ayoub Khan.

two classes of data such that the margin is maximized. This method is similar to the support vector machine (SVM) but differs in that SVM attempts to compute the solution for the hyperplane using all available training data at once, i.e., SVM is a batch learning algorithm, while ALMA adjusts its separating hyperplane parameters after learning datum one-by-one.

Online gradient descent (OGD) introduced by [5] is a gradient descent algorithm that operates in an online learning environment. Gradient descent is a method for solving optimization problems by iteration. In neural networks, a cost function is usually defined as the difference between the actual outputs and the target outputs. Gradient descent is used to find a set of optimal parameters for the network to minimize the cost function.

Second-order perceptron (SOP) proposed by [6] is an extension to the classic Rosenblatt's perceptron algorithm [7]. The classic perceptron is essentially a gradient descent algorithm that updates the weight vectors according to an error function. The SOP algorithm also considers the information of a data correlation matrix, which can be incrementally computed from new data to classify data more accurately.

Crammer *et al*. [8] presented an online margin-based classification algorithm called the passive-aggressive (PA) algorithm. The concept is to aggressively update the separating hyperplane to achieve at least a certain margin size on the most recent training data while also attempting to remain as close to the current state as possible. This method can be very sensitive to noise or mislabeled data samples due to the forced margin update.

The confidence-weighted (CW) classifier was introduced by [9]. CW learning is an online learning method that adds parameter confidence information. During the learning process, parameters and their confidence values are adjusted based on training data. Parameters with low confidence values are adjusted more aggressively, while high-confidence parameters are less sensitive to change. Crammer *et al*. [10] presented the exact convex confidence-weighted (ECCW) algorithm based on CW learning with a change in the optimization constraint. These two methods provide good classification accuracy on high-dimensional natural language processing (NLP) datasets.

Adaptive regularization of weight vectors (AROW) proposed by [11] is an online learning method that combines the concept of large-margin training and confidence weighting found in earlier works. AROW is similar to SOP with the difference that SOP only updates its weight vectors when it makes a prediction error. AROW also updates when the prediction is correct but the margin is not large enough. This method can handle label noise better than CW learning.

Xiao proposed the regularized dual averaging (RDA) and enhanced regularized dual averaging (ERDA) methods [12]. The optimization in these methods involves computing the running average of past subgradients of the loss function and also includes a regularization term. The focus of these methods is to create a sparse neural network, i.e., set width vectors to be zero as much as possible so that fewer calculations are required to reduce the computational time.

Duchi *et al*. introduced adaptive subgradient methods (AdaGrad) for online learning [13]. The geometrical knowledge of previously learned data is also included in gradient-based learning. The derived algorithms include AdaGrad-FOBOS based on the forward-backward splitting algorithm [14] and AdaGrad-RDA based on the regularized dual averaging concept [12].

Learn++ [15] is a chunk-wise incremental learning algorithm based on using an ensemble of weak classifiers and weighted majority voting. Several different classifiers can be used in Learn++, such as SVM or MLP.

The incremental support vector machine (ISVM) introduced by [16] is an incremental version of the batch learning algorithm SVM. The algorithm works by storing a limited set of samples as candidates for support vectors. Using a small candidate set can result in missing support vectors and low accuracy, while using all previously seen data as a candidate set may give the same result as batch SVM.

LASVM proposed by [17] is another online SVM algorithm. Unlike ISVM, LASVM only considers whether the current training sample is a support vector. It retains only the samples considered as support vectors but ignores the candidate set. This produces an approximate solution but also achieves faster training time.

The online sequential extreme learning machine (OS-ELM) developed by [18] can learn data incrementally in both one-by-one and chunkwise domains, with fixed or variable chunk size. The neural network structure is static and the number of hidden neurons must be predefined. No other parameters have to be chosen as the weights, biases, and other parameters of hidden neurons are randomized.

Saffari *et al*. introduced an online random forest (ORF) [19], which was adapted from the random forest algorithm combined with online bagging and extremely randomized forests. The predefined number of trees is independently set up from each other. Nodes are split when there are enough samples by using randomized test functions and thresholds. The performance is shown to converge to that of the offline version.

Incremental learning vector quantization (ILVQ) proposed by [20] is a prototype-based method searching for a set of prototypes representing the original dataset. ILVQ can incrementally learn new prototypes, automatically set the number of prototypes needed according to the data, and remove unnecessary prototypes to reduce noise. Although these methods perform rather well, the issues of memory overflow due to tremendous and dynamical increase in temporal data and the lower bound of learning time complexity have not been much involved in the development of learning algorithms for actual big data scenarios.

In addition to online learning, another core concept is known as the morphological neural network (MNN) [21]. MNN generates decision boundaries by creating hypershapes, (commonly hyperboxes) around clusters of data.

Multiple training methods for MNN, such as elimination, merging, divide-and-conquer, and evolutionary-based methods were discussed in [22].

Hernández *et al*. presented hybrid neural networks, combining MNN and a classic perceptron layer [23]. By using morphological neurons as a feature-extracting hidden layer and perceptrons as the output layer, the resulting model achieved higher accuracy while also requiring fewer learning parameters to train compared to traditional models such as MLP and SVM.

Arce *et al*. proposed the dendrite ellipsoidal neuron (DEN) training algorithm for classification problems based on dendrite morphological neural networks [24]. This method uses k-means++ [25] to find clusters and form k hyperellipsoids for each class of training data. The process starts with $k = 1$ and increases k until the classification error rate is less than a predefined constant. This ellipsoidal-neuron classifier yielded competitive results compared to traditional classifiers such as MLP, SVM, and the radial basis network (RBN).

Via *et al.* proposed a training algorithm for dendrite morphological neural networks. This algorithm uses k-medoids to cluster data into hyperboxes [26] because the k-medoids method is less sensitive to outliers compared to k-means [27]. Thus it is possible that the trained network could yield higher accuracy.

Recently, the concept of one-pass discard-after-learn was developed to deal with the memory overflow of streaming data and uncontrollable learning epochs. The time complexity of this concept is in a polynomial form with fewer neurons than other approaches. Moreover, the network possesses plasticity. The versatile elliptic basis function (VEBF) neural network [28] incrementally learns new data by creating hyperellipsoids to capture data clusters of different classes. VEBF is transposable, expandable, and rotatable according to the distribution of captured data. Junsawang *et al*. proposed the class-wise incremental learning (CIL) algorithm to improve the speed by learning one class at a time [29]. Although both methods produced higher accuracy and fewer neurons than other methods, the accuracy can be further improved. The problem of both methods is due to the steps of adjusting the size of hyperellipsoids of different classes with respect to the new incoming data during the training period. The advantages and disadvantages of the VEBF neural network and CIL algorithm can be summarized as shown in Table 1.

In this study, we aim to improve the classification accuracy of the VEBF neural networks [28], [29]. A new neural learning process concept to reduce the misclassification once detected is introduced. This concept based on the animal behavior of recoil in forms of both neuron shrinking and shifting during the learning process can handle the disadvantages of the original algorithms. Moreover, the issues of memory overflow and the lower learning time complexity in terms of incoming data are also concerned.

This paper is organized as follows. Section II summarizes the structure of VEBF network. Section III presents the proposed neuron shrinking and shifting concepts on neural learning with a recoil behavior algorithm. Section IV describes the experimental setup and presents the experimental results with a comparison to 19 other algorithms on 15 datasets of different sizes. Section V concludes the paper.

**TABLE 1. Comparison of one-pass discard-after-learn algorithms.**

| Algorithm | Advantages | Disadvantages |
|---|---|---|
| VEBF [28] | • Incrementally learns new data and discards the data after learning to reduce the memory requirement<br>• Uses a hyperellipsoidal model to create a smooth decision boundary between classes | • The order of incoming data has a significant impact on the resulting network structure<br>• No strategies to prevent the creation of overlapping neurons of different classes, leading to classification errors |
| CIL [29] | • Improves the learning speed of the VEBF algorithm by learning multiple data points of the same class at once | • Each class is learned separately, which can still result in overlapping neurons like VEBF |

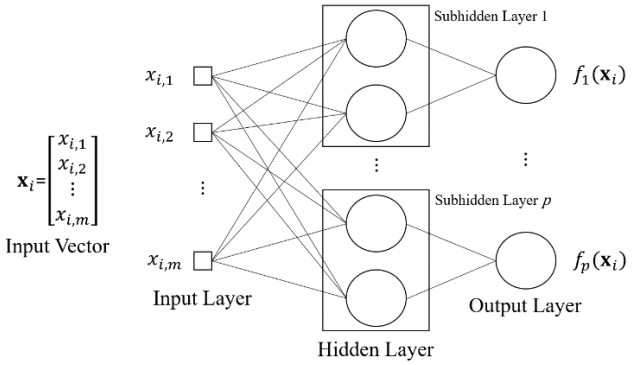


**FIGURE 1. Neural network structure based on a versatile hyperellipsoidal function.**

## II. BRIEF SUMMARY OF THE NETWORK BASED ON A VERSATILE HYPERELLIPSOIDAL STRUCTURE

The concept is to capture the incoming data of each class by using a set of hyperellipsoids. The hyperellipsoidal structure is capable of locally defining the boundary and the region of data, which is good for representing the data chunk and its distribution. It is remarkable that after capturing a cluster of data by a hyperellipsoid, the data can be completely discarded from the learning data pool. Thus, the problem of memory overflow can be efficiently solved by this concept. Furthermore, this concept can help the learning process achieve the lower bound of learning time complexity. There is no cost function involved for deriving the parameter adjusting as in the perceptron-like learning concept. All parameters are adjusted according to the distribution of present incoming data in each class and the existing hyperellipsoids.

The versatile hyperellipsoidal neural network is made up of three layers: the input layer, the hidden layer, and the output layer. Fig. 1 illustrates the structure of the network. This structure is different from the widely used feedforward network where each neuron in each layer is fully distributed to all neurons in the upper layer. The hidden layer consists of clusters of neurons represented in terms of hyperellipsoidal functions for each class. This function of hidden neuron $h$ is

denoted by $\psi_h(\mathbf{x}_i)$, where $\mathbf{x}_i$ is the input vector. At the output layer, there exists one neuron per class. The output of each class is computed by the following equation. Let $f_k(\mathbf{x}_i)$ be the output of class $k$ with respect to input vector $\mathbf{x}_i$. Assume $\boldsymbol{H}_k$ is the set of all hidden neurons in class $k$.

$$f_k(\mathbf{x}_i) = \min_{h \in \boldsymbol{H}_k} (\psi_h(\mathbf{x}_i)) \tag{1}$$

The class label assigned to input vector $\mathbf{x_i}$ is determined by the following equation.

$$C(\mathbf{x}_i) = \arg\min_k (f_k(\mathbf{x}_i)) \tag{2}$$

The hyperellipsoidal function $\psi_h(\mathbf{x}_i)$ is defined as follows.

$$\psi_h(\mathbf{x}_i) = \sum_{d=1}^{m} \frac{(\mathbf{x}_i - \mathbf{c}_h)^T \mathbf{u}_{h,d}}{w_{h,d}^2} - 1 \tag{3}$$

where $\mathbf{u}_{h,d}$ is the $d^{th}$ eigenvector computed from the covariance matrix of all vectors captured by the hyperellipsoidal function of neuron $h$, and $w_{h,d}$ is the $d^{th}$ eigenvalue of $\mathbf{u}_{h,d}$. A versatile hyperellipsoidal neuron $\Omega_h$ can be viewed as a tuple $\Omega_h = (\mathbf{c}_h, \mathbf{w}_h, \mathbf{S}_h, n_h, y_h)$, where $\mathbf{c}_h \in \mathbb{R}^m$ is the center of the neuron, an $m$-dimensional vector $\mathbf{w}_h$ specifies the width vector whose each element is the eigenvalue, a $\mathbf{S}_h$ is the covariance matrix of the captured data cluster, $n_h$ is the total number of data learned, and $y_h$ is the class label.

### A. UPDATING PARAMETERS OF HYPERELLIPSOID

Since all learned data are completely discarded, updating the center and covariance matrix of a neuron must be done in forms of recursive functions where a minimum piece of information from the previously learned and discarded data and the currently incoming datum are used as the variables of the function. To distinguish between the updated parameters and the current parameters before being updated of neuron $\Omega_h$, the following notations are employed:

- $\mathbf{c}_h$ : the updated center.
- $\mathbf{c}_h^{(cur)}$ : the center before being updated.
- $\mathbf{S}_h$ : the updated covariance matrix.
- $\mathbf{S}_h^{(cur)}$ : the covariance matrix before being updated.
- $\mathbf{x}_i$ : the $i^{th}$ incoming datum.
- $n_h$ : the number of learned data before updating $\mathbf{c}_h^{(cur)}$ and $\mathbf{S}_h^{(cur)}$.

The center and covariance matrix are updated by the following recursive functions.

$$\mathbf{c}_h = \frac{n_h \mathbf{c}_h^{(cur)} + \mathbf{x}_i}{n_h + 1} \tag{4}$$

$$\mathbf{S}_h = \frac{n_h \mathbf{S}_h^{(cur)} + \mathbf{x}_i \mathbf{x}_i^T - \mathbf{c}_h^{(cur)} \mathbf{c}_h^{(cur)^T}}{n_h + 1} - \mathbf{c}_h \mathbf{c}_h^T + \mathbf{c}_h^{(cur)} \mathbf{c}_h^{(cur)^T} \tag{5}$$

When the neuron has been updated and the number of data $n_k$ becomes more than a predetermined constant $N_0$, which by default is set to 2, the new width vector $\mathbf{w}_h = [w_{h,1}\ w_{h,2} \ldots w_{h,m}]^T$ is also updated as follows.

$$w_{h,d} = w_{h,d}^{(cur)} + \left| \left( \mathbf{c}_h - \mathbf{c}_h^{(cur)} \right)^T \mathbf{u}_{h,d} \right|; \quad d = 1, 2, \ldots, m \tag{6}$$

where $\mathbf{u}_{h,1}, \mathbf{u}_{h,2}, \ldots, \mathbf{u}_{h,m}$ are the eigenvectors of $\mathbf{S}_h$ and $\mathbf{w}_h^{(cur)}$ is the weight vector before being updated.

In the original algorithm by [28], a VEBF neuron is only updated if there exists an incoming datum of the same class falls into it. There is no step for checking if an incoming datum of another class falsely falls into the neuron. This can lead to the misclassification of the incoming datum.

### B. MERGING TWO HYPERELLIPSOIDS

To reduce the number of neurons in the network, two neurons of the same class locating close together are merged into one neuron during the learning process. Let $\Omega_a = (\mathbf{c}_a, \mathbf{w}_a, \mathbf{S}_a, n_a, y_a)$ and $\Omega_b = (\mathbf{c}_b, \mathbf{w}_b, \mathbf{S}_b, n_b, y_b)$ be two neurons of the same class. Both neurons can be merged when the following condition is met:

$$\psi_a(\mathbf{c}_b) \le \theta, \quad \text{and} \quad \psi_b(\mathbf{c}_a) \le \theta \tag{7}$$

where $\theta$ is the threshold value to determine whether two hyperellipsoids overlap each other. The default value is $\theta = 0$. After merging two hyperellipsoids into a new $\Omega_k$, the parameters $\mathbf{c}_k, \mathbf{S}_k$, each $w_{k,d}$, and $n_k$ are computed as follows.

$$\mathbf{c}_k = \frac{n_a \mathbf{c}_a + n_b \mathbf{c}_b}{n_a + n_b} \tag{8}$$

$$\mathbf{S}_k = \frac{n_a \mathbf{S}_a + n_b \mathbf{S}_b}{n_a + n_b} + \frac{n_a n_b (\mathbf{c}_a - \mathbf{c}_b)(\mathbf{c}_a - \mathbf{c}_b)^T}{(n_a + n_b)^2} \tag{9}$$

$$n_k = n_a + n_b \tag{10}$$

$$w_{k,d} = \sqrt{2\pi\,|\lambda_d|}, \quad d = 1, 2, \ldots, m \tag{11}$$

where $\lambda_d$ is the $d^{th}$ eigenvalue computed from $\mathbf{S}_k$.

### C. INITIAL WIDTH COMPUTATION

During the training process, the initial width vector $\mathbf{w}_h^{(init)} = [w_{h,1}\ w_{h,2} \ldots w_{h,m}]^T$ of $\Omega_h$ depends on the incoming data of the corresponding class. Let $\mathbf{x}_i$ and $\mathbf{x}_j$ be in the same class and also in the same incoming cluster. The width of each dimension is computed by the following equation.

$$w_{h,d} = \delta \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \|\mathbf{x}_i - \mathbf{x}_j\| \tag{12}$$

A constant $\delta$ is used to empirically adjust $\mathbf{w}_h^{(init)}$. Note that the most appropriate $\delta$ is controlled by the first distribution of incoming data and also an application dataset.

## III. PROPOSED CONCEPT AND ALGORITHM

Although the structure of the VEBF network is rather flexible to deal with a streaming data environment, the accuracy is not high enough for some datasets. The difficulty of this problem is due to the unknown probability distribution of incoming data. In this study, we make no assumption of

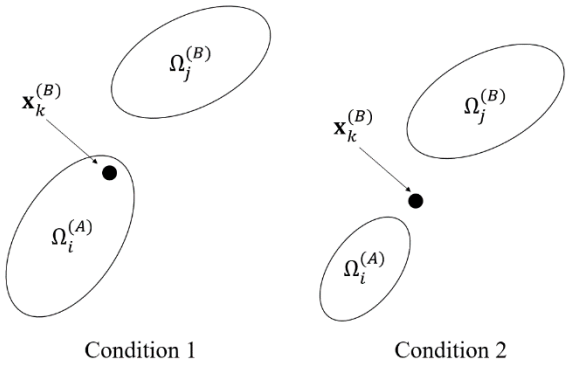


**FIGURE 2.** The cause of misclassification due to conditions 1 and 2.

this distribution. This implies that defining the proper size of each hyperellipsoid according to the incoming data is very critical. If the size of the hyperellipsoid is set too large, it may overlap with other classes. But if it tightly fits the data cluster, it can induce errors for the testing data because the data cannot fall into the hyperellipsoid.

The proposed solution is based on the animal behavior of *recoil*. Recoil is one of four animal behaviors directly related to pain and discomfort [30]. When a datum of class B falsely falls into a neuron of class A with a hyperellipsoidal structure, this neuron will be irritated by this false datum. To recuperate from the irritation, the neuron must recoil from the false datum. In terms of neuron learning, this recoil behavior is equivalent to misclassification of data. Based on this behavior, we mathematically transform this recoil behavior into mathematical equations and a learning algorithm.

Assume there are two neurons named $\Omega_i^{(A)}$ of class A and $\Omega_j^{(B)}$ of class B in forms of hyperellipsoids. Both neurons are close to each other. When an incoming datum of class B denoted by $\mathbf{x}_k^{(B)}$ enters the training process, one of two possible following conditions can lead to misclassification.

1. Condition 1: $\mathbf{x}_k^{(B)}$ falls into the region of $\Omega_i^{(A)}$ which makes
$$\psi_i^{(A)}\left(\mathbf{x}_k^{(B)}\right) \leq 0 \tag{13}$$
2. Condition 2: The distance from $\mathbf{x}_k^{(B)}$ to the center of $\Omega_i^{(A)}$ is shorter than the distance to the center of $\Omega_j^{(B)}$ and $\mathbf{x}_k^{(B)}$ is outside $\Omega_i^{(A)}$.
$$\psi_i^{(A)}\left(\mathbf{x}_k^{(B)}\right) \leq \psi_j^{(B)}\left(\mathbf{x}_k^{(B)}\right) \tag{14}$$

The meanings of both conditions is illustrated in Fig. 2. Condition 1 occurs because of the overestimation of the size of the hyperellipsoid. To solve this problem, the size of neuron $\Omega_i^{(A)}$ in all dimensions must shrink to make $\mathbf{x}_k^{(B)}$ stay outside the boundary of the hyperellipsoid. For condition 2, $\mathbf{x}_k^{(B)}$ is outside both neurons but it is closer to neuron A than neuron B. This condition occurs when the size of neuron A is smaller than that of neuron B. To solve this problem, neuron A must shift away from neuron B along the path between $\mathbf{x}_k^{(B)}$ and the center of neuron A. The details of how to shrink the hyperellipsoid and how to shift it are discussed in the next sections.

### A. SHRINKING HYPERELLIPSOID

The width of $\Omega_i^{(A)}$ in each direction is separately adjusted according to the projected distance $p_d^{(A)}$ between $\mathbf{x}_k^{(B)}$ and the center of $\Omega_i^{(A)}$ onto each eigenvector computed from its covariance matrix. Let $\mathbf{c}_i^{(A)}$ and $\mathbf{u}_{i,d}^{(A)}$ be the center and the $d^{th}$ eigenvector of $\Omega_i^{(A)}$, respectively. The projected distance onto the $d^{th}$ eigenvector is computed by (15)

$$p_d^{(A)} = \left(\mathbf{x}_k^{(B)} - \mathbf{c}_i^{(A)}\right)^T \mathbf{u}_{i,d}^{(A)} \tag{15}$$

The width of the hyperellipsoid cannot be arbitrarily adjusted because using the hyperellipsoid to capture the trained data is based on the concept of *discard-after-learn*. This implies that after capturing the data, all data are completely discarded. The quantity of data and the direction of the data distribution are represented in the form of a covariance matrix. If the width shrinks too much, then some already captured and discarded data are left outside the boundary of the hyperellipsoid and obviously, these data can be misclassified when being tested. To avoid this problem, the width must be properly adjusted by involving the quantity of captured data as one computing factor. Let $n_i$ be the quantity of captured data by $\Omega_i^{(A)}$ and $0 < \alpha < 1$ be an adjusting constant called the *shrink multiplier*. To distinguish the new width from the current width, let $w_{i,d}^{(new)}$ and $w_{i,d}^{(cur)}$ be the $d^{th}$ new width and the $d^{th}$ current width, respectively, of neuron $i$. The new width is computed as follows.

$$w_{i,d}^{(new)} = \max\left(\frac{n_i w_{i,d}^{(cur)} + \left(\mathbf{x}_k^{(B)} - \mathbf{c}_i^{(A)}\right)^T \mathbf{u}_{i,d}^{(A)}}{n_i + 1}, \alpha w_{i,d}^{(cur)}\right) \tag{16}$$

The purpose of shrink multiplier $\alpha$ in (16) is to prevent excessive shrinking when datum $\mathbf{x}_k^{(B)}$ falls close to the center of the neurons of class A, which might be the cause of noisy or outlier data. The shrinking process also considers the number of targeted data $n_i$ in the denominator of (16). The following theorem states the relation of $n_i$ and $w_{i,d}^{(cur)}$.

*Theorem 1:* When $n_i$ approaches infinity, there is no need to shrink the neuron size.

*Proof:* Let us take the limit of $n_i$ in the first term inside the *max* function of (16) as follows.

$$size = \lim_{n_i \to \infty} \frac{n_i w_{i,d}^{(cur)} + \left(\mathbf{x}_k^{(B)} - \mathbf{c}_i^{(A)}\right)^T \mathbf{u}_{i,d}^{(A)}}{n_i + 1} \tag{17}$$

$$= \lim_{n_i \to \infty} \frac{w_{i,d}^{(cur)} + \frac{\left(\mathbf{x}_k^{(B)} - \mathbf{c}_i^{(A)}\right)^T \mathbf{u}_{i,d}^{(A)}}{n_i}}{1 + \frac{1}{n_i}} \tag{18}$$

$$= w_{i,d}^{(cur)} \tag{19}$$

Since $0 < \alpha < 1$, we have $\alpha w_{i,d}^{(cur)} < w_{i,d}^{(cur)}$. Thus, the value of *size* is equal to $w_{i,d}^{(cur)}$. ■

Theorem 1 indicates that if the number of data captured by the hyperellipsoid $\Omega_i^{(A)}$ densely increases, then the region covered by $\Omega_i^{(A)}$ truly belongs to class A and any data from any other class falling into this region can be treated as discarded noisy data of the other class. Note that the shrunk neuron may still cover $\mathbf{x}_k^{(B)}$, but $\mathbf{x}_k^{(B)}$ will stay closer to the edge of the neuron and the output $\psi_i^{(A)}\left(\mathbf{x}_k^{(B)}\right)$ will become higher, so it is less likely that the other incoming data located near $\mathbf{x}_k^{(B)}$ will be classified to class A.

### B. SHIFTING HYPERELLIPSOID

After the width vector of neuron $\Omega_i^{(A)}$ is adjusted by the shrinking process, the center $\mathbf{c}_i^{(A)}$ is updated by shifting it in the opposite direction from $\mathbf{x}_k^{(B)}$ as follows. Let $\mathbf{c}_i^{(A)(cur)}$ be the current center of $\Omega_i^{(A)}$ before updating.

$$\mathbf{c}_i^{(A)} = \mathbf{c}_i^{(A)(cur)} - \frac{\mathbf{x}_k^{(B)} - \mathbf{c}_i^{(A)(cur)}}{n_i} \tag{20}$$

The center of a neuron that covers a large quantity of its actual class data will be shifted less than a neuron that covers only a few actual class data points. Note that when $n_i$ approaches infinity, there is no need to shift the center because of the same reason for data density as explained in the shifting process.

### C. NEURAL LEARNING WITH RECOIL BEHAVIOR ALGORITHM

The recoil behavior in terms of mathematical shrinking and shifting the structure of hyperellipsoid is attached to the learning process discussed in Section II. The details of neural learning with recoil behavior are shown as Algorithm 1.

After applying the shrinking and shifting processes in step 9, an incoming datum $\mathbf{x}_k^{(B)}$ might be covered by a hyperellipsoid in class B, resulting in parameter updating. However, in the case in which the datum remains outside any hyperellipsoid in class B, a new neuron is be added to learn the datum.

### D. TIME COMPLEXITY

Assume there is a total of $n$ streaming data entering the training process in $m$-dimensional space. Steps 1-6 take the time complexity of $O(nm)$ for $O(n)$ neurons in the worst-case scenario. The recoil behavior in steps 7-11 takes the time complexity of $O\left(nm^2\right)$. Computing the temporary center and temporary covariance matrix in steps 12-13 takes $O\left(m^2\right)$. Computing eigendecomposition in step 14 takes $O\left(m^3\right)$. Then, a corresponding hyperellipsoidal function is computed in step 15, which takes $O\left(m^2\right)$. There are two possibilities in steps 16-30. Creating a new neuron in step 17 or updating parameters in step 24 takes equal time complexity of $O\left(m^2\right)$, and the merging process takes time complexity of $O\left(nm^3\right)$ for at most $O(n)$ neurons. Thus, steps 16-30 take $O\left(nm^3\right)$ in total. The time complexity of learning with recoil behavior in the hyperellipsoidal structure is $O(nm)+O\left(nm^2\right)+O\left(m^2\right)+O\left(m^3\right)+O\left(m^2\right)+O\left(m^2\right)+O\left(nm^3\right)=O\left(nm^3\right)$.

**Algorithm 1** Learning With Recoil Behavior in Hyperellipsoid Structure (LRHE)

Input: (1) $\mathbf{x}_k^{(B)} \in \mathbb{R}^m$ of class B.
(2) An identity matrix $\boldsymbol{I}$ of size $m \times m$.
Output: A neuron for capturing $\mathbf{x}_k^{(B)}$.
1. **If** there is no neuron in class B **then**
2. Create a new neuron $\Omega_1^{(B)}$ such that $\mathbf{c}_1^{(B)} = \mathbf{x}_k^{(B)}$, $\mathbf{w}_1^{(cur)} = \mathbf{w}_1^{(init)}$, $S_1^{(B)} = \mathbf{I}$, and $n_1 = 1$.
3. Exit the Algorithm.
4. **Else**
5. Find a neuron $\Omega_l^{(B)}$ whose center $\mathbf{c}_l^{(B)}$ is closest to $\mathbf{x}_k^{(B)}$.
6. **EndIf**
7. **For** each neuron $\Omega_i^{(Y)}$ in other classes such that $Y \neq B$ **do**
8. **If** $\psi_i^{(Y)}\left(\mathbf{x}_k^{(B)}\right) \leq 0$ and $\psi_i^{(Y)}\left(\mathbf{x}_k^{(B)}\right) \leq \psi_j^{(B)}\left(\mathbf{x}_k^{(B)}\right)$ **then**
9. Do shrinking and shifting processes for $\Omega_i^{(Y)}$.
10. **EndIf**
11. **EndFor**
12. Compute a temporary center $\mathbf{c}_t^{(B)}$ from a neuron $\Omega_l^{(B)}$ and $\mathbf{x}_k^{(B)}$ using (4).
13. Compute a temporary covariance matrix $\mathbf{S}_t^{(B)}$ from a neuron $\Omega_l^{(B)}$ and $\mathbf{x}_k^{(B)}$ using (5).
14. Compute the eigenvalues and the corresponding eigenvectors of $\mathbf{S}_t^{(B)}$.
15. Compute a hyperellipsoidal function $\psi_t(\mathbf{x}_k)$ from $\mathbf{c}_t^{(B)}$ and $\mathbf{S}_t^{(B)}$.
16. **If** $\psi_t(\mathbf{x}_k) > 0$ **then**
17. Create a new neuron $\Omega_b^{(B)}$ such that $\mathbf{c}_b^{(B)} = \mathbf{x}_k^{(B)}$, $\mathbf{w}_b^{(cur)} = \mathbf{w}_1^{(init)}$, $S_b^{(B)} = \mathbf{I}$, and $n_b = 1$
18. **For** each other neuron $\forall j \neq b \Omega_j^{(B)} do$
19. **If** the merging condition is satisfied **then**
20. Merge $\Omega_j^{(B)}$ into $\Omega_b^{(B)}$.
21. **EndIf**
22. **EndFor**
23. **else**
24. Update parameters of $\Omega_l^{(B)}$ such that $\mathbf{c}_l^{(B)} = \mathbf{c}_t^{(B)}$, $\mathbf{S}_l^{(B)} = \mathbf{S}_t^{(B)}$, $n_l = n_l + 1$, and calculate width vector using (6).
25. **For** each other neuron $\forall j \neq l \Omega_j^{(B)} do$
26. **If** the merging condition is satisfied **then**
27. Merge $\Omega_j^{(B)}$ into $\Omega_l^{(B)}$.
28. **EndIf**
29. **EndFor**
30. **Endif**

## IV. EXPERIMENTS

There are 15 experimental datasets summarized in Table 2. Datasets 1 to 12 were obtained from the University of California at Irvine's Machine Learning Repository [31], and datasets 13 to 15 were obtained from public datasets on OpenML's repository [32]. Samples with missing values were removed from all datasets.

The classification accuracy, defined as the number of correctly classified test data divided by the total number

**TABLE 2. Details of datasets used in the experiment.**

| No | Dataset | Number of Classes | Number of Features | Number of Instances |
|---|---|---|---|---|
| 1 | Iris | 3 | 4 | 150 |
| 2 | Liver | 2 | 6 | 345 |
| 3 | Heart | 2 | 13 | 270 |
| 4 | Glass | 6 | 9 | 214 |
| 5 | E. Coli | 8 | 7 | 336 |
| 6 | Yeast | 10 | 8 | 1,484 |
| 7 | Sonar | 2 | 60 | 208 |
| 8 | Ionosphere | 2 | 34 | 351 |
| 9 | Musk V1 | 2 | 166 | 476 |
| 10 | Anuran Calls | 10 | 22 | 7,195 |
| 11 | Letter Recognition | 26 | 16 | 20,000 |
| 12 | MiniBooNE | 2 | 50 | 130,064 |
| 13 | BNG-Glass | 7 | 9 | 137,781 |
| 14 | Cod-RNA | 2 | 8 | 487,867 |
| 15 | Higgs | 2 | 28 | 98,049 |

**TABLE 3. Online learning algorithms from LIBSOL.**

| Algorithm | Description | Reference |
|---|---|---|
| P | Perceptron | [7] |
| OGD | Online Gradient Descent | [5] |
| PA | Passive-Aggressive Algorithms | [8] |
| ALMA | Approximate Large Margin Algorithm | [4] |
| RDA | Regularized Dual Averaging | [12] |
| SOP | Second-Order Perceptron | [6] |
| CW | Confidence-Weighted Learning | [9] |
| ECCW | Exactly Convex Confidence-Weighted Learning | [10] |
| AROW | Adaptive Regularization of Weight Vectors | [11] |
| Ada-FOBOS | Adaptive Gradient Descent | [13] |
| Ada-RDA | Adaptive Regularized Dual Averaging | [13] |
| ERDA | Enhanced Regularized Dual Averaging | [12] |

of test data, of the proposed LRHE on each dataset was compared to the results from the following online learning algorithms:

- The original VEBF algorithm [28].
- The class-wise incremental learning (CIL) algorithm, which is also based on the concept of the VEBF neural network, which can be trained by feeding a group of data at once instead of learning each datum one-by-one [29].
- Various online learning algorithms in LIBSOL: Library for Scalable Online Learning implementation [33]. The list of algorithms is shown in Table 3.

In addition to the online learning algorithms, five batch learning algorithms were also included in the result comparison for benchmarking purposes. These algorithms require the whole training set to be used all at once during the training process, unlike online learning where the neural network or classifier learns new datum one-by-one. The included algorithms are:

- Multilayer perceptron (MLP) with one hidden layer. The MLP network needs to be trained by the whole training set multiple times. TensorFlow [34] was used for MLP implementation.
- Support vector machine (SVM) with linear kernel, using scikit-learn's Python implementation [35].
- Probabilistic neural network with radial basis function (RBF-PNN), using NeuPy [36], a Python library for neural networks.
- Dendrite morphological neural network (DMNN) using hyperellipsoidal neurons model. Two batch learning algorithms are DMN1 using k-means++ clustering based on [24] and DMN2 using k-medoids clustering, as suggested by [26].

**TABLE 4. Choices of parameter settings for the online learning algorithms in LIBSOL.**

| Algorithm | Hyperparameters |
|---|---|
| OGD | Learning rate: $\eta \in \{0.0625, 0.125, 0.25, \ldots, 128\}$ |
| ALMA | Final margin parameter: $\alpha \in \{0.1, 0.2, 0.3, \ldots, 1\}$ |
| SOP | Parameter for positive-definite normalization matrix: $\alpha \in \{0.0625, 0.125, 0.25, \ldots, 16\}$ |
| CW and ECCW | Initial confidence: $\alpha \in \{0.0625, 0.125, 0.25, \ldots, 1\}$<br>Threshold of inverse normal distribution: $\phi \in \{0, 0.25, 0.5, \ldots, 2\}$ |
| AROW | Passive-aggressive update tradeoff parameters: $r \in \{0.0625, 0.125, 0.25, \ldots, 16\}$ |
| Ada-FOBOS and Ada-RDA | Learning rate: $\eta \in \{0.0625, 0.125, 0.25, \ldots, 128\}$<br>Parameter to ensure positive-definite property of the adaptive weighting matrix: $\delta \in \{0.0625, 0.125, 0.25, \ldots, 16\}$ |

### A. EXPERIMENTAL SETUP

Stratified 5-fold cross validation was used to evaluate the performance of learning algorithms. For online learning algorithms, the training and testing process is run ten times with training data shuffled differently and the best classification accuracy is used to represent that fold. For MLP, the initial weights are randomized 10 times in each fold. The number of neurons in the hidden layer of MLP is set to the same number of neurons used in our proposed method. For SVM and the online learning algorithms in LIBSOL, which are binary classifiers, the multiclass datasets are handled according to the one-vs-the-rest scheme by using $m$ classifiers for $m$-class dataset. The $i^{th}$ classifier only needs to learn to separate between class $i$ and the other classes. The results from all $m$ classifiers are evaluated and the one that gives the highest confidence value is used for assigning the class label.

The best parameters for LIBSOL algorithms were selected by conducting cross validation on the training set using the choices of parameters listed in Table 4. Algorithms that are not listed in the table use the default parameters in LIBSOL implementation.

Originally, the DEN algorithm as proposed by [24] requires an error threshold parameter, which is the maximum allowed classification error when attempting to learn one class using $k$ neurons. The number of neurons, $k$, of that class is increased until the error on the training set is lower than the threshold. In this experiment, the maximum number of neurons is given to DMN1 and DMN2 algorithms instead to speed up the learning process for large datasets and to create networks

**TABLE 5. Choices of parameter settings for the LRHE, CIL, and VEBF algorithms.**

| Dataset | LRHE | | CIL | VEBF |
|---|---|---|---|---|
| | $\delta$ | $\alpha$ | $\delta$ | $\delta$ |
| Iris | 0.9 | 0.6 | 0.7 | 0.33 |
| Liver | 1.2 | 0.8 | 0.7 | 1 |
| Heart | 3 | 0.8 | 0.85 | 1 |
| Glass | 1 | 0.99 | 0.6 | 1 |
| E. Coli | 1.1 | 0.99 | 0.4 | 1 |
| Yeast | 1.1 | 0.8 | 0.4 | 1 |
| Sonar | 1.5 | 0.99 | 0.5 | 1 |
| Ionosphere | 1 | 0.99 | 0.4 | 1 |
| Musk V1 | 1 | 0.99 | 0.6 | 1 |
| Anuran Calls | 4 | 0.5 | 0.7 | 2 |
| Letter Recognition | 2 | 0.5 | 0.7 | 1 |
| MiniBooNE | 3 | 0.6 | 1 | 2 |
| BNG-Glass | 3 | 0.5 | 0.7 | 2 |
| Cod-RNA | 4 | 0.4 | 0.4 | 2 |
| Higgs | 3 | 0.99 | 0.7 | 2 |

with similar complexity to LRHE but trained in batch fashion. In each dataset, the maximum number of neurons is set to twice the average number of neurons created by the LRHE algorithm.

For the proposed LRHE, CIL, and VEBF algorithms, the following parameter settings in Table 5 are used. The threshold value $N_0$ for adjusting the width in VEBF is set to 2 for all datasets. For CIL, $N_0$ is set to 3 for all datasets.

The initial width of neurons for all three algorithms is calculated by using the average pairwise distance of training data as in (12). To achieve faster computational time, if the data size exceeds 5,000 then only a subset of randomly selected 5,000 training samples is used. Otherwise, all of the training data are used instead.

### B. ACCURACY

The accuracy results with standard deviation (denoted by pm number) from a 5-fold cross validation on the selected datasets are shown in Table 6. For each dataset, the highest mean accuracy value is highlighted in bold. The independent t-test of difference, which is a statistical method for determining whether the mean difference of two groups is considered statistically significant, is used to compare whether the results of other algorithms are significantly worse than the best result for that dataset. The underlined values in the table show that there is no statistically significant difference ($p < 0.05$) between that accuracy result and the highest accuracy value for that dataset, i.e., the result in that cell is considered to be as good as the highest accuracy result.

From Table 6, it can be seen that the proposed LRHE algorithm yielded the highest classification accuracy on seven datasets: Iris, Liver, Glass, E. Coli, Sonar, Ionosphere, and BNG-Glass. In four datasets, namely Heart, Yeast, Musk V1, and Anuran Calls, the result of LRHE was not statistically different from the best result given by AROW, SVM, RBF-PNN, and CIL, respectively.

In the Letter Recognition, MiniBooNE, Cod-RNA, and Higgs datasets where the proposed algorithm yielded higher accuracy than the original VEBF algorithm, it still performed significantly worse than the algorithms providing the highest accuracy in those datasets, with the difference of 2.22%, 2.24%, 1.13% and 2.56% respectively.

### C. PERFORMANCE ON IMBALANCED DATASETS

The experimental results of the LRHE algorithm on the Yeast and Anuran Calls datasets, which are two of the imbalanced datasets used in the experiment, are compared to those of the original VEBF algorithm and CIL algorithm using four metrics including precision, recall (sensitivity), specificity, and F1 score, which are better for judging the classification performance than using accuracy alone. The results are shown in Table 7 and Table 8. The best values of the three algorithms are highlighted in bold. The two bottom rows are the average values of the metrics weighted by class size and the unweighted average values.

In the Yeast dataset, which contains 10 classes, original VEBF yielded better recalls and F1 scores for most classes, while CIL yielded better precision and specificity values in most classes. CIL yielded three out of the four best average values of the metrics in both weighted and unweighted cases. LRHE yielded the best specificity averages. In the weighted case, LRHE yielded better precision, recall, and F1 score than the original VEBF but still slightly lower than CIL. For this dataset, if all classes were considered equally important, the performance of LRHE was not as good as the other two algorithms as the unweighted average metrics have lower values.

In the Anuran Calls dataset, which also contains 10 classes, LRHE yielded better precisions, specificities, and F1 scores in more classes than VEBF and CIL. The weighted average values of the four metrics were also higher than those of VEBF and CIL. In the unweighted average case, LRHE and VEBF yielded equal specificity and F1 score, but LRHE had a higher precision, while VEBF had a higher recall. CIL algorithm performed worse than the other two algorithms in this dataset.

### D. NUMBER OF NEURONS

The resulting average number of hidden neurons for LRHE, CIL, and VEBF algorithms in each dataset is shown in Table 9. The number of hidden neurons for the MLP algorithm, which is also included in Table 9, was set to be a number between the lowest number of neurons of the three other algorithms to more than $4C$, where $C$ is the number of classes in each dataset. The test was performed, starting from the least number of hidden neurons and the number of neurons was increased until the accuracy became worse. For DMN1 and DMN2 algorithms, the number of hidden neurons yielding the highest accuracy among 5-fold cross validation was also given.

From Table 6 and Table 9, it can be seen that the proposed LRHE algorithm yielded higher classification accuracy than

**TABLE 6.** Comparison of accuracy results of different methods and their standard deviations.

| Dataset | VEBF-Based Algorithms | | | Batch Learning Algorithms | |
|---|---|---|---|---|---|
| | LRHE | CIL | VEBF | DMN1 | DMN2 |
| Iris | **98.67 ± 1.83** | 96.67 ± 2.36 | **98.67 ± 1.83** | 96.67 ± 2.36 | 96.67 ± 2.36 |
| Liver | **72.75 ± 5.36** | 71.30 ± 5.65 | 71.01 ± 5.12 | 69.28 ± 7.13 | 69.28 ± 7.13 |
| Heart | 82.96 ± 7.57 | 78.89 ± 10.28 | 74.81 ± 12.60 | 79.63 ± 8.18 | 79.63 ± 8.18 |
| Glass | **70.14 ± 2.00** | 62.20 ± 6.67 | 68.16 ± 4.55 | 59.80 ± 2.76 | 59.80 ± 2.76 |
| E. Coli | **88.74 ± 1.87** | 87.86 ± 3.02 | 86.39 ± 3.04 | 82.73 ± 2.52 | 82.14 ± 1.83 |
| Yeast | 55.81 ± 4.26 | 56.68 ± 3.36 | 50.55 ± 2.98 | 49.33 ± 4.04 | 49.94 ± 3.20 |
| Sonar | **86.05 ± 3.21** | 77.50 ± 6.49 | 82.67 ± 4.11 | 72.09 ± 6.57 | 72.09 ± 6.57 |
| Ionosphere | **92.02 ± 4.37** | 89.17 ± 2.20 | 91.73 ± 3.71 | 69.51 ± 6.91 | 69.51 ± 3.15 |
| Musk V1 | 85.74 ± 4.56 | 74.58 ± 4.07 | 83.00 ± 2.49 | 79.62 ± 4.77 | 74.17 ± 5.10 |
| Anuran Calls | 96.79 ± 0.34 | **96.87 ± 0.40** | 96.14 ± 0.57 | 96.61 ± 0.33 | 96.61 ± 0.33 |
| Letter Recognition | 87.76 ± 0.29 | 87.81 ± 0.31 | 79.60 ± 0.54 | **89.98 ± 0.26** | 88.67 ± 0.67 |
| MiniBooNE | 87.33 ± 0.75 | 88.42 ± 0.23 | 86.69 ± 2.07 | 88.94 ± 0.21 | 88.92 ± 0.20 |
| BNG-Glass | **57.88 ± 0.07** | 47.26 ± 1.40 | 52.52 ± 0.27 | 39.82 ± 0.81 | 39.80 ± 0.85 |
| Cod-RNA | 94.64 ± 0.08 | 94.90 ± 0.19 | 93.81 ± 0.14 | **95.77 ± 0.10** | 95.41 ± 0.19 |
| Higgs | 61.48 ± 0.30 | 61.74 ± 0.28 | 61.45 ± 0.31 | 61.45 ± 0.31 | 61.35 ± 0.24 |

| Dataset | Batch Learning Algorithms | | | Online Learning Algorithms | |
|---|---|---|---|---|---|
| | MLP | RBF-PNN | SVM | P | OGD |
| Iris | 96.67 ± 4.08 | 96.00 ± 1.49 | 95.33 ± 4.47 | 93.10 ± 8.99 | 77.93 ± 13.90 |
| Liver | 60.87 ± 4.91 | 62.32 ± 5.98 | 60.29 ± 2.20 | 66.47 ± 6.80 | 64.41 ± 2.85 |
| Heart | 58.15 ± 3.61 | 61.11 ± 5.07 | 73.70 ± 6.20 | 69.06 ± 3.50 | 69.06 ± 3.07 |
| Glass | 46.55 ± 10.10 | 28.27 ± 7.27 | 47.24 ± 2.63 | 35.95 ± 1.51 | 47.82 ± 2.34 |
| E. Coli | 72.00 ± 7.73 | 80.72 ± 3.28 | 86.99 ± 3.56 | 76.30 ± 7.58 | 77.41 ± 4.57 |
| Yeast | 52.22 ± 3.47 | 50.08 ± 3.54 | **58.15 ± 3.12** | 49.08 ± 3.85 | 53.95 ± 2.91 |
| Sonar | 71.26 ± 7.57 | 81.31 ± 3.93 | 77.27 ± 11.72 | 75.38 ± 2.96 | 76.28 ± 4.37 |
| Ionosphere | 84.61 ± 2.58 | 87.46 ± 4.35 | 89.47 ± 5.27 | 85.84 ± 2.09 | 86.43 ± 2.28 |
| Musk V1 | 53.18 ± 4.48 | **86.15 ± 3.10** | 83.38 ± 3.98 | 77.50 ± 2.40 | 78.78 ± 2.83 |
| Anuran Calls | 89.65 ± 0.49 | 96.01 ± 0.95 | 95.25 ± 0.72 | 94.33 ± 0.70 | 95.48 ± 0.62 |
| Letter Recognition | 75.21 ± 0.77 | 82.85 ± 0.55 | 56.12 ± 6.54 | 58.73 ± 2.65 | 67.74 ± 0.94 |
| MiniBooNE | 86.85 ± 1.20 | 82.81 ± 2.61 | 86.90 ± 2.64 | 83.49 ± 0.44 | 84.63 ± 0.66 |
| BNG-Glass | 41.80 ± 8.87 | 45.89 ± 0.43 | 38.88 ± 9.02 | 48.39 ± 2.41 | 49.11 ± 2.22 |
| Cod-RNA | 95.41 ± 0.10 | 94.03 ± 0.07 | 95.17 ± 0.10 | 94.79 ± 0.06 | 94.97 ± 0.04 |
| Higgs | 61.96 ± 0.85 | 56.64 ± 0.23 | **64.04 ± 0.12** | 58.99 ± 0.67 | 62.50 ± 0.27 |

| Dataset | Online Learning Algorithms (cont.) | | | | |
|---|---|---|---|---|---|
| | PA | ALMA | RDA | SOP | CW |
| Iris | 65.52 ± 0.00 | 79.31 ± 7.23 | 69.66 ± 8.28 | 94.48 ± 3.52 | 95.17 ± 2.76 |
| Liver | 60.00 ± 1.44 | 57.35 ± 0.00 | 67.06 ± 4.80 | 70.88 ± 5.13 | 61.76 ± 4.92 |
| Heart | 63.02 ± 4.86 | 56.60 ± 2.92 | 68.68 ± 4.40 | 81.51 ± 5.65 | 79.62 ± 4.68 |
| Glass | 35.95 ± 1.51 | 37.51 ± 5.83 | 35.95 ± 1.51 | 54.95 ± 2.89 | 56.50 ± 3.44 |
| E. Coli | 79.57 ± 4.20 | 77.44 ± 4.24 | 72.68 ± 4.99 | 76.58 ± 6.52 | 82.09 ± 6.40 |
| Yeast | 45.25 ± 4.09 | 55.50 ± 1.53 | 50.58 ± 2.71 | 49.01 ± 4.93 | 48.61 ± 8.98 |
| Sonar | 72.46 ± 4.74 | 75.35 ± 5.68 | 60.10 ± 5.84 | 79.30 ± 2.58 | 80.28 ± 2.84 |
| Ionosphere | 85.85 ± 3.28 | 84.69 ± 2.92 | 71.40 ± 2.00 | 85.28 ± 4.06 | 79.19 ± 8.34 |
| Musk V1 | 81.53 ± 3.15 | 62.03 ± 5.44 | 79.00 ± 2.99 | 78.35 ± 1.86 | 75.59 ± 2.88 |
| Anuran Calls | 95.01 ± 0.80 | 95.75 ± 0.64 | 58.37 ± 0.43 | 95.41 ± 0.83 | 95.86 ± 0.46 |
| Letter Recognition | 60.09 ± 1.62 | 61.89 ± 1.47 | 57.03 ± 2.43 | 59.70 ± 2.06 | 67.08 ± 1.01 |
| MiniBooNE | 84.31 ± 0.73 | 76.22 ± 0.08 | 83.99 ± 0.28 | 87.77 ± 0.39 | 72.07 ± 0.27 |
| BNG-Glass | 48.61 ± 3.44 | 50.06 ± 1.22 | 51.24 ± 0.55 | 49.47 ± 1.61 | 41.11 ± 4.94 |
| Cod-RNA | 94.78 ± 0.16 | 94.97 ± 0.04 | 66.63 ± 0.00 | 94.77 ± 0.31 | 94.70 ± 0.09 |
| Higgs | 58.07 ± 0.86 | 63.28 ± 0.33 | 52.86 ± 0.00 | 58.67 ± 0.94 | 59.55 ± 3.44 |

| Dataset | Online Learning Algorithms (cont.) | | | | |
|---|---|---|---|---|---|
| | ECCW | AROW | Ada-FOBOS | Ada-RDA | ERDA |
| Iris | 84.83 ± 63.40 | 84.14 ± 6.40 | 97.24 ± 2.58 | 88.28 ± 11.66 | 88.28 ± 12.83 |
| Liver | 64.71 ± 7.08 | 68.53 ± 3.03 | 68.82 ± 5.29 | 70.00 ± 5.39 | 64.41 ± 4.30 |
| Heart | 80.00 ± 5.15 | **84.15 ± 7.12** | 78.49 ± 6.27 | 78.49 ± 5.01 | 69.43 ± 4.03 |
| Glass | 55.57 ± 3.20 | 58.84 ± 3.25 | 54.15 ± 3.51 | 49.74 ± 4.37 | 35.95 ± 1.51 |
| E. Coli | 80.18 ± 5.29 | 80.23 ± 6.87 | 78.70 ± 6.13 | 75.10 ± 6.05 | 74.44 ± 3.70 |
| Yeast | 49.71 ± 3.36 | 56.39 ± 1.79 | 55.04 ± 3.33 | 51.86 ± 1.57 | 52.62 ± 2.86 |
| Sonar | 79.32 ± 4.27 | 79.31 ± 3.63 | 71.91 ± 8.96 | 71.41 ± 10.97 | 74.91 ± 2.99 |
| Ionosphere | 79.77 ± 8.59 | 84.98 ± 1.86 | 84.99 ± 3.29 | 84.40 ± 2.43 | 85.57 ± 3.46 |
| Musk V1 | 73.45 ± 1.52 | 79.21 ± 2.59 | 79.63 ± 1.73 | 78.36 ± 3.65 | 79.00 ± 2.99 |
| Anuran Calls | 96.05 ± 0.45 | 95.84 ± 0.80 | 96.27 ± 0.68 | 95.68 ± 0.51 | 89.05 ± 0.44 |
| Letter Recognition | 65.54 ± 2.55 | 68.10 ± 0.86 | 67.45 ± 1.28 | 61.94 ± 0.75 | 58.63 ± 0.80 |
| MiniBooNE | 85.67 ± 1.00 | **89.57 ± 0.27** | 88.47 ± 0.74 | 85.90 ± 0.60 | 83.59 ± 0.43 |
| BNG-Glass | 42.02 ± 4.85 | 53.09 ± 0.55 | 54.43 ± 0.43 | 53.43 ± 0.61 | 49.31 ± 1.47 |
| Cod-RNA | 94.78 ± 0.11 | 80.76 ± 0.87 | 94.96 ± 0.03 | 94.96 ± 0.04 | 94.94 ± 0.05 |
| Higgs | 56.19 ± 2.82 | 61.28 ± 0.20 | 62.60 ± 0.40 | 61.84 ± 0.21 | 62.19 ± 0.36 |

**TABLE 7.** Classification results on the Yeast dataset.

| Class | Size | LRHE | | | | VEBF | | | | CIL | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Prec. | Rec. | Spec. | $F_1$ | Prec. | Rec. | Spec. | $F_1$ | Prec. | Rec. | Spec. | $F_1$ |
| 0 | 463 | **0.56** | **0.52** | 0.82 | 0.54 | 0.48 | 0.27 | **0.87** | 0.34 | 0.53 | 0.60 | 0.76 | **0.56** |
| 1 | 5 | 0.83 | **1.00** | **1.00** | 0.91 | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | 0.80 | **1.00** | 0.89 |
| 2 | 35 | 0.34 | **0.60** | 0.97 | 0.44 | 0.53 | **0.60** | **0.99** | **0.56** | **0.56** | 0.51 | **0.99** | 0.54 |
| 3 | 44 | 0.44 | 0.61 | 0.98 | 0.51 | 0.62 | **0.68** | 0.99 | **0.65** | **0.73** | 0.43 | **1.00** | 0.54 |
| 4 | 51 | 0.28 | 0.37 | **0.97** | 0.32 | **0.33** | 0.41 | **0.97** | 0.37 | 0.28 | **0.59** | 0.95 | **0.38** |
| 5 | 163 | **0.77** | 0.66 | **0.98** | 0.71 | 0.75 | **0.77** | 0.97 | **0.76** | 0.75 | 0.69 | 0.97 | 0.72 |
| 6 | 244 | 0.60 | 0.55 | **0.93** | 0.57 | 0.58 | **0.64** | 0.91 | **0.61** | **0.63** | 0.57 | **0.93** | 0.60 |
| 7 | 429 | 0.55 | 0.58 | 0.81 | **0.57** | 0.49 | **0.59** | 0.75 | 0.53 | **0.57** | 0.54 | **0.83** | 0.55 |
| 8 | 20 | 0.27 | 0.35 | 0.99 | 0.30 | 0.45 | **0.45** | 0.99 | 0.45 | **0.82** | **0.45** | **1.00** | **0.58** |
| 9 | 30 | **0.05** | 0.03 | 0.99 | 0.04 | 0.04 | **0.13** | 0.94 | **0.07** | 0.00 | 0.00 | **1.00** | 0.00 |
| Weighted Avg. | | 0.56 | 0.55 | **0.87** | 0.55 | 0.52 | 0.51 | 0.86 | 0.50 | **0.57** | **0.57** | 0.86 | **0.57** |
| Unweighted Avg. | | 0.47 | 0.53 | **0.94** | 0.49 | 0.53 | **0.55** | **0.94** | 0.53 | **0.59** | 0.52 | **0.94** | **0.54** |

**TABLE 8.** Classification results on the Anuran Calls dataset.

| Class | Size | LRHE | | | | VEBF | | | | CIL | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Prec. | Rec. | Spec. | $F_1$ | Prec. | Rec. | Spec. | $F_1$ | Prec. | Rec. | Spec. | $F_1$ |
| 0 | 672 | **0.96** | **0.97** | **1.00** | **0.96** | 0.92 | 0.94 | 0.99 | 0.93 | 0.70 | 0.95 | 0.96 | 0.80 |
| 1 | 3478 | **1.00** | 0.99 | **1.00** | **1.00** | 0.99 | **1.00** | 0.99 | 0.99 | 0.98 | **1.00** | 0.98 | 0.99 |
| 2 | 542 | **0.99** | **0.94** | **1.00** | **0.96** | 0.96 | 0.92 | **1.00** | 0.94 | 0.93 | 0.05 | **1.00** | 0.09 |
| 3 | 310 | **0.88** | **0.99** | **0.99** | **0.93** | 0.84 | 0.83 | **0.99** | 0.83 | 0.41 | 0.65 | 0.96 | 0.50 |
| 4 | 472 | 0.79 | **1.00** | 0.98 | 0.88 | **0.96** | 0.95 | **1.00** | **0.95** | 0.93 | 0.92 | **1.00** | 0.93 |
| 5 | 1121 | **0.99** | 0.94 | **1.00** | **0.97** | **0.99** | **0.96** | **1.00** | **0.97** | 0.90 | 0.95 | 0.98 | 0.93 |
| 6 | 270 | **0.98** | 0.91 | **1.00** | **0.95** | 0.93 | **0.92** | **1.00** | 0.93 | 0.94 | 0.90 | **1.00** | 0.92 |
| 7 | 114 | **0.85** | 0.82 | **1.00** | **0.84** | 0.68 | **0.85** | 0.99 | 0.75 | 0.55 | 0.71 | 0.99 | 0.62 |
| 8 | 68 | **1.00** | 0.51 | **1.00** | 0.68 | 0.86 | **0.87** | **1.00** | **0.86** | 0.56 | 0.84 | 0.99 | 0.67 |
| 9 | 148 | **0.99** | **0.95** | **1.00** | **0.97** | 0.94 | 0.92 | **1.00** | 0.93 | **0.99** | 0.45 | **1.00** | 0.62 |
| Weighted Avg. | | **0.97** | **0.97** | **1.00** | **0.97** | 0.96 | 0.96 | 0.99 | 0.96 | 0.90 | 0.87 | 0.98 | 0.85 |
| Unweighted Avg. | | **0.94** | 0.90 | **1.00** | **0.91** | 0.91 | **0.92** | **1.00** | **0.91** | 0.79 | 0.74 | 0.99 | 0.71 |

**TABLE 9.** Average number of hidden neurons.

| Dataset | LRHE | CIL | VEBF | MLP (Predetermined) | DMN1 | DMN2 |
|---|---|---|---|---|---|---|
| Iris | 3 | 3 | 3.8 | 5 | 3 | 3 |
| Liver | 6.6 | 6.2 | 5.2 | 7 | 2 | 2 |
| Heart | 2 | 3.6 | 3 | 4 | 2 | 2 |
| Glass | 9.6 | 15 | 10.4 | 12 | 6 | 6 |
| E. Coli | 8 | 15.8 | 8 | 8 | 15.2 | 8 |
| Yeast | 40.8 | 54.6 | 18.2 | 24 | 10 | 30 |
| Sonar | 2 | 4 | 2 | 2 | 2 | 2 |
| Ionosphere | 2 | 42.8 | 2 | 4 | 4 | 4 |
| Musk V1 | 2 | 3.6 | 2 | 3 | 4 | 4 |
| Anuran Calls | 10 | 17.4 | 10 | 10 | 10 | 10 |
| Letter Recognition | 26 | 30.2 | 26 | 26 | 52 | 52 |
| MiniBooNE | 8.6 | 13.8 | 8 | 8 | 4 | 4 |
| BNG-Glass | 7 | 12.6 | 7 | 7 | 7 | 7 |
| Cod-RNA | 7.8 | 14 | 2 | 8 | 16 | 14 |
| Higgs | 2 | 10 | 2 | 4 | 2 | 2 |

VEBF and CIL and also used no greater number of neurons in 8 datasets: Iris, Heart, Glass, E. Coli, Sonar, Ionosphere, Musk V1, and BNG-Glass. In the Liver dataset, LRHE created more hidden neurons than both VEBF and CIL methods but provided higher accuracy. In the Yeast, MiniBooNE, and Cod-RNA datasets, LRHE created more neurons than VEBF but less than CIL, while the test accuracy was also higher than VEBF but lower than CIL. In the Anuran Calls, Letter Recognition, and Higgs datasets, LRHE created the same number of neurons as VEBF but provided higher accuracy, while CIL used more neurons and provided slightly higher accuracy than LRHE.

MLP provided the highest accuracy in the Higgs datasets with a higher number of hidden neurons than LRHE and provided almost the highest accuracy in the Iris and Cod-RNA datasets with a higher number of neurons than LRHE and VEBF. In the Yeast dataset, LRHE used almost twice the number of MLP's hidden neurons but provided

higher accuracy. In the MiniBooNE dataset, MLP used the same number of hidden neurons as VEBF but provided slightly higher accuracy, while LRHE used slightly more neurons to increase accuracy further. In the other datasets, MLP provided lower accuracy while used an equal or higher number of neurons than LRHE.

To compare with DMN1 and DMN2 in terms of the number of hidden neurons, LRHE used no greater number of hidden neurons with higher accuracy in 9 datasets: Iris, Heart, E. Coli, Sonar, Ionosphere, Musk V1, Anuran Calls, BNG-Glass, and Higgs datasets. For the Liver, Glass, and Yeast datasets, LRHE generated more hidden neurons than DMN1 and DMN2 but yielded higher accuracy. For the Cod-RNA dataset, DMN1 provided the highest accuracy, but LRHE used the lowest number of hidden neurons. In addition, LRHE provided a lower number of hidden neurons in the Letter Recognition dataset, but its accuracy became lower whereas, in the MiniBooNE dataset, LRHE provided a higher number of hidden neurons and lower accuracy.

### E. TIME COMPLEXITY

The time complexity of the proposed LRHE algorithm is $O\left(n^2m^3\right)$ for learning the whole dataset of $n$ samples and $m$ dimensions. The original VEBF algorithm also takes $O\left(n^2m^3\right)$ time. From the analysis in Section III-D, the time complexity for shrinking and shifting processes is rather small when compared to the total time complexity. However, this shrinking and shifting processes can lengthen the training time. The increased training time is a tradeoff for the increased classification accuracy.

### F. DISCUSSION ON PARAMETER DETERMINATION

In this study, the initial width of each dimension is a product of a constant $\delta$ and the average pairwise distance of training data as in (12). The initial width becomes larger with greater average pairwise distance, resulting in a larger coverage area with respect to the data cluster. Predominantly, the constant $\delta$ is set to 1 by default. However, it is possible that $\delta$ can be increased to expand the coverage area of the hyperellipsoid, corresponding to a high number of incoming data in a large dataset. Another parameter is a shrink multiplier $\alpha$ in (16), which is responsible for preventing excessive shrinking when an incoming noisy or outlier datum falls close to the center of the hyperellipsoid. The hyperellipsoid can be shrunk to a singular point when $\alpha = 0$ and the shrinking is not allowed when $\alpha = 1$. For this reason, the multiplier can be set to 0.99 by default for gradual shrinking. In contrast to $\delta$, the multiplier $\alpha$ can be decreased to obtain the great shrinking quantity of the hyperellipsoid in a large dataset.

## V. CONCLUSION

This paper proposed an improved learning algorithm with recoil behavior to gain more accuracy than the previously proposed concept of discard-after-learn with a versatile hyperellipsoidal structure. To model the recoil behavior, a set of mathematical equations for shrinking and shifting the hyperellipsoids is introduced to reduce the misclassification rate. The proposed algorithm can learn a dataset in $O\left(n^2m^3\right)$ time, where $n$ is the number of data and $m$ is the number of attributes.

**TABLE 10.** Accuracy improvement of LRHE over VEBF algorithm.

| Datasets | LRHE Mean Acc. | VEBF Mean Acc. | Accuracy Improvement |
|---|---|---|---|
| Iris | 98.67% | 98.67% | 0.00% |
| Liver | 72.75% | 71.01% | 1.74% |
| Heart | 82.96% | 74.81% | 8.15% |
| Glass | 70.14% | 68.16% | 1.98% |
| E. Coli | 88.74% | 86.39% | 2.35% |
| Yeast | 55.81% | 50.55% | 5.26% |
| Sonar | 86.05% | 82.67% | 3.38% |
| Ionosphere | 92.02% | 91.73% | 0.29% |
| Musk V1 | 85.74% | 83.00% | 2.74% |
| Anuran Calls | 96.79% | 96.14% | 0.65% |
| Letter Recognition | 87.76% | 79.60% | 8.16% |
| MiniBooNE | 87.33% | 86.69% | 0.64% |
| BNG-Glass | 57.88% | 52.52% | 5.36% |
| Cod-RNA | 94.64% | 93.81% | 0.83% |
| Higgs | 61.48% | 61.45% | 0.03% |

From the experimental results in Table 6, the classification accuracy improvement of the proposed LRHE learning algorithm was compared to the original VEBF algorithm as shown in Table 10. Only in the Iris dataset, in which the VEBF algorithm already achieved near-perfect accuracy of 98.67%, the LRHE algorithm could not increase the accuracy further. While in the other 14 datasets, the increased accuracy ranges from a minor increase of 0.03% in the Higgs dataset up to an increase of 8.16% in the Letter Recognition dataset.

In addition to comparing with the original VEBF, the accuracy result of the LRHE algorithm was also compared to the class-wise incremental learning (CIL) algorithm, 12 other online learning algorithms, and five batch learning methods. In 15 datasets used in the experiment, LRHE provided the highest classification accuracy in seven datasets and the second-highest in three datasets.

### A. SUGGESTIONS FOR FURTHER RESEARCH

Some disadvantages of the proposed LRHE algorithm are listed below, along with some suggestions for further investigation and development.

- The learning speed for high-dimensional datasets is rather slow due to the $O\left(d^3\right)$ time complexity needed for computing basis vectors.
- The algorithm assumes all data to have complete values of all attributes, and thus not support data with missing values.
- The concept of shrinking and shifting can be applied to the chunk-incremental learning algorithm such as the CIL algorithm.
- It is possible that the shrinking and shifting equations can be modified for better results.
- By combining LRHE with more neural layers, similar to the MNN-perceptron hybrid network concept presented

by [23], it is possible that the accuracy could be further improved, although this approach will complicate the current online, discard-after-learn process of the LRHE algorithm.

## REFERENCES

[1] S. Sagiroglu and D. Sinanc, "Big data: A review," in *Proc. Int. Conf. Collaboration Technol. Syst. (CTS)*, San Diego, CA, USA, May 2013, pp. 42–47.
[2] A. Gepperth and B. Hammer, "Incremental learning algorithms and applications," in *Proc. Eur. Symp. Artif. Neural Netw., Comput. Intell. Mach. Learn.*, Bruges, Belgium, Apr. 2016, pp. 357–368.
[3] G. De Francisci Morales and A. Bifet, "SAMOA: Scalable advanced massive online analysis," *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 149–153, Jan. 2015.
[4] C. Gentile, "A new approximate maximal margin classification algorithm," *J. Mach. Learn. Res.*, vol. 2, pp. 213–242, Dec. 2001.
[5] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proc. 20th Int. Conf. Mach. Learn.*, Washington, DC, USA, Aug. 2003, pp. 928–935.
[6] N. Cesa-Bianchi, A. Conconi, and C. Gentile, "A second-order perceptron algorithm," *SIAM J. Comput.*, vol. 34, no. 3, pp. 640–668, Mar. 2005. [Online]. Available: https://dl.acm.org/doi/10.1137/S0097539703432542
[7] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychol. Rev.*, vol. 65, no. 6, pp. 386–408, 1958, doi: 10.1037/h0042519.
[8] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, "Online passive-aggressive algorithms," *J. Mach. Learn. Res.*, vol. 7, pp. 551–585, Dec. 2006.
[9] M. Dredze, K. Crammer, and F. Pereira, "Confidence-weighted linear classification," in *Proc. 25th Int. Conf. Mach. Learn. (ICML)*, Helsinki, Finland, Jul. 2008, pp. 264–271.
[10] K. Crammer, M. Dredze, and F. Pereira, "Exact convex confidence-weighted learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, Vancouver, BC, Canada, Dec. 2008, pp. 345–352.
[11] K. Crammer, A. Kulesza, and M. Dredze, "Adaptive regularization of weight vectors," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, Vancouver, BC, Canada, Dec. 2009, pp. 414–422.
[12] L. Xiao, "Dual averaging methods for regularized stochastic learning and online optimization," *J. Mach. Learn. Res.*, vol. 11, pp. 2543–2596, Oct. 2010.
[13] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, Jul. 2011. [Online]. Available: https://dl.acm.org/doi/10.5555/1953048.2021068
[14] J. C. Duchi and Y. Singer, "Efficient online and batch learning using forward backward splitting," *J. Mach. Learn. Res.*, vol. 10, pp. 2899–2934, Dec. 2009.
[15] R. Polikar, L. Upda, S. S. Upda, and V. Honavar, "Learn++: An incremental learning algorithm for supervised neural networks," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 31, no. 4, pp. 497–508, Nov. 2001.
[16] G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, Denver, CO, USA, Jan. 2000, pp. 388–394.
[17] A. Bordes, S. Ertekin, J. Weston, and L. Bottou, "Fast kernel classifiers with online and active learning," *J. Mach. Learn. Res.*, vol. 6, pp. 1579–1619, Dec. 2005. [Online]. Available: https://dl.acm.org/doi/10.5555/1046920.1194898
[18] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate online sequential learning algorithm for feedforward networks," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1411–1423, Nov. 2006.
[19] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof, "On-line random forests," in *Proc. IEEE 12th Int. Conf. Comput. Vis. Workshops (ICCV Workshops)*, Kyoto, Japan, Sep. 2009, pp. 1393–1400.
[20] Y. Xu, F. Shen, and J. Zhao, "An incremental learning vector quantization algorithm for pattern classification," *Neural Comput. Appl.*, vol. 21, no. 6, pp. 1205–1215, Sep. 2012.
[21] G. X. Ritter, L. Iancu, and G. Urcid, "Morphological perceptrons with dendritic structure," in *Proc. 12th IEEE Int. Conf. Fuzzy Syst.*, St. Louis, MO, USA, May 2003, pp. 1296–1301.
[22] H. Sossa, F. Arce, E. Zamora, and E. Guevara, "Morphological neural networks with dendritic processing for pattern classification," in *Advanced Topics on Computer Vision, Control and Robotics in Mechatronics*, O. V. Villegas, M. Nandayapa, and I. Soto, Eds. Cham, Switzerland: Springer, 2018, pp. 27–47.
[23] G. Hernández, E. Zamora, H. Sossa, G. Téllez, and F. Furlán, "Hybrid neural networks for big data classification," *Neurocomputing*, vol. 390, pp. 327–340, May 2020.
[24] F. Arce, E. Zamora, C. Fócil-Arias, and H. Sossa, "Dendrite ellipsoidal neurons based on k-means optimization," *Evol. Syst.*, vol. 10, no. 3, pp. 381–396, Sep. 2019, doi: 10.1007/s12530-018-9248-6.
[25] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proc. 18th Ann. ACM-SIAM Symp. Discrete Algorithms*, New Orleans, LA, USA, Jan. 2007, pp. 1027–1035.
[26] Y. V. Via, C. A. Putra, and R. Alit, "Training algorithm for dendrite morphological neural network using k-medoids," in *Proc. Int. Conf. Sci. Technol. (ICST)*, Bali, Indonesia, Oct. 2018, pp. 476–480, doi: 10.2991/icst-18.2018.99.
[27] P. Arora, Deepali, and S. Varshney, "Analysis of k-means and k-medoids algorithm for big data," in *Procedia Computer Science*, vol. 78, J. Abraham and V. Bhatnagar, Eds. Amsterdam, The Netherlands: Elsevier, 2016, pp. 507–512.
[28] S. Jaiyen, C. Lursinsap, and S. Phimoltares, "A very fast neural learning for classification using only new incoming datum," *IEEE Trans. Neural Netw.*, vol. 21, no. 3, pp. 381–392, Mar. 2010.
[29] P. Junsawang, S. Phimoltares, and C. Lursinsap, "A fast learning method for streaming and randomly ordered multi-class data chunks by using one-pass-throw-away class-wise learning concept," *Expert Syst. Appl.*, vol. 63, pp. 249–266, Nov. 2016.
[30] J. Taylor, L. Vinatea, R. Ozorio, R. Schuweitzer, and E. R. Andreatta, "Minimizing the effects of stress during eyestalk ablation of litopenaeus vannamei females with topical anesthetic and a coagulating agent," *Aquaculture*, vol. 233, nos. 1–4, pp. 173–179, Apr. 2004. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0044848603006616, doi: 10.1016/j.aquaculture.2003.09.034.
[31] D. Dua and C. Graff, "UCI machine learning repository," School Inf. Comput. Sci., Univ. California, Irvine, Irvine, CA, USA, 2017. [Online]. Available: https://archive.ics.uci.edu/ml/citation_policy.html
[32] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo, "OpenML: Networked science in machine learning," *ACM SIGKDD Explor. Newslett.*, vol. 15, no. 2, pp. 49–60, 2013.
[33] Y. Wu, S. C. H. Hoi, and N. Yu, "LIBSOL: A library for scalable online learning algorithms," Singapore Manage. Univ., Singapore, Tech. Rep. SMU-TR-2016-07-25, 2016.
[34] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2015, *arXiv:1603.04467*. [Online]. Available: https://arxiv.org/abs/1603.04467
[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.
[36] Y. Shevchuk. *NeuPy: Neural Networks in Python*. Accessed: Apr. 15, 2018. [Online]. Available: http://neupy.com/pages/home.html

**KANOKSILP JINDADOUNGRUT** received the B.Sc. degree (Hons.) in computer science from Chulalongkorn University, Bangkok, Thailand, in 2018. He is currently a Researcher with the Advanced Virtual and Intelligent Computing (AVIC) Research Center, Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University. His research interests include data science and machine learning.

**SUPHAKANT PHIMOLTARES** (Member, IEEE) received the B.Eng. degree (Hons.) in electrical engineering from Thammasat University, Bangkok, Thailand, in 1998, the M.Eng. degree in electrical engineering from the King Mongkut's University of Technology Thonburi, Bangkok, in 2000, and the Ph.D. degree in computer science from Chulalongkorn University, Bangkok, in 2006. He is currently an Assistant Professor in computer science with the Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University. His research interests include neural networks, machine learning, image processing, and computer vision.

**CHIDCHANOK LURSINSAP** (Member, IEEE) received the B.Eng. degree (Hons.) in computer engineering from Chulalongkorn University, Bangkok, Thailand, in 1978, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana–Champaign, Champaign, IL, USA, in 1982 and 1986, respectively. He was a Lecturer with the Department of Computer Engineering, Chulalongkorn University, in 1979. In 1986, he was a Visiting Assistant Professor with the Department of Computer Science, University of Illinois at Urbana–Champaign. From 1987 to 1996, he worked at the Center for Advanced Computer Studies, University of Louisiana at Lafayette, as an Assistant and Associate Professor. After that, he came back to Thailand to establish Ph.D. program in computer science at Chulalongkorn University, where he became a Full Professor. His major research interest includes neural learning and its applications to other science and engineering areas.