A hybrid machine learning technique for complex non-stationary classification problems

Vijaya Krishna Yalavarthi

2018

Vijaya Krishna Yalavarthi. (2018). A hybrid machine learning technique for complex non-stationary classification problems. Master's thesis, Nanyang Technological University, Singapore.

<http://hdl.handle.net/10356/73267>

<https://doi.org/10.32657/10356/73267>

A HYBRID MACHINE LEARNING TECHNIQUE FOR COMPLEX NON-STATIONARY CLASSIFICATION PROBLEMS

VIJAYA KRISHNA YALAVARTHI

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Master of Engineering

2016

ACKNOWLEDGEMENTS

As rest of the document contains the technical work done, this is the only place to express the personal gratitude. First and foremost, I would like to thank my supervisor, Prof. Meng Joo Er for his tremendous mentorship who has helped me to grow as a researcher. The whole credit goes to him as he is the foundation for the development of this thesis. A special thanks to Mr. Rajasekar Venkatesan, for providing valuable suggestions and criticisms on my work.

I take this opportunity to thank the M. Eng. Confirmation Exercise panel members for their valuable feedback and suggestions on the thesis. I am thankful to the laboratory in charge Mr. Chia Chiu Kiat for providing the required facilities in the laboratory without which this thesis would not have been possible. I would like to thank Mr. Sriram Sridhar for his valuable support in the laboratory. I am very thankful to all my friends at NTU, Singapore who make me feel like home in the university. A special thanks to all the Graduate Program Office members of School of EEE for their constant support throughout the thesis.

Last but not the least, I would like to thank my mother Smt. Yasodhara Yalavarthi, father Sri Siva Ranga Rao Yalavarthi and elder brother Rahul Yalavarthi for their love and encouragement.

Vijaya Krishna Yalavarthi

TABLE OF CONTENTS

Acknowledgements	ii
Table of Contents	iii
Abstract.....	vi
List of Figures.....	vii
List of Tables	ix
Chapter 1: Introduction	1
1.1 Background and Motivation	1
1.1.1 Classification in Machine Learning	3
1.1.1.1 Binary Classification.....	4
1.1.1.2 Multi-class Classification.....	5
1.1.2 Incremental Learning	10
1.1.2.1 Example Incremental Learning	11
1.1.2.2 Class Incremental Learning	11
1.1.2.3 Attribute Incremental Learning.....	11
1.2 Objectives	12
1.3 Problem statement.....	13
Organisation of Thesis	13

Chapter 2: Literature Review	14
2.1 Class Incremental Learning	14
2.2 Extreme Learning Machine.....	16
2.3 Online Sequential Extreme Learning Machine	19
Chapter 3: Proposed Class Incremental Learning Algorithm	24
3.1 Initialisation Phase	24
3.2 Sequential Learning Phase	25
3.3 Class Incremental Learning Phase	25
3.4 Algorithm.....	29
3.5 Summary	32
Chapter 4: Experimentation and Simulation Results.....	33
4.1 Description of Datasets	33
4.2 Selection of Network Parameters.....	35
4.3 Functionality of Proposed Algorithm	36
4.3.1 Learning one new class	36
4.3.1.1 Experiments with Waveform Dataset	37
4.3.1.2 Experiments with Image Segmentation Dataset	39
4.3.1.3 Experiments with Satellite Image Dataset	40

4.3.1.4 Experiments with PRHD Dataset.....	41
4.3.1.5 Experiment with ORHD Dataset	42
4.3.1.6 Experiment with Letter Recognition Dataset.....	43
4.3.2 Learning new class at different instances	43
4.3.3 Learning multiple new classes	46
4.3.4 Retaining the existing knowledge	48
4.4 Consistency of Proposed Algorithm	49
4.5 Comparison of Performance	49
Chapter 5: Conclusions and Future Recommendations.....	52
5.1 Conclusions.....	52
5.2 Recommendations for further research	52
Author's Publications	54
Bibliography	55

ABSTRACT

Classification problems in machine learning have a wide range of applications including but not limited to, medical imaging, drug discovery, geostatistics, biometric identification, language processing, etc. In general, machine learning algorithms used for classification work on static input data. i.e. the number of classes in the dataset usually are known a priori or remains constant. In contrast, for several real-life applications, the data are dynamic and non-stationary in nature. The number of target labels is not fixed and can increase in real time. This results in an impending need to develop new machine learning methods to address sequential learning for non-stationary data samples featuring learning parameters. In this project, a novel technique that is independent of the number of class constraints and can adapt to the introduction of new classes it will encounter is developed. The developed technique will enable the system to remodel by itself adapting to dynamic needs of non-stationary input data samples. To be more specific novel machine learning technique based on Extreme Learning Machine is developed. Application of the proposed technique on several benchmark datasets demonstrate that the proposed technique is superior in terms of accuracy and consistency.

LIST OF FIGURES

Figure 3.1: Network architecture before introduction of new class.....	28
Figure 3.2: Network architecture after introduction of new class	29
Figure 4.1: Testing accuracy curve of waveform data.....	38
Figure 4.2: Testing accuracy curve of waveform data for variable testing data.....	38
Figure 4.3: Testing accuracy curve of image segmentation data.....	39
Figure 4.4: Testing accuracy curve of satellite image data.....	40
Figure 4.5: Testing accuracy curve of PRHD data	41
Figure 4.6: Testing accuracy curve of ORHD data.....	42
Figure 4.7: Testing accuracy curve of letter recognition data	43
Figure 4.8: Testing accuracy curve for introduction of a new class at different instances for image segmentation dataset	45
Figure 4.9: Testing accuracy curve for introduction of new class at different instances for satellite image dataset.....	45
Figure 4.10: Testing accuracy curve of PRHD data for introduction of multiple new classes	47
Figure 4.11: Testing accuracy curve of ORHD data for introduction of multiple new classes	47
Figure 4.12: Overall and individual testing accuracy curves of letter recognition data	48

Figure 4.13: Consistency graph of the proposed algorithm	49
--	----

LIST OF TABLES

Table 1.1: Example of one-per-class Coding.....	6
Table 1.2: Example of distributed Coding	6
Table 4.1: Description of datasets.....	35
Table 4.2: Distribution of waveform data.....	38
Table 4.3: Distribution of image segmentation data	39
Table 4.4: Distribution of satellite image data.....	40
Table 4.5: Distribution of PRHD data	41
Table 4.6: Distribution of ORHD data.....	42
Table 4.7: Distribution of letter recognition data.....	43
Table 4.8: Distribution of dataset for introduction of new class at different instances	44
Table 4.9: Distribution of dataset for introduction of multiple new classes	46
Table 4.10: Distribution of datasets for performance comparison	50
Table 4.11: Comparison of testing accuracy of proposed algorithm, OS-ELM, CIELM and Learn++.....	50

CHAPTER 1: INTRODUCTION

In this Chapter, an introduction to machine learning and classification are given. Different types of classification methodologies and incremental learning are discussed. Subsequently, the objectives of the thesis and the problem statement are given. This Chapter ends with the organisation of the thesis.

1.1 Background and Motivation

Machine learning is a part of computer science which comes under artificial intelligence. Arthur Lee Samuel who was a pioneer in the field of Artificial Intelligence, Machine Learning and Computer Gaming defined machine learning as *“Field of study that gives computers the ability to learn without being explicitly programmed”*. The more formal definition for machine learning was given by Tom M. Mitchel as, *“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T , as measured by P , improves with experience E ”*.

Based on the availability of the data for learning, machine learning tasks are divided into three categories:

1. Supervised Learning

In supervised learning, the learning system is given with the data having inputs and outputs labelled by a teacher. The task of the system is to find a function that maps the inputs to the outputs. It can be related to a student learning under the supervision of a teacher.

2. Unsupervised Learning

In unsupervised learning, data do not have the labelled outputs for the inputs. The task of the learning system is to identify the structures hidden in the inputs by its own. It can be related to the student learning on his own without the supervision of the teacher.

3. Semi-supervised learning

Semi-supervised learning lies in between supervised and unsupervised learning. It uses both labelled and unlabelled data for training, typically a large amount of unlabeled data with small amount labelled data.

4. Reinforcement learning

In reinforcement learning, the computer program interacts with the active environment to perform a specific task, without a trainer explicitly telling how far the goal is achieved. Learning a video game by playing against the opponent is an example of reinforcement learning.

Machine learning tasks can also be categorised based on the desired output. They are

1. Classification

It is a supervised way of learning in which input sequences are grouped into different classes. The objective is to develop a model that imputes the unseen data into the appropriate classes. In classification, the number of classes is discrete and the output may fall in one or more classes.

2. Regression

Regression is a statistical process of estimating the relationships among a few dependent variables and many independent variables. It is also a supervised way of learning in which the output is continuous. It is widely used for prediction and forecasting.

3. Clustering

Clustering is the grouping of the objects in such a way that similar objects fall in the same group. It is an unsupervised way of learning as the groups are not known apriori.

1.1.1 Classification in Machine Learning

Andre et. al formally defined classification as [1], “Given a set of training examples composed of pairs $\{x_i, y_i\}$, find a function $f(x)$ that maps each attribute vector x_i to its associated class y_i , $i = 1, 2, 3, \dots, n$, where n is the total number of training examples”.

On the basis of label association, classification problems are categorised into two.

1. Single label classification

It is a common type of classification problem in which output of any instance is associated uniquely to any of the L disjoint classes. Based on the number of disjoint classes L , single-label classification can be further divided into binary classification for $L=2$ and multi-class classification for $L>2$ [2].

2. Multi-label classification

In multi-label classification the output class of an instance can be associated with more than one of the L disjoint classes. This means the input sequence may be labelled with more than one target class.

On the basis of learning paradigms, there are two types of classification problems.

1. Batch learning

In batch learning, all the input data are available a priori and the training takes place throughout the data in one go. Batch learning has limited applications as the complete data are required beforehand for training, but in general, all the data are not available at one instance before training.

2. Sequential learning

In real life situations, all the data are not available for training. Data come chunk by chunk or one by one. Sequential learning or online learning takes the data that come chunk by chunk or one by one for training. Sequential learning has very wide range of applications because it doesn't need all the data for training in advance and the training parameters can be updated whenever new data are received.

1.1.1.1 Binary Classification

In binary classification, the objective is to classify the input sequences of a given dataset into two classes given by a trainer using a classification rule. Some typical tasks which use binary classification are:

- A disease diagnosis test giving the result of positive or negative which gives the information of the existence of the diseases in the patient.
- In factories, using binary classification, a “pass or fail” test is conducted to decide if the object meets the specifications or not.
- In retrieving the information like deciding whether an article or a page should be in the search result or not.

1.1.1.2 Multi-class Classification

Unlike binary classification where the input sequences are grouped into two disjoint classes, in multi-class classification, the number of output classes is more than two.

In the literature, some algorithms can be directly used for classifying the multiple classes, whereas other are used for binary classification. However, these binary classifiers can be used to learn multiple classes by various strategies.

Extended Methods from Binary Classification

Some binary classification techniques are extended for multi-class classification problems. The techniques fall into this category are:

Neural Networks:

A Multilayer Perceptron (MLP) neural network can be used for the multiclass classification problems. An MLP neural network has one input layer, n hidden layers, and one output layer. For binary classification, there is only one node in the output layer. By extending the single node to N nodes, multi-class classification can be achieved. Based on the assignment of output to the output layer, it can be further divided into One-per-class coding and distributed coding [3].

In the one-per-class mode, each output neuron is uniquely associated with one class. Whereas in distributed coding, every class has a specific binary code and that is realisable by the output neurons. A distance measure like hamming distance can be used to find the winning class. Examples of both the classes are given in Table 1.1 and Table 1.2.

Table 1.1: Example of one-per-class Coding

CLASS OUTPUT	
1	1 0 0
2	0 1 0
3	0 0 1

Table 1.2: Example of distributed Coding

CLASS OUTPUT	
1	0000
2	0011
3	1111

Decision Trees:

A decision tree has a structure of a flowchart. It uses tree like model of decisions and their potential consequences. Based on the input feature, nodes of the decision tree are split for the maximum information gain. In a decision tree, each of the internal

nodes symbolises a “test” on an attribute, every branch corresponds to the result of the test and the leaf node relates to a class label. The two widely used algorithms for building decision trees are Classification and Regression trees [4]–[7] and ID3/C4.5 [8].

k-Nearest Neighbours:

k-Nearest Neighbours (KNN) [9] is an instance-based learning and a non-parametric method used for classification where the input comprises *k* nearest training examples in the input feature space. It uses a distance measure like Euclidean distance for comparing the test data. The output is realised by a majority vote of its neighbours, with the target is associated with the most common class among its *k* nearest neighbours. As the number of class labels does not affect the methodology of the *k*-NN algorithm, it is indifferent to both binary and multi-class classification problems.

Naive Bayes Classifier:

Naïve Bayes classifier is a simple probabilistic classifier established on the Bayes’ theorem [10]. In Naïve Bayes classifier, class labels are assigned to problem instances which correspond to vectors of feature values, where the targets are taken from a finite set. In this technique, it is assumed that the features of every target class are independent of the others’. For the classifier, a probability model is built and from the conditional probabilities given by Bayes’ theorem, the posterior probability is calculated. The assumption of the conditional independence facilitates this technique for extending to multi-class classification problems.

Support Vector Machine:

Support Vector Machine (SVM) builds a hyperplane or set of hyperplanes over the high or infinite dimensional space which can be used for the multi-class classification purpose. The hyperplane is constructed to maximise the minimum distance between the hyperplane and the nearest example [11], [12]. Initially, SVM is developed for binary classification. Later, several variants in the SVM are developed for multi-class classification problems [13]–[15]. The classification problems in the SVM can be classified into linear separable and non-linear separable problems. In the linear separable problem, the classification can be done easily by constructing a hyperplane, whereas, in non-linear separable problems, the dataset is to be converted into higher dimensions where the features can be easily classified using a hyperplane.

Decomposition to Binary Classification Methods:

The decomposition to binary classification methods can also be considered as problem transformation methods, where a multi-class classification problem is reduced to multiple binary classification problems. Several techniques have been proposed in the literature for these transformation methods to solve the multi-class classification problems[16]. These methods can be categorised as follows [17], [18].

One-versus-All Classification:

One versus All (or One-vs.-Rest) is the simplest way to transform the multi-class classification problem into multiple binary classification problems. Consider that the training dataset of multi-class classification problem has K classes in it. One-vs.-All

classification needs K binary classifiers where each classifier is used to recognise one class from the remaining $K - 1$ classes. In the testing phase, the classifier that gives the maximum output or the classifier with the highest margin is named as the classification label. This technique has a few drawbacks. As the number of classes in the data increase, the number of classifiers are to be increased which takes more memory. The training time increases with increase in the number of classifiers as every classifier needs to be trained with the data. In this technique each class is compared with the rest all classes which imbalances the training data to the ratio of $1: K - 1$.

One-versus-One Classification:

In the One-versus-One decomposition, every class is compared with the each of the other classes. The training dataset with K classes is trained with $K(k - 1)/2$ binary classifiers. In this type of classification, every class is compared with all the other classes individually and the class with maximum occurrences is considered as the target. The major disadvantage of this type of classification is usage of $K(K - 1)/2$ classes for the classification which makes the algorithm computationally expensive and consumes more memory.

Error Correcting Output Coding (ECOC):

In this approach, a code word of N bits is assigned to each of the K classes in the multi-class classification problem. Each one of the N bits of the code word corresponds to a classifier and these classifiers are trained using the training dataset. In the testing phase, the input sequence produces a code word which is compared

using the hamming distance with each of the code word of the classes. The class which gives the minimum hamming distance is the winner.

Directed Acyclic Graph SVM:

Directed Acyclic Graph SVM (DAGSVM) is developed based on the SVM method used for multi-class classification problems. In the DAGSVM, decision trees are used in the testing stage. Each node in the decision tree is a classifier. The problem of the presence of unclassifiable regions in One-vs.-All and One-vs.-One methods can be solved using this approach.

Hierarchical Classification:

In the hierarchical classification method, input data is mapped into the definite subsumptive output classes which form like a tree. Each parent node of the tree is divided into child nodes. The initial classification occurs at a low level and then combined systematically and classify at the higher level. This process is done iteratively until one class is obtained [19].

1.1.2 Incremental Learning

Incremental learning is very importing in machine learning for solving the real world problems. It is computationally very economical and takes less time to modify an existing algorithm rather than developing a new algorithm for learning new data. In incremental learning, there would be a change in the architecture of the network. There would be an increase or decrease in the network parameters. As per Zhi-Hua Zhou et. al [20], incremental learning is divided into three categories.

1.1.2.1 Example Incremental Learning

After training a learning system, new training examples are fed to it for the learning. The trained system should be modified so that it can learn the knowledge encoded in the new examples without sacrificing the previous knowledge.

1.1.2.2 Class Incremental Learning

After training a learning system, new training examples which have more number of classes are fed to it for learning. The learning system should be modified automatically so that it can gain the information from the new classes it encounters and should hold the previous knowledge.

1.1.2.3 Attribute Incremental Learning

After training a learning system, new training examples with more input attributes are fed to it for learning. There is a need to modify the learning system so that it can learn the knowledge incorporated in the new input attributes without losing the previous knowledge.

All the above-mentioned categories can be explained with an example of a smart door locking system which can recognise the members of the house using face recognition technique. The facial features of the members vary due to aging. The locking system should be able to learn the varying features of the people. This type of learning is considered as example incremental learning. Whenever a new person joined the house, the locking system should be able to recognise the new person without the need of the data of the existing residents. This type of learning can be considered as class incremental learning. The locking system may be upgraded with new sensors

like fingerprint detection or IR scanning. The locking system should be able to take the data from the new sensors able to learn. This type of learning can be considered as the attribute incremental learning.

1.2 Objectives

Till now many techniques have been evolved in sequential learning and example incremental learning in multi-class classification. The problem with present day learning algorithms is, once the algorithm is trained with the initial number of classes and input attributes, it is very difficult to learn the new classes and input attributes which it didn't encounter before. The whole algorithm is to be trained again including the previous dataset which makes it computationally expensive and time-consuming. There are many situations where the data may not be available to retrain.

For the present day technologies where the cognitive robotics are playing the vital role, this type of classification could be a limitation. So, there is a need for the development of algorithms which can learn new classes and the input attributes that are encountered during online learning without losing the previous knowledge it gained while training. The new classes come with the new information which system have not seen before. So the algorithm should not only learn the new classes but also the new information of high magnitude. Even though an incremental learning technique with all the incremental learning features are necessary for present day problems, in this thesis the development of a learning technique which has both the features of class incremental learning and example incremental learning is given.

1.3 Problem statement

The problem statement of this thesis is given as:

“Development of an incremental learning algorithm to learn the new classes without losing the existing knowledge”.

In other words, it can be given as development of a novel machine learning technique that supports example incremental learning and class incremental learning.

Organisation of Thesis

Literature review, discussion on the Extreme Learning Machine and the Online Sequential Extreme Learning Machine are discussed in Chapter 2. The proposed algorithm is discussed in Chapter 3. Chapter 4 focusses on the experimentation, simulation results and discussion. Thesis conclusions and the recommendations for the future work are given in Chapter 5. This thesis ends with the bibliography.

CHAPTER 2: LITERATURE REVIEW

In this Chapter, a review of the related work in the Class Incremental Learning, the Extreme Learning Machine and the Online Sequential Extreme Learning Machine are covered and discussed.

2.1 Class Incremental Learning

As given in the Section 1.1.2.2, a system performing class incremental learning should be able to learn new classes arrive in the data by growing the network structure. The class incremental algorithm should have the following criteria:

1. **Learning new information:** The algorithm should be able to learn the additional information that comes with the new data.
2. **No access to the original data:** The algorithm shall not access the data that was used for the initial training.
3. **Preserve acquired knowledge:** The algorithm shall not lose the knowledge of the classes that were trained before even though it does not have the access to the data that were used.
4. **Accommodate new classes:** The system shall get modified, so that it can adapt to the new classes that arrive sequentially.

The algorithm that possess these requirements would be an essential tool for the machine learning and pattern recognition. There are many learning techniques in the literature, for example incremental learning, but a few discussed class incremental

learning. The algorithms that perform class incremental learning techniques are given below:

ARTMAP of [30] was implemented by Carpenter et al., using fuzzy approach. Whenever a new unseen instance arrives, new decision clusters are generated. A vigilance parameter is set for identifying the new instances. ARTMAP is sensitive to the order in which training data is presented and the selection of vigilance parameters.

Polikar et al. introduced Learn++ of [21] for learning new classes. In this methodology, whenever new data arrive, an ensemble of classifiers is generated for learning. The final result is generated by the combination of the classifiers using weighted majority voting. But, it suffers from the outvoting problem. Voting of the initial classifiers for the unseen classes introduces lot of error in the decision making. Many variants of Learn++ are developed in the literature [22]–[26] to overcome its drawbacks and using it for other applications.

Tetsuya Hoya of [27] proposed class incremental learning using probabilistic neural networks. In this, learning new classes by the system is equivalent to adding new subnets in the network. The main drawback of this algorithm is, it is not able to meet the criteria of not using the previously used data for learning new classes.

A class incremental learning technique based on support vector machines was proposed by Zhang et. al, [28]. The class incremental learning is considered as a binary classification problem where all the previous classes are considered as one class and the new class as the other. The problem with this is, the data become unbalance which reduces the accuracy.

Raja et al. proposed Progressive Learning Technique (PLT) of [29] and Zhao et al. proposed Class Incremental Extreme Learning Machine (CIELM) of [30] for learning new classes using the OS-ELM. These research works focussed on increasing the output layer in the network. But, the number of output classes that can be learned by the system is limited by the hidden layer neurons. If the number of hidden layer neurons are fixed, the accuracy of the system after adapting new classes will be reduced. Hence there is a need to develop a methodology for increasing the hidden layer neurons and output layer neurons simultaneously.

2.2 Extreme Learning Machine

In this section, a brief description of Extreme Learning Machine which is used for batch learning is given.

Extreme Learning Machine of [31] was first introduced by Huang et. al. Since the introduction, it has been widely used in the field of machine learning for several applications [32]–[34]. It is developed from the single layer feed forward neural network (SLFN).

Consider training data of N samples represented by $\{(x_i, y_i)\}_{i=1}^N$, where x_i is the input vector and y_i is the target vector. Assume there are P neurons in the hidden layer and M neurons in the output layer. Let G be the activation function, and the output of the single layer feedforward neural network can be given as

$$f_P(x) = \sum_{j=1}^P \beta_j G(w_j \cdot x_j + b_j) = t_j, \quad j = 1, 2, 3, \dots, P \quad (2.1)$$

where $w_j = \{w_{ji}, w_{j2}, \dots, w_{jn}\}^T$ is the input weight vector connecting input neurons to the j^{th} hidden layer neuron and $b = \{b_1, b_2, \dots, b_P\}^T$ is the bias vector of the hidden layer neurons. Input weight vector and bias vector are randomly defined and $\beta_j = \{\beta_{j1}, \beta_{j2}, \dots, \beta_M\}^T$ is the output weight vector.

For the general SLFN mentioned in the Eqn. (2.1) to perform as a classifier, the network output, and the labelled data output are given to the classifier shall be equal. Hence,

$$\sum_{j=1}^N \|t_j - y_j\| = 0$$

The network output equation can be given as

$$f_P(x) = \sum_{j=1}^P \beta_j G(w_j \cdot x_j + b_j) = y_j, \quad j = 1, 2, 3, \dots, P. \quad (2.2)$$

The output weight vector can be determined using

$$\beta = H^+ Y \quad (2.3)$$

where

$$H(w_1, \dots, w_P, b_1, \dots, b_P, x_1, \dots, x_P) = \begin{bmatrix} g(w_1 \cdot x_1 + b_1) & \dots & g(w_P \cdot x_1 + b_P) \\ \vdots & & \vdots \\ g(w_1 \cdot x_N + b_1) & \dots & g(w_P \cdot x_N + b_P) \end{bmatrix}$$

and H^+ is the Moore-Penrose generalized inverse of hidden layer output matrix H

and $Y = [y_1, \dots, y_N]^T$ given by

$$H^+ = (H^T H)^{-1} H^T \quad (2.4)$$

Then

$$\beta = (H^T H)^{-1} H^T Y \quad (2.5)$$

The training process and the mathematical framework is rigorously discussed in the literature and the key results are summarized.

Lemma 1: Given a standard SLFN with N hidden nodes and an activation function $g: R \rightarrow R$ which is infinitely differentiable in any interval, for N arbitrary distinct samples (x_i, y_i) , where $x_i \in R_n$ and $y_i \in R_m$, for any w_i and b_i randomly chosen from any intervals of R_n and R , respectively, according to any continuous probability distribution, then with probability one, the hidden layer output matrix H of the SLFN is invertible and $\|H\beta - Y\| = 0$ [35].

Lemma 2: Given any small positive value $\varepsilon > 0$ and activation function $g: R \rightarrow R$ which is infinitely differentiable in any interval, there exists $P \leq N$ such that for N arbitrary distinct samples (x_i, y_i) , where $x_i \in R_n$ and $y_i \in R_m$, for any w_i and b_i randomly chosen from any intervals of R_n and R , respectively, according to any continuous probability distribution, then with probability one, $\|H_{N \times P} \beta_{P \times m} - Y_{N \times m}\| < \varepsilon$ [35].

It can be observed that the input weights w_i and hidden layer neuron bias b_i are randomly initiated in the ELM. In the ELM, the output weights β are determined in training such that the relationship $H\beta = Y$ is true.

The output weight vector β of the ELM is obtained using $\beta = H^+T$, where H^+ is the Moore-Penrose inverse of the hidden layer output matrix H .

The overall batch learning ELM algorithm with P hidden neurons can be implemented using following steps:

Step 1: Randomly initialise input weight vector w and bias vector b .

Step 2: Calculate hidden layer output matrix H .

Step 3: Determine the output weight vector using:

$$\beta = (H^T H)^{-1} H^T Y$$

In recent years, many variants in the ELM are proposed. Variety of algorithms based on the ELM for the regression problems are studied in [32], [34], [36], [37]. These algorithms can be easily adapted to the classification problems. Moreover, voting based ELM of [38] was proposed for the classification problems. The ELM based algorithms are proposed for multi-label classification [39], [40]. Several ELM based algorithms were developed with pruning, increasing and optimising the hidden layer neurons. Various ELM algorithms are proposed for variety of applications with different features and can be studied in [41]–[52]. Among all the variants, a powerful sequential learning algorithm called Online Sequential Extreme Learning Machine (OS-ELM) of [53] is developed.

2.3 Online Sequential Extreme Learning Machine

In real time situations, it might not be possible for the complete data to be available in the first instance of training [54]. Data arrive in batches and each batch may have

one or more data points [53]. Hence, sequential learning algorithms are used for training the system for sequentially arriving data.

The OS-ELM is developed by combining the ELM and the recursive least square algorithm [53]. The OS-ELM has two phases of training. They are,

Initialisation phase

In the initialisation phase, the OS-ELM works like the batch learning ELM. Consider an initial dataset with N_0 samples represented by $\{(x_i, y_i)\}_{i=1}^{N_0}$. The hidden layer output matrix for the initial training section be H_0 . Output weights connecting hidden layer to output layer β^0 is calculated using the Eqn. (2.5) and represented as

$$\begin{aligned}\beta^0 &= (H_0^T H_0)^{-1} H_0^T Y_0 \\ &= M_0^{-1} H_0^T Y_0\end{aligned}\tag{2.6}$$

where, $M_0 = H_0^T H_0$.

$$H_0 = \begin{bmatrix} g(w_1 \cdot x_1 + b_1) & \dots & g(w_P \cdot x_1 + b_P) \\ \vdots & & \vdots \\ g(w_1 \cdot x_{N_0} + b_1) & \dots & g(w_P \cdot x_{N_0} + b_P) \end{bmatrix}$$

$$Y_0 = \begin{bmatrix} y_1^T \\ y_2^T \\ \vdots \\ y_{N_0}^T \end{bmatrix}$$

Sequential learning phase

After initialisation phase, sequentially arriving batches of data is used for training.

Let us assume k^{th} batch of data represented by $\{(x_i, y_i)\}_{i=N_{k-1}+1}^{N_k}$ is given for training. The hidden layer output matrix for the k^{th} block be H_k .

The output weight vector after the k^{th} block is given as

$$\beta^k = \beta^{k-1} + M_k^{-1} H_k^T (Y_k - H_k \beta^{k-1}) \quad (2.7)$$

$$\text{where } M_k = \begin{bmatrix} H'_{k-1} \\ H_k \end{bmatrix}^T \begin{bmatrix} H'_{k-1} \\ H_k \end{bmatrix} = H'^T_{k-1} H'_{k-1} + H_k^T H_k = M_{k-1} + H_k^T H_k$$

$$M_k = M_{k-1} + H_k^T H_k \quad (2.8)$$

where H'_{k-1} is the hidden layer output matrix for all the batches of data arrived before k^{th} batch.

In Eqn. (2.7), to calculate β^k it is needed to find M_k^{-1} . To reduce the computational complexity and time, Woodbery formula of [55] is used

$$\begin{aligned} M_k^{-1} &= (M_{k-1} + H_k^T H_k)^{-1} \\ &= M_{k-1}^{-1} - M_{k-1}^{-1} H_k^T (I + H_k M_{k-1}^{-1} H_k^T)^{-1} H_k M_{k-1}^{-1} \end{aligned} \quad (2.9)$$

The overall OS-ELM algorithm for the initial batch of N_0 data having m output classes and P hidden neurons can be implemented using following steps:

Initialisation Phase:

Step 1: Initiate input weight vector w and hidden layer bias vector b randomly.

Step 2: For the initial training, determine hidden layer output matrix H_0 using

$$H_0 = \sum_{j=1}^P G(w_j \cdot x_{0j} + b_j)$$

Step 3: Determine the values of M_0 and output weight vector β_0 using

$$M_0 = H_0^T H_0$$

$$\beta_0 = M_0^{-1} H_0^T Y_0$$

Sequential Learning Phase

Step 4: For each sequentially arriving data, calculate the hidden layer output vector

H_k of the k^{th} batch.

Step 5: Calculate the output weight vector

$$\beta^k = \beta^{k-1} + M_k^{-1} H_k^T (Y_k - H_k \beta^{k-1})$$

where M_k^{-1} is

$$M_k^{-1} = M_{k-1}^{-1} - M_{k-1}^{-1} H_k^T (I + H_k M_{k-1}^{-1} H_k^T)^{-1} H_k M_{k-1}^{-1}$$

Many variants of the OS-ELM algorithm [56]–[65] are developed for a variety of applications.

The proposed algorithm is developed on the OS-ELM for learning new classes in the sequentially arriving data.

Summary

For any Class Incremental Learning algorithm the system needs to update its structure automatically when any new class arrives in the data. As many of the sequential

learning techniques are used for static data where the number of classes is fixed, there is a need to develop the incremental learning algorithms to take the dynamic changes in the data. The system shall not lose the existing knowledge of classifying the base classes after updating. As online sequential Extreme Learning Machine has the feature of selecting the neurons randomly and identifying the output using recursive least square algorithm, it is used for developing the proposed algorithm.

CHAPTER 3: PROPOSED CLASS INCREMENTAL LEARNING ALGORITHM

The proposed algorithm is presented in this Chapter. Whenever the information in sequentially arriving data changes, there might be a need to change the structure of hidden layer. One might prune, increase or keep the structure as it is based on the information arrived. If the new information have significantly less dimensions than that of the initial data, it is quite possible that the system over fits the data. Whereas if the new information high dimensional than that of the initial data, system will under fits the data [66]. To avoid these problems one shall prune, increase or keep the hidden layer neurons according to the input information.

Whenever a new class (classes) is (are) arrived in the sequentially arriving data, it is quite possible that the significant information is added to the present one. To learn them together, new nodes are to be added in the hidden layer. As the number of classes are getting increased, output layer has to be increased accordingly. Hence, it's been shown how to grow the hidden layer and output layer simultaneously for learning new classes and examples simultaneously.

3.1 Initialisation Phase

In the initialisation phase, basic network parameters required for the sequential learning are determined. Consider the initial set of training data N_0 represented by $\{(x_i, y_i)\}_{i=1}^{N_0}$, where x_i and y_i represent input and output vectors respectively. Assume, there are m classes in the output vector. As per the theory of ELM, input

weight vector w_i and the hidden layer bias vector b_i are generated randomly. The proposed algorithm works like the OS-ELM until new classes arrive. Hence for the initial training dataset, the output weight vector can be obtained using Eqn. (2.6).

$$\begin{aligned}\beta^0 &= (H_0^T H_0)^{-1} H_0^T Y_0 \\ &= M_0^{-1} H_0^T Y_0\end{aligned}$$

3.2 Sequential Learning Phase

In the sequential learning phase, system learns the new examples in the data when there is no increase in the number of output classes. For sequentially arriving data, where there is no change in the number of classes, the output weight vector is obtained using eqn (2.7) and (2.8).

$$\beta^k = \beta^{k-1} + M_k^{-1} H_k^T (Y_k - H_k \beta^{k-1})$$

where M_k^{-1} is given as

$$M_k^{-1} = M_{k-1}^{-1} - M_{k-1}^{-1} H_k^T (I + H_k M_{k-1}^{-1} H_k^T)^{-1} H_k M_{k-1}^{-1}$$

3.3 Class Incremental Learning Phase

In this phase, there will be a change in the network architecture to learn the new classes in the sequentially arriving data.

Assume Δm new classes are arrived in the $(k + 1)^{th}$ block. To adapt to the new classes, the system should update itself by growing up the output layer. Hence, the number of output neurons are increased to $m + \Delta m$ which is m^* . As the hidden layer neurons are able to learn the information in the input data, number of neurons in the

hidden layer are to be increased automatically. Let us assume ΔP nodes are added in the hidden layer. Hence the number of neurons in hidden layer is $P + \Delta P$ which is P^* . The input weight vector and the bias of the newly added hidden layer neurons are generated randomly. Now the output weight vector β^{k+1} is given as

$$\beta^{k+1} = \left(\begin{bmatrix} H'_k & X'_k \\ H_{k+1} & X_{k+1} \end{bmatrix}^T \begin{bmatrix} H'_k & X'_k \\ H_{k+1} & X_{k+1} \end{bmatrix} \right)^{-1} \begin{bmatrix} H'_k & X'_k \\ H_{k+1} & X_{k+1} \end{bmatrix}^T \begin{bmatrix} Y'_k & T'_k \\ Y_{k+1} & T_{k+1} \end{bmatrix}$$

where H'_k is the hidden layer output matrix and Y'_k is the output vector for all the k batches of data. H_{k+1} is the hidden layer output matrix of P neurons for the $(k + 1)^{th}$ batch of data. X_{k+1} is the hidden layer output matrix for the newly added ΔP neurons. Y_{k+1} is the output vector of the existing classes for $(k + 1)^{th}$ batch and T_{k+1} is the output vector for the newly introduced classes in $(k + 1)^{th}$ batch. As there were no new classes and new hidden layer neurons before the $(k + 1)^{th}$ batch, T'_k and X'_k are zero matrices.

Hence,

$$\begin{aligned} M'_{k+1} &= \begin{bmatrix} H'_k & X'_k \\ H_{k+1} & X_{k+1} \end{bmatrix}^T \begin{bmatrix} H'_k & X'_k \\ H_{k+1} & X_{k+1} \end{bmatrix} \\ &= \begin{bmatrix} H_k'^T & H_{k+1}^T \\ X_k'^T & X_{k+1}^T \end{bmatrix} \begin{bmatrix} H'_k & X'_k \\ H_{k+1} & X_{k+1} \end{bmatrix} \\ &= \begin{bmatrix} H_k'^T H'_k + H_{k+1}^T H_{k+1} & H_k'^T X'_k + H_{k+1}^T X_{k+1} \\ X_k'^T H'_k + X_{k+1}^T H_{k+1} & X_k'^T X'_k + X_{k+1}^T X_{k+1} \end{bmatrix} \\ &= \begin{bmatrix} H_k'^T H'_k + H_{k+1}^T H_{k+1} & H_{k+1}^T X_{k+1} \\ X_{k+1}^T H_{k+1} & X_{k+1}^T X_{k+1} \end{bmatrix} \\ &= \begin{bmatrix} M_{k+1} & H_{k+1}^T X_{k+1} \\ X_{k+1}^T H_{k+1} & X_{k+1}^T X_{k+1} \end{bmatrix} \end{aligned}$$

Where from Eqn. (2.9)

$$M_{k+1}^{-1} = M_k^{-1} - M_k^{-1} H_{k+1}^T (I + H_{k+1} M_k^{-1} H_{k+1}^T)^{-1} H_{k+1} M_k^{-1}$$

Inverse of M'_{k+1} can be found using the Schur compliment [67].

$$M'_{k+1}{}^{-1} = \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{bmatrix}$$

$$E_{11} = M_{k+1}^{-1} + M_{k+1}^{-1} H_{k+1}^T X_{k+1} S^{-1} X_{k+1}^T H_{k+1} M_{k+1}^{-1}$$

$$E_{12} = -M_{k+1}^{-1} H_{k+1}^T X_{k+1} S^{-1}$$

$$E_{21} = -S^{-1} X_{k+1}^T H_{k+1} M_{k+1}$$

$$E_{22} = S^{-1}$$

where S is the Schur complement of M_{k+1} .

$$S = X_{k+1}^T X_{k+1} - X_{k+1}^T H_{k+1} M_{k+1}^{-1} H_{k+1}^T X_{k+1}$$

Now,

$$\begin{aligned} & \begin{bmatrix} H_k & X_k \\ H_{k+1} & X_{k+1} \end{bmatrix}^T \begin{bmatrix} Y_k & T_k \\ Y_{k+1} & T_{k+1} \end{bmatrix} \\ &= \begin{bmatrix} H'_k & X'_k \\ H_{k+1} & X_{k+1} \end{bmatrix}^T \begin{bmatrix} Y'_k & T'_k \\ Y_{k+1} & T_{k+1} \end{bmatrix} \\ &= \begin{bmatrix} H_k^T Y'_k + H_{k+1}^T Y_{k+1} & H'_k T'_k + H_{k+1}^T T_{k+1} \\ X_k^T Y'_k + X_{k+1}^T Y_{k+1} & X'_k T'_k + X_{k+1}^T T_{k+1} \end{bmatrix} \\ &= \begin{bmatrix} H_k^T Y'_k + H_{k+1}^T Y_{k+1} & H_{k+1}^T T_{k+1} \\ X_{k+1}^T Y_{k+1} & X_{k+1}^T T_{k+1} \end{bmatrix} \\ &= \begin{bmatrix} M_{k+1} \beta^k - H_{k+1}^T H_{k+1} \beta^k + H_{k+1}^T Y_{k+1} & H_{k+1}^T T_{k+1} \\ X_{k+1}^T Y_{k+1} & X_{k+1}^T T_{k+1} \end{bmatrix} \\ &= \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix} \end{aligned} \tag{3.1}$$

$$M'_{k+1}{}^{-1} = \begin{bmatrix} M_{k+1} & H_{k+1}^T X_{k+1} \\ X_{k+1}^T H_{k+1} & X_{k+1}^T X_{k+1} \end{bmatrix}^{-1} = \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{bmatrix}$$

$$\beta^{k+1} = \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{bmatrix} \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix}$$

$$= \begin{bmatrix} E_{11}F_{11} + E_{12}F_{21} & E_{11}F_{12} + E_{12}F_{22} \\ E_{21}F_{11} + E_{22}F_{21} & E_{21}F_{12} + E_{22}F_{22} \end{bmatrix} = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Then

$$B_{11} = \beta^k + M_{k+1}^{-1} H_{k+1}^T (I + X_{k+1} S^{-1} X_{k+1}^T (H_{k+1} M_{k+1}^{-1} H_{k+1}^T - I)) \times (Y_{k+1} - H_{k+1} \beta^k)$$

$$B_{12} = M_{k+1}^{-1} H_{k+1}^T (I + X_{k+1} S^{-1} X_{k+1}^T (H_{k+1} M_{k+1}^{-1} H_{k+1}^T - I)) \times T_{k+1}$$

$$B_{21} = -S^{-1} X_{k+1}^T H_{k+1} (\beta^k + M_{k+1}^{-1} H_{k+1}^T (Y_{k+1} - H_{k+1} \beta^k)) + S^{-1} X_{k+1}^T Y_{k+1}$$

$$B_{22} = S^{-1} X_{k+1}^T (I - H_{k+1} M_{k+1}^{-1} H_{k+1}^T) \times T_{k+1}$$

Figure 3.1: Network architecture before introduction of new class

From the methodology, it can be understood that the proposed algorithm does not need to access previously used data. The updated network parameters are not taken

randomly but evolved from the original structure which means that the system is not losing the existing knowledge acquired during the initial training. Figure 3.1 and Figure 3.2 shows the sample architecture of the network before and after adding new class. There are four hidden layer neurons and two output layer neurons in the architecture shown in Figure 3.1. When new class is arrived two hidden layer neurons and output neuron are added to the system as shown in Figure 3.2.

Figure 3.2: Network architecture after introduction of new class

3.4 Algorithm

The step by step procedure for the proposed algorithm is given below:

Initialisation Phase

Step 1. Initialise input weight vector w and hidden layer bias vector b randomly.

Step 2: For the initial training block N_0 , determine hidden layer output matrix H_0 using

$$H_0 = \sum_{j=1,}^P G(w_j \cdot x_{0j} + b_j)$$

Step 3: Determine the values of M_0 and output weight vector β^0 using

$$M_0 = H_0^T H_0$$

$$\beta^0 = M_0^{-1} H_0^T Y_0$$

Sequential Learning Phase

Step 4: For each sequentially arriving dataset of batch k , calculate the hidden layer output vector H_k .

Step 5: Calculate the output weight vector using

$$\beta^k = \beta^{k-1} + M_k^{-1} H_k^T (Y_k - H_k \beta^{k-1})$$

where M_k^{-1} is given as

$$M_k^{-1} = M_{k-1}^{-1} - M_{k-1}^{-1} H_k^T (I + H_k M_{k-1}^{-1} H_k^T)^{-1} H_k M_{k-1}^{-1}$$

Incremental learning phase

Consider Δm new classes are arrived after the initial training. Let the total number of output classes be m^* . Δp hidden nodes are added to the hidden layer. Input weight vector and bias of the ΔP hidden nodes are generated randomly. Total number of hidden neurons are P^* .

Step 6: Calculate H_{k+1} which is hidden layer output matrix for P neurons, and X_{k+1} which is hidden layer output matrix for newly introduced ΔP neurons.

Step 7: Calculate M_{k+1}^{-1}

$$M_{k+1}^{-1} = M_k^{-1} - M_k^{-1} H_{k+1}^T (I + H_{k+1} M_k^{-1} H_{k+1}^T)^{-1} H_{k+1} M_k^{-1}$$

Step 8: Calculate $M'_{k+1}{}^{-1}$

$$M'_{k+1}{}^{-1} = \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{bmatrix}$$

Where

$$E_{11} = M_{k+1}^{-1} + M_{k+1}^{-1} H_{k+1}^T X_{k+1} S^{-1} X_{k+1}^T H_{k+1} M_{k+1}^{-1}$$

$$E_{12} = -M_{k+1}^{-1} H_{k+1}^T X_{k+1} S^{-1}$$

$$E_{21} = -S^{-1} X_{k+1}^T H_{k+1} M_{k+1}$$

$$E_{22} = S^{-1}$$

$$S = X_{k+1}^T X_{k+1} - X_{k+1}^T H_{k+1} M_{k+1}^{-1} H_{k+1}^T X_{k+1}$$

Step 9:

Determine β^{k+1}

$$\beta^{k+1} = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Where

$$B_{11} = \beta^k + M_{k+1}^{-1} H_{k+1}^T (I + X_{k+1} S^{-1} X_{k+1}^T (H_{k+1} M_{k+1}^{-1} H_{k+1}^T - I)) \times (Y_{k+1} - H_{k+1} \beta^k)$$

$$B_{12} = M_{k+1}^{-1} H_{k+1}^T \left(I + X_{k+1} S^{-1} X_{k+1}^T (H_{k+1} M_{k+1}^{-1} H_{k+1}^T - I) \right) \times T_{k+1}$$

$$B_{21} = -S^{-1} X_{k+1}^T H_{k+1} \left(\beta^k + M_{k+1}^{-1} H_{k+1}^T (Y_{k+1} - H_{k+1} \beta^k) \right) + S^{-1} X_{k+1}^T Y_{k+1}$$

$$B_{22} = S^{-1} X_{k+1}^T (I - H_{k+1} M_{k+1}^{-1} H_{k+1}^T) \times T_{k+1}$$

3.5 Summary

The proposed algorithm works like the OS-ELM until new classes arrive. Whenever new output classes arrive, hidden layer and output layer of the network are grown automatically. Number of neurons added in the hidden layer depends on the number of new classes added. The proposed algorithm can learn the new classes it encounters along with the additional information in the input sequences.

CHAPTER 4: EXPERIMENTATION AND SIMULATION RESULTS

The performance of the proposed algorithm is evaluated with various standard datasets. The datasets are of the Waveform, Satellite imaging, Image processing, Letter recognition, Pen-based recognition of handwritten Digits (PRHD) and Optical recognition of handwritten Digits (ORHD). All the data sets are taken from the UCI machine learning repository.

4.1 Description of Datasets

The description of datasets is given below:

1. Waveform dataset

Waveform dataset has three classes of waves where each class is originated from two or three base waves. The dataset has 21 attributes, where all of them have noise added with zero mean and unit variance. It is a balanced dataset with 5000 instances.

2. Image Segmentation dataset

Image segmentation dataset has seven classes, namely brick face, sky, foliage, cement, window, path, and grass. The dataset has 19 attributes which are continues. All the images are hand segmented to create a classification for every pixel. It is a balanced dataset with 2310 instances.

3. *Satellite Imaging dataset*

Satellite imaging dataset has six classes which are red soil, cotton crop, grey soil, damp grey soil, soil with vegetation stubble and very damp grey soil. It has 36 attributes which are numerical and in the range of 0 to 255. It is a balanced dataset with 6435 instances.

4. *Letter Recognition dataset*

In the Letter recognition dataset, the objective is to identify the letter which the input sequence belongs to. The dataset of only seven letters is considered. Hence, the dataset has seven classes, namely A, B, C, D, E, F and G. There are 16 input attributes, all are numeric and the values are in the range of 0 to 15. It is a balanced dataset with 5412 instances.

5. *Pen-Based Recognition of Handwritten Digits*

The Pen-based recognition of handwritten digits' (PRHD) database has the collection of 250 samples written by 44 different people. The original dataset has all the digits as classes. However, for the experimentation purpose, only 7 digits that are from 0 to 6 are considered. All the 16 input attributes are numeric in nature and has 5242 instances.

6. *Optical Recognition of Handwritten Digits*

The dataset of Optical recognition of handwritten digits (ORHF) contains the classes of digits from 0 to 9 taken from 43 people. There are 64 input attributes whose values ranges from 0 to 15. For the experimental purpose, only 3382 instances containing 6 classes which are 0 to 5 are considered.

Table 4.1 gives the details of the datasets used for evaluation of performance of the proposed algorithm.

Table 4.1: Description of datasets

Dataset	Attributes	Instances	No. of Classes
Waveform	21	5000	3
Image segmentation	19	2310	7
Satellite image	36	6435	6
Letter recognition	16	4639	6
PRHD	17	5242	7
ORHD	64	3382	6

4.2 Selection of Network Parameters

Every dataset mentioned in the Table 4.1 is used for evaluating the performance of the proposed algorithm. Each dataset is divided into two parts, the first part has 70% of the whole data taken randomly and is used for training, and remaining is given for testing. The training dataset is further divided into two sets where the first dataset has a few classes less than that of the second dataset. The first dataset is utilised for the initial training and the classes present in it can be considered as the base classes. Hence, it can be understood that the network does not know about the new classes it will encounter in future.

The only system parameter required before starting the proposed algorithm is the number of hidden layer neurons. It is identified by using the training and validation method. 80% of the initial training dataset is used for training and the remaining 20% is utilised for the validation. Simulation experiments are conducted 50 times and the number of neurons that give the best validation accuracy is considered for the training. It is considered that the number of neurons added to the hidden layer is linearly dependent on the number of new output classes added in the new data which means the ratio of hidden layer neurons to the output layer neurons remains constant. After training with the initial data, hidden layer neurons and output layer neurons are grown automatically, and the system will be trained for the new classes and the input examples. If the number of neurons to be added to the hidden layer obtained is non-integer, nearest integer value is considered.

The performance of the proposed algorithm is evaluated on the above-mentioned datasets. The simulation results taken for the addition of one new class at different instances and multiple new classes in the datasets are given below.

4.3 Functionality of Proposed Algorithm

The proposed algorithm is validated for the various functionalities and the simulation results are given in this section.

4.3.1 Learning one new class

Firstly, the introduction of one new class in the sequentially arriving data is shown. The initial training data have a few classes considered as base classes. After the initial training, one new class is added to the system sequentially. However, the testing data

has all the classes in it. Some experiments are done for varying testing data also which means, the testing data given for the system during initial training has only base classes. New class is added to the testing data along with the training data.

4.3.1.1 Experiments with Waveform Dataset

Figure 4.1 shows the testing accuracy curve of the waveform data. The X-axis is the sample index and Y-axis is the testing accuracy. Dataset distribution of the waveform data used for this experiment is shown in Table 4.2. Initially, the system is trained for two classes up to 705 samples. Using testing and validation method, the number of hidden layer neurons obtained is 100. A new class is added at 706th sample which makes the system to grow automatically. A node is added to the output layer and 50 nodes are added in the hidden layer. It can be observed that the testing accuracy during initial training is very less because the system is not trained for the unseen class, but it is present in the testing data. Later, testing accuracy suddenly rises and settles down after few training samples.

In real time situations, initial data may not have all the classes in it. As the testing data is taken from the initial dataset given, testing dataset cannot have all the classes in it. Hence, two sets of testing data have been taken where the first set has the same classes present in the initial training data and the second set has all the classes. The testing accuracy curve is shown in Figure 4.2. Initially testing accuracy is high and when a new class is added, it reduced suddenly. After few instances, it got settled down. This is due to the new examples in the unseen class. The system is able to learn initial classes more efficiently. As the number of classes increase system efficiency to the particular dataset decreased.

Table 4.2: Distribution of waveform data

Data range	No. of classes	Point of introduction of new class	Hidden Nodes
1–705	2	-	100
706–3500	3	706	150

Figure 4.1: Testing accuracy curve of waveform data

Figure 4.2: Testing accuracy curve of waveform data for variable testing data

The functionality of the proposed algorithm for adding one new class is checked with the other datasets mentioned.

4.3.1.2 Experiments with Image Segmentation Dataset

The distribution of the Image Segmentation data is given in Table 4.3. Figure 4.3 shows the testing accuracy curve of the Image Segmentation dataset. Six classes are trained with the initial training dataset containing 416 samples. For initial training, 150 hidden layer neurons are used. For learning the new class, system is updated with 25 more neurons in the hidden layer and one neuron in output layer automatically. The data relates to the class of grass is the new class given the system after initial training.

Table 4.3: Distribution of image segmentation data

Data range	No. of classes	Point of introduction of new class	Hidden Nodes
1–416	6	-	150
417–1617	7	417	175

Figure 4.3: Testing accuracy curve of image segmentation data

4.3.1.3 Experiments with Satellite Image Dataset

The distribution of the Satellite Image dataset is given in Table 4.4. Figure 4.4 shows the testing accuracy curve of Satellite Image data. Initially, system is trained with 1033 samples of five base classes belong to red soil, cotton crop, grey soil, damp grey soil and soil with vegetation stubble. 150 hidden neurons are used for the initial training. At 1034th sample a new class belongs to very damp grey soil is introduced and the network is updated with addition of 30 neurons to the hidden layer automatically.

Table 4.4: Distribution of satellite image data

Data range	No. of classes	Point of introduction of new class	Hidden Nodes
1–1033	5	-	150
1034–4504	6	1034	180

Figure 4.4: Testing accuracy curve of satellite image data

4.3.1.4 Experiments with PRHD Dataset

The distribution of PRHD dataset is given in Table 4.5. During initial training, the system is trained with 962 data samples of six base classes belong to ‘0’, ‘1’, ‘2’, ‘3’, ‘4’ and ‘5’. For initial training 240 hidden layer neurons are used. At 963rd sample a new class belongs ‘6’ is introduced to the system. For the incremental class learning, 40 hidden layer neurons are added automatically to the system. Figure 4.5 shows the testing accuracy curve of the PRHD data.

Table 4.5: Distribution of PRHD data

Data range	No. of classes	Point of introduction of new class	Hidden Nodes
1–962	6	-	240
963–3697	7	963	280

Figure 4.5: Testing accuracy curve of PRHD data

4.3.1.5 Experiment with ORHD Dataset

Table 4.6 gives the description of the ORHD dataset used for the simulation experiment. Initially, the system containing 250 hidden layer neurons is trained with five base classes using 811 data samples. A new class belongs to ‘6’ is added to the system at 812th sample. Along with the output neuron, 50 hidden layer neurons are added to the system. Figure 4.6 shows the testing accuracy curve of the PRHD data.

Table 4.6: Distribution of ORHD data

Data range	No. of classes	Point of introduction of new class	Hidden Nodes
1–597	5	-	250
598–2368	6	598	300

Figure 4.6: Testing accuracy curve of ORHD data

4.3.1.6 Experiment with Letter Recognition Dataset

Table 4.7: Distribution of letter recognition data

Data range	No. of classes	Point of introduction of new class	Hidden Nodes
1–811	5	-	350
812–3247	6	812	420

Figure 4.7: Testing accuracy curve of letter recognition data

The distribution of the Letter Recognition data is given in Table 4.7. Figure 4.7 shows the testing accuracy curve of Letter Recognition data.

4.3.2 Learning new class at different instances

The functionality of the proposed algorithm when the new class is added at three different instances is shown. The new class is added after the initial training for 30%, 50% and 70% of base classes' data. For understanding Image Segmentation and

Satellite Image datasets are taken for demonstration. The distribution of datasets for this experiment is shown in Table 4.8.

Table 4.8: Distribution of dataset for introduction of new class at different instances

Data set	Training data range	Point of introduction of new class
Image Segmentation	1-416 417-1617	417
	1-693 694-1617	694
	1-973 974-1617	974
Satellite Image	1-1033 1034-4504	1034
	1-1718 1719-4504	1719
	1-2423 2424-4504	2424

Figure 4.8 and Figure 4.9 show the testing accuracy graphs of the Image Segmentation and the Satellite Image datasets respectively. In the graph, the three curves show testing accuracy when the new classes are added in three different instances. From the figures, it can be understood that even though the new class is added at the very late instance the accuracy didn't change significantly.

Figure 4.8: Testing accuracy curve for introduction of a new class at different instances for image segmentation dataset

Figure 4.9: Testing accuracy curve for introduction of new class at different instances for satellite image dataset

4.3.3 Learning multiple new classes

The performance of the proposed algorithm is evaluated for the introduction of multiple new classes. For the demonstration, PRHD and ORHD data sets are used for the simultaneous introduction of new classes. The dataset distribution of PRHD and ORHD for the introduction of multiple new classes shown in Table 4.9.

Table 4.9: Distribution of dataset for introduction of multiple new classes

Dataset	Training data range	Point of introduction of new class	No. of new classes added
PRHD	1-962 963-3697	963	1
	1-806 807-3697	807	2
	1-650 651-3697	651	3
ORHD	1-597 598-2368	598	1
	1-472 473-2368	473	2
	1-354 355-2368	355	3

Figure 4.10 and Figure 4.11 show the testing accuracy curves of PRHD and ORHD respectively. Graph shows the testing accuracy when one, two and three classes are added simultaneously to the system after initial training. It can be observed that the testing accuracy did not vary significantly even after the multiple classes are added

simultaneously which shows that the proposed algorithm can learn multiple new classes.

Figure 4.10: Testing accuracy curve of PRHD data for introduction of multiple new classes

Figure 4.11: Testing accuracy curve of ORHD data for introduction of multiple new classes

4.3.4 Retaining the existing knowledge

To show that the proposed algorithm can learn new classes without affecting its performance on the previously learned classes, testing accuracy curves of the individual classes and the overall testing accuracy for the Letter Recognition data are shown in Figure 4.12.

The system is initially trained for A, B, C, D and E classes till 811 samples and later, F class is added at 812th sample. The knowledge of the system to classify the bases classes is the existing knowledge. After adding the new class or set of new classes, system will automatically adapt to the changes. After adding the new class, system was able to classify the base classes without significant loss in accuracy which means system did not lose the knowledge acquired during the initial training. When the new class is added, there are wavering in the testing accuracy of the base classes, but it was settled after a few iterations.

Figure 4.12: Overall and individual testing accuracy curves of letter recognition data

4.4 Consistency of Proposed Algorithm

Consistency is a crucial feature to be tested for any algorithm. An algorithm with high accuracy but less consistent is not useful for the practical and real time applications. The results of the algorithm over every trial shall not deviate significantly from the mean value for the algorithm to be consistent. Hence, the consistency of the proposed algorithm is verified on all the datasets mentioned. In Figure 4.13, the consistency graph of the proposed algorithm for all the mentioned datasets are shown.

Figure 4.13: Consistency graph of the proposed algorithm

4.5 Comparison of Performance

The performance of the proposed algorithm for introducing one new class is compared with the existing class incremental learning algorithms learn++ and CIELM. As mentioned, for the class incremental learning algorithm to be used successfully, its performance should be at par with the sequential learning technique. Hence, the performance of the proposed algorithm is compared with the OS-ELM

also. Distribution of the datasets for the introduction of one new class is given in Table 4.10. Mean and standard deviation of the testing accuracy for all the datasets given are compared and shown in Table 4.11. A normal lab PC with core i7 processor and 8 GB RAM is used for the simulation works.

Table 4.10: Distribution of datasets for performance comparison

Dataset	Train data Range	Test Data	No. of Classes	Hidden Nodes
Waveform	1-705	1500	2	100
	706-3500		3	150
Image Segmentation	1-416	963	6	150
	417-1617		7	175
Satellite Image	1-1033	1931	5	150
	1034-4504		6	180
Letter Recognition	1-811	1392	5	350
	812-3247		6	420
PRHD	1-953	1583	6	240
	953-3659		7	280
ORHD	1-597	1014	5	250
	598-2368		6	300

Table 4.11: Comparison of testing accuracy of proposed algorithm, OS-ELM, CIELM and Learn++

Data Type	Proposed Algorithm	OS-ELM	CIELM	Learn ++
Waveform	83.04 $\pm$ 0.87	85.30 $\pm$ 0.68	82.60 $\pm$ 0.97	61.24 $\pm$ 3.81
Image segmentation	96.73 $\pm$ 0.75	95.07 $\pm$ 0.73	96.62 $\pm$ 0.60	92.40 $\pm$ 1.28
Satellite image	87.97 $\pm$ 0.69	88.93 $\pm$ 0.57	87.54 $\pm$ 0.70	74.61 $\pm$ 1.75
Letter recognition	97.88 $\pm$ 0.43	98.50 $\pm$ 0.34	97.55 $\pm$ 0.44	86.40 $\pm$ 1.81

PRHD	99.41 ± 0.17	99.39 ± 1.10	99.34 ± 0.18	94.57 ± 1.02
ORHD	98.38 ± 0.38	98.14 ± 0.46	98.14 ± 0.46	92.26 ± 1.63

It can be observed from the simulation results that the testing accuracy of proposed algorithm is at par with the OS-ELM and better than the CIELM and the Learn++. In summary, the proposed algorithm updates itself to learn new classes and the additional information without losing the existing knowledge significantly.

CHAPTER 5: CONCLUSIONS AND FUTURE RECOMMENDATIONS

5.1 Conclusions

In this thesis, a novel machine learning technique for classification of non-stationary data is presented. The proposed algorithm learns the new output classes it encounters and the additional information created in the input sequences. The proposed algorithm gains the new knowledge without significantly losing the existing one. The proposed algorithm has the ability for both the Example Incremental Learning and the Class Incremental Learning. In the real-time applications like cognitive robotics where the number of output classes varies dynamically, the proposed algorithm is well suitable. The performance of the proposed algorithm is compared with both the sequential learning technique and the existing state of the art class incremental learning techniques. From the simulation results, it can be understood that the performance of the proposed algorithm is promising.

5.2 Recommendations for further research

In this thesis, a novel algorithm for learning new classes in the in the sequentially arriving data is shown. A detailed framework for growing the hidden layer and the output layer simultaneously is shown. The further directions of the proposed research are:

The proposed research can be extended for the development of a robust technique having all the features of example incremental, class incremental and attribute incremental learning.

As artificial intelligence and cognitive robotics are one of the most key research areas, there is a need for the development of algorithms for concept drift and the proposed methodology can be extended for these requirements.

In the proposed methodology, number of hidden layer neurons to be added is linearly proportional to the number of output layer neurons (No. of classes). A research study on identifying the optimal number of neurons when new classes are added would be very useful.

The proposed methodology is useful for the multi-class classification and it can be extended to multi-label classification where the number of disjoint classes for one input sequence might be more than one.

AUTHOR'S PUBLICATIONS

M. J. Er, V. K. Yalavarthi, N. Wang, and R. Venkatesan, "A Novel Incremental Class Learning Technique for Multi-class Classification," 13th International Symposium on Neural Networks, pp. 474–481, July 2016, Springer

BIBLIOGRAPHY

- [1] J. Cao, K. Mao, J. Wu, and A. Lendasse, “Proceedings of ELM-2015 Volume 2,” *Springer*.
- [2] G. Tsoumakas and I. Katakis, “Multi-label classification: An overview,” *Dept. Informatics, Aristotle Univ. ...*, 2006.
- [3] T. G. Dietterich and G. Bakiri, “Solving Multiclass Learning Problems via Error-Correcting Output Codes,” *J. Artif. Intell. Res.*, vol. 2, pp. 263–286, 1995.
- [4] L. Olshen and C. Stone, “Classification and regression trees,” *Wadsworth Int. Gr.*, 1984.
- [5] L. Breiman, J. Friedman, C. Stone, and R. Olshen, “Classification and regression trees,” 1984.
- [6] W. Loh, “Classification and regression trees,” *Wiley Interdiscip. Rev. Data Min. ...*, 2011.
- [7] M. King and P. Resick, “Data mining in psychological treatment research: A primer on classification and regression trees,” *J. Consult. Clin. Psychol.*, 2014.
- [8] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Elsevier, 2014.
- [9] S. Bay, “Combining Nearest Neighbor Classifiers Through Multiple Feature Subsets,” *ICML*, 1998.

- [10] J. Rennie, “Improving multi-class text classification with naive Bayes,” 2001.
- [11] N. Mehra and S. Gupta, “Survey on multiclass classification methods,” 2013.
- [12] M. Aly, “Survey on multiclass classification methods,” *Neural Netw*, 2005.
- [13] E. Bredensteiner and K. Bennett, “Multicategory classification by support vector machines,” *Comput. Optim.*, 1999.
- [14] K. Crammer and Y. Singer, “On the algorithmic implementation of multiclass kernel-based vector machines,” *J. Mach. Learn. Res.*, vol. 2, pp. 265–292, Mar. 2002.
- [15] Y. Lee, Y. Lin, and G. Wahba, “Multicategory Support Vector Machines,” *J. Am. Stat. Assoc.*, vol. 99, no. 465, pp. 67–81, Mar. 2004.
- [16] G. Bo and H. Xianwu, “SVM multi-class classification,” *J. Data Acquis. Process.*, 2006.
- [17] M. Pal, “Multiclass Approaches for Support Vector Machine Based Land Cover Classification,” p. 16, Feb. 2008.
- [18] C. Hsu and C. Lin, “A comparison of methods for multiclass support vector machines,” *Neural Networks, IEEE Trans.*, 2002.
- [19] S. Kumar, J. Ghosh, and M. M. Crawford, “Hierarchical Fusion of Multiple Classifiers for Hyperspectral Data Analysis,” *Pattern Anal. Appl.*, vol. 5, no. 2, pp. 210–220.
- [20] Z.-H. Zhou and Z.-Q. Chen, “Hybrid decision tree,” *Knowledge-Based Syst.*, vol. 15, no. 8, pp. 515–528, Nov. 2002.

- [21] R. Polikar, L. Upda, S. S. Upda, and V. Honavar, “Learn++: an incremental learning algorithm for supervised neural networks,” *IEEE Trans. Syst. Man Cybern. Part C (Applications Rev.)*, vol. 31, no. 4, pp. 497–508, 2001.
- [22] R. Polikar, J. Byorick, S. Krause, and A. Marino, “Learn++: A classifier independent incremental learning algorithm for supervised neural networks,” *, 2002. IJCNN’02. ...*, 2002.
- [23] M. Muhlbaier, A. Topalis, and R. Polikar, “Learn++.MT: A New Approach to Incremental Learning,” Springer Berlin Heidelberg, 2004, pp. 52–61.
- [24] M. Muhlbaier, “Learn. NC: combining ensemble of classifiers with dynamically weighted consult-and-vote for efficient incremental learning of new classes,” *Neural Networks, IEEE ...*, 2009.
- [25] R. Polikar, J. DePasquale, H. Syed Mohammed, G. Brown, and L. I. Kuncheva, “Learn++.MF: A random subspace approach for the missing feature problem,” *Pattern Recognit.*, vol. 43, no. 11, pp. 3817–3832, 2010.
- [26] G. Ditzler, M. D. Muhlbaier, and R. Polikar, “Incremental Learning of New Classes in Unbalanced Datasets: Learn++.UDNC,” Springer Berlin Heidelberg, 2010, pp. 33–42.
- [27] T. Hoya, “On the capability of accommodating new classes within probabilistic neural networks,” *IEEE Trans. Neural Netw.*, vol. 14, no. 2, pp. 450–3, Jan. 2003.
- [28] B. Zhang, J. Su, and X. Xu, “A Class-Incremental Learning Method for Multi-Class Support Vector Machines in Text Classification,” in *2006 International*

Conference on Machine Learning and Cybernetics, 2006, pp. 2581–2585.

- [29] R. Venkatesan and M. J. Er, “A novel progressive learning technique for multi-class classification,” *Neurocomputing*, 2016.
- [30] Z. Zhao, Z. Chen, Y. Chen, S. Wang, and H. Wang, “A Class Incremental Extreme Learning Machine for Activity Recognition,” *Cognit. Comput.*, vol. 6, no. 3, pp. 423–431, Apr. 2014.
- [31] G. Huang, Q. Zhu, and C. Siew, “Extreme learning machine: a new learning scheme of feedforward neural networks,” *Neural Networks*, 2004. ..., 2004.
- [32] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, “Extreme learning machine for regression and multiclass classification,” *IEEE Trans. Syst. Man. Cybern. B. Cybern.*, vol. 42, no. 2, pp. 513–29, Apr. 2012.
- [33] G. Huang, Q. Zhu, and C. Siew, “Extreme learning machine: theory and applications,” *Neurocomputing*, 2006.
- [34] G. Huang, G.-B. Huang, S. Song, and K. You, “Trends in extreme learning machines: A review,” *Neural Networks*, vol. 61, pp. 32–48, 2015.
- [35] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, “Extreme learning machine: Theory and applications,” *Neurocomputing*, vol. 70, no. 1–3, pp. 489–501, Dec. 2006.
- [36] N. Wang, M. J. Er, and M. Han, “Generalized single-hidden layer feedforward networks for regression problems,” *IEEE Trans. neural networks Learn. Syst.*, vol. 26, no. 6, pp. 1161–76, Jun. 2015.
- [37] N. Wang, M. J. Er, and M. Han, “Parsimonious extreme learning machine

- using recursive orthogonal least squares,” *IEEE Trans. neural networks Learn. Syst.*, vol. 25, no. 10, pp. 1828–41, Oct. 2014.
- [38] J. Cao, Z. Lin, G.-B. Huang, and N. Liu, “Voting based extreme learning machine,” *Inf. Sci. (Ny)*, vol. 185, no. 1, pp. 66–77, Feb. 2012.
- [39] M. J. Er, R. Venkatesan, and N. Wang, “A High Speed Multi-label Classifier Based on Extreme Learning Machines,” Springer International Publishing, 2016, pp. 437–454.
- [40] R. Venkatesan, M. J. Er, M. Dave, M. Pratama, and S. Wu, “A novel online multi-label classifier for high-speed streaming data applications,” *Evol. Syst.*, pp. 1–13, Aug. 2016.
- [41] M.-B. Li, G.-B. Huang, P. Saratchandran, and N. Sundararajan, “Fully complex extreme learning machine,” *Neurocomputing*, vol. 68, pp. 306–314, Oct. 2005.
- [42] G.-B. Huang, M.-B. Li, L. Chen, and C.-K. Siew, “Incremental extreme learning machine with fully complex hidden nodes,” *Neurocomputing*, vol. 71, no. 4, pp. 576–583, 2008.
- [43] Z. Shao, M. J. Er, and N. Wang, “Robust incremental extreme learning machine,” in *2014 13th International Conference on Control Automation Robotics & Vision (ICARCV)*, 2014, pp. 607–612.
- [44] G. Huang, G.-B. Huang, S. Song, and K. You, “Trends in extreme learning machines: A review,” *Neural Networks*, vol. 61, pp. 32–48, Jan. 2015.

- [45] G.-B. Huang, D. H. Wang, and Y. Lan, “Extreme learning machines: a survey,” *Int. J. Mach. Learn. Cybern.*, vol. 2, no. 2, pp. 107–122, May 2011.
- [46] G.-B. Huang, “An Insight into Extreme Learning Machines: Random Neurons, Random Features and Kernels,” *Cognit. Comput.*, vol. 6, no. 3, pp. 376–390, 2014.
- [47] E. Cambria, G.-B. Huang, L. L. C. Kasun, H. Zhou, C. M. Vong, J. Lin, J. Yin, Z. Cai, Q. Liu, K. Li, V. C. M. Leung, L. Feng, Y.-S. Ong, M.-H. Lim, A. Akusok, A. Lendasse, F. Corona, R. Nian, Y. Miche, P. Gastaldo, R. Zunino, S. Decherchi, X. Yang, K. Mao, B.-S. Oh, J. Jeon, K.-A. Toh, A. B. J. Teoh, J. Kim, H. Yu, Y. Chen, and J. Liu, “Extreme Learning Machines [Trends & Controversies],” *IEEE Intell. Syst.*, vol. 28, no. 6, pp. 30–59, Nov. 2013.
- [48] X. Wang and M. Han, “Improved extreme learning machine for multivariate time series online sequential prediction,” *Eng. Appl. Artif. Intell.*, vol. 40, pp. 28–36, Apr. 2015.
- [49] Q.-Y. Zhu, A. K. Qin, P. N. Suganthan, and G.-B. Huang, “Evolutionary extreme learning machine,” *Pattern Recognit.*, vol. 38, no. 10, pp. 1759–1763, Oct. 2005.
- [50] N. Liu and H. Wang, “Ensemble Based Extreme Learning Machine,” *IEEE Signal Process. Lett.*, vol. 17, no. 8, pp. 754–757, Aug. 2010.
- [51] G.-B. Huang and L. Chen, “Convex incremental extreme learning machine,” 2007.

- [52] G. Guorui Feng, G.-B. Guang-Bin Huang, Q. Qingping Lin, and R. Gay, “Error Minimized Extreme Learning Machine With Growth of Hidden Nodes and Incremental Learning,” *IEEE Trans. Neural Networks*, vol. 20, no. 8, pp. 1352–1357, Aug. 2009.
- [53] 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. 1411–23, Nov. 2006.
- [54] G. Huang and N. Liang, “On-Line Sequential Extreme Learning Machine,” *Comput. ...*, 2005.
- [55] G. H. (Gene H. Golub and C. F. Van Loan, *Matrix computations*. Johns Hopkins University Press, 1996.
- [56] Y. Yuan Lan, Y. C. Soh, and G.-B. Huang, “A constructive enhancement for Online Sequential Extreme Learning Machine,” in *2009 International Joint Conference on Neural Networks*, 2009, pp. 1708–1713.
- [57] Y. Jun and M. J. Er, “An Enhanced Online Sequential Extreme Learning Machine algorithm,” in *2008 Chinese Control and Decision Conference*, 2008, pp. 2902–2907.
- [58] B. Li, J. Wang, Y. Li, and Y. Song, “An improved on-line sequential learning algorithm for extreme learning machine,” *Adv. Neural Networks–ISNN 2007*, vol. 4491, pp. 1087–1093, 2007.
- [59] H.-J. Rong, G.-B. Huang, N. Sundararajan, and P. Saratchandran, “Online sequential fuzzy extreme learning machine for function approximation and

- p>classification problems.,”
- IEEE Trans. Syst. Man. Cybern. B. Cybern.*
- , vol. 39, no. 4, pp. 1067–72, Aug. 2009.
- [60] Y. Lan, Y. C. Soh, and G.-B. Huang, “Ensemble of online sequential extreme learning machine,” *Neurocomputing*, vol. 72, no. 13–15, pp. 3391–3395, Aug. 2009.
 - [61] J. Zhao, Z. Wang, and D. S. Park, “Online sequential extreme learning machine with forgetting mechanism,” *Neurocomputing*, vol. 87, pp. 79–89, 2012.
 - [62] Y. Ye, S. Squartini, and F. Piazza, “Online sequential extreme learning machine in nonstationary environments,” *Neurocomputing*, vol. 116, pp. 94–101, 2013.
 - [63] H. Zou, H. Jiang, X. Lu, and L. Xie, “An online sequential extreme learning machine approach to WiFi based indoor positioning,” in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, 2014, pp. 111–116.
 - [64] S. Scardapane, D. Comminiello, M. Scarpiniti, and A. Uncini, “Online Sequential Extreme Learning Machine With Kernels,” *IEEE Trans. Neural Networks Learn. Syst.*, vol. 26, no. 9, pp. 2214–2220, Sep. 2015.
 - [65] B. P. Chacko and A. P. Babu, “Online sequential extreme learning machine based handwritten character recognition,” in *IEEE Technology Students’ Symposium*, 2011, pp. 142–147.
 - [66] Guang-Bin Huang, “Learning capability and storage capacity of two-hidden-layer feedforward networks,” *IEEE Trans. Neural Networks*, vol. 14, no. 2, pp. 274–281, Mar. 2003.

- [67] F. Zhang, Ed., *The Schur Complement and Its Applications*, vol. 4. New York: Springer-Verlag, 2005.