Online Sequential Learning with Non-iterative Strategy for Feature Extraction, Classification and Data Augmentation

by

Adhri Nandini Paul
Lakehead University

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTERS

in the Department of Computer Science

Lakehead University

Online Sequential Learning with Non-iterative Strategy for Feature Extraction, Classification and Data Augmentation

by

Adhri Nandini Paul
Lakehead University

Supervisory Committee

Dr. Yimin Yang, Supervisor
(Department of Computer Science, Lakehead University, Canada)

ABSTRACT

Network aims to optimize for minimizing the cost function and provide better performance. This experimental optimization procedure is widely recognized as gradient descent, which is a form of iterative learning that starts from a random point on a function and travels down its slope, in steps, until it reaches to the steepest point which is time-consuming and slow to converge. Over the last couple of decades, several variations of the non-iterative neural network training algorithms have been proposed, such as Random Forest and Quicknet. However, the non-iterative neural network training algorithms do not support online training that given a very large-sized training data, one needs enormous computing resources to train neural network. In this thesis, a non-iterative learning strategy with online sequential has been exploited. In Chapter 3, a single layer Online Sequential Sub-Network node (OS-SN) classifier has been proposed that can provide competitive accuracy by pulling the residual network error and feeding it back into hidden layers. In Chapter 4, a multi-layer network is proposed where the first portion built by transforming multi-layer autoencoder into an Online Sequential Auto-Encoder(OS-AE) and use OS-SN for classification. In Chapter 5, OS-AE is utilized as a generative model that can construct new data based on subspace features and perform better than conventional data augmentation techniques on real-world image and tabular datasets.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

First of all, I express my deepest gratitude and special thanks to my supervisor **Dr. Yimin Yang**, for providing me the chance to work under his guidance. He is a teacher and mentor with an unmatched combination of intellect, intuition and wit. His constructive suggestions, and encouragement is the reason I was able to learn and grow as a researcher. It was an absolute privilege to work with him on this research. I express my deepest thanks to my parents for taking part in giving necessary advice and guidance and arranging all facilities to make learning easier. I choose this moment to acknowledge their contribution gratefully.

Chapter 1

Introduction

1.1 Overview

The goal of AI is to ameliorate the interaction between computers and the modern world and associate the connection intelligently. Nowadays, deep learning has penetrated this concert with an artificial neural network and revolutionized many real-world applications such as computer vision [58], speech process [35], and computational biotechnology [96], and so on. To cope with the increasing demand for usage of these applications, a wide variety of neural structures appeared. Deep Neural Network(DNN) is a variant of neural structures where more layers can add to solve complex nonlinear problems and produce an unprecedented result by utilizing iterative learning. So we start with a question: What is iterative learning? Generally speaking, it is a training process of neural networks to improve their learning accuracy. Every neural network has been build upon input and output nodes, which connect through some hidden layers. Raw data fed into the network through the input node. A set of actions performed to generate a result as the output. Output

compared with the actual output of the dataset and error is fed back to that network to update the weights of hidden layers, which referred to as back-propagation. This iterative learning continues until the produced output reaches close to the actual output of the dataset. As we move backward to update the weights, it takes a long time to get the proper gradient values. Sometimes, the gradient gets too small and gets stuck into local minimum value, which termed as vanishing gradient. For reducing the complexity faced by gradient descent, some non-iterative learning algorithms have emerged, such as Random Forest (RF) [6], Quicknet [27], Extreme Learning Machine (ELM). Unlike iterative learning, the fact that one needs to feed all the training data to the model at once for non-iterative learning, and load the whole huge dataset into the computer memory is infeasible.

The goal of a machine learning model is to capture underlying patterns well from any

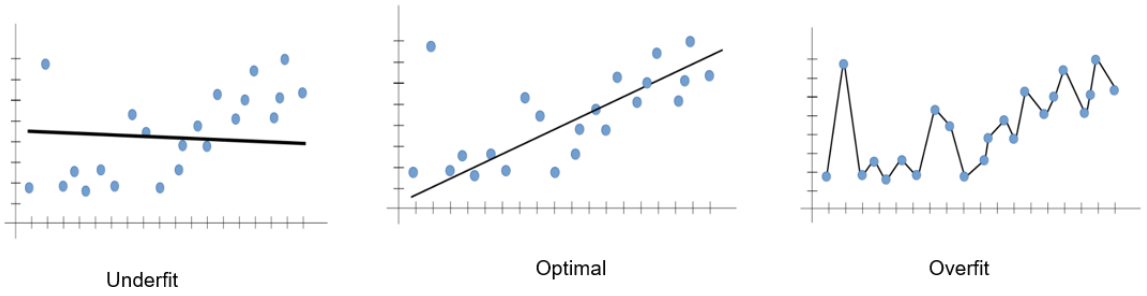

Figure 1.1: Problems faced by network when defining the number of layers.

data of the problem domain. However, underfitting and overfitting are the biggest problems faced by machine learning algorithms. Underfitting occurs when a model is quite simple and unable to learn prominent features of the data. As a result, it suffers from high bias and low variance. On the other hand, overfitting appears when a model captures all underlying patterns from data rather than a prominent one. These models face low bias and high variance.

In Figure 1.1, underfitting and overfitting problems are demonstrated through a regression model where the model is trying to fit through the actual data point. In the first graph, the model learns a less dominant feature by intersecting fewer data points, which implies an underfitting problem. In the second graph, the model yields a smaller distance from data points that define the reliability of the model. In the third graph, the model captured all trends rather than the dominant one, which causes overfitting problems. To solve these problems, DNN required to define an optimal number of layers. Having fewer layers leads to underfitting, and then an increasing

number of layers will cause an overfitting problem. For reducing these problems, a large number of labeled training data required. Various data augmentation techniques are available such as: rotating images, flipping images, adding blur, adjusting saturation, cropping, and so on. Some DNN based generative models named Generative Adversarial Networks(GANs)[29] and Variational Autoencoders (VAEs) [55] has gained attention from researchers because of generating coherent and detailed images in an unsupervised manner. However, these models are trapped in back-propagation and suffer high-convergence. That is the reason for these models are expensive and time-consuming. Moreover, they focused on only image data.

1.2 Problem Description

The main problems on which we are focused on to be solved are as follows:

- Iterative learning-based models suffer from gradient descent problems, and as a result, it required lots of time to converge for getting a small norm of network weight vector.
- Most of the traditional hierarchical networks need a tremendous amount of computation power and time to train how to reduce time complexity and gain better performance.
- Neural networks often face overfitting problems due to the small scale of the dataset. So we focus on finding out a way to improve the generative model efficiently for both tabular and image data augmentation.
- Neural networks suffer from selecting an optimal number of hidden layers. How to efficiently generate the number of hidden nodes as well as layers?

In our research, we focus on building SLFN by utilizing online sequential training, which can obtain better classification accuracy than traditional ones. The whole process and the experimental result described in Chapter 3. By transforming high dimensional data into the essential feature, the sub-set can increase the efficiency of machine learning algorithms significantly for object classification. For that reason, we proposed an autoencoder and concatenated it with the classifier to build a hierarchical network in Chapter 4 (See Figure 1.2). This hierarchical network can solve the problems faced by gradient descent and provide better performance without consuming a massive amount of memory as well as time. We noticed that our proposed

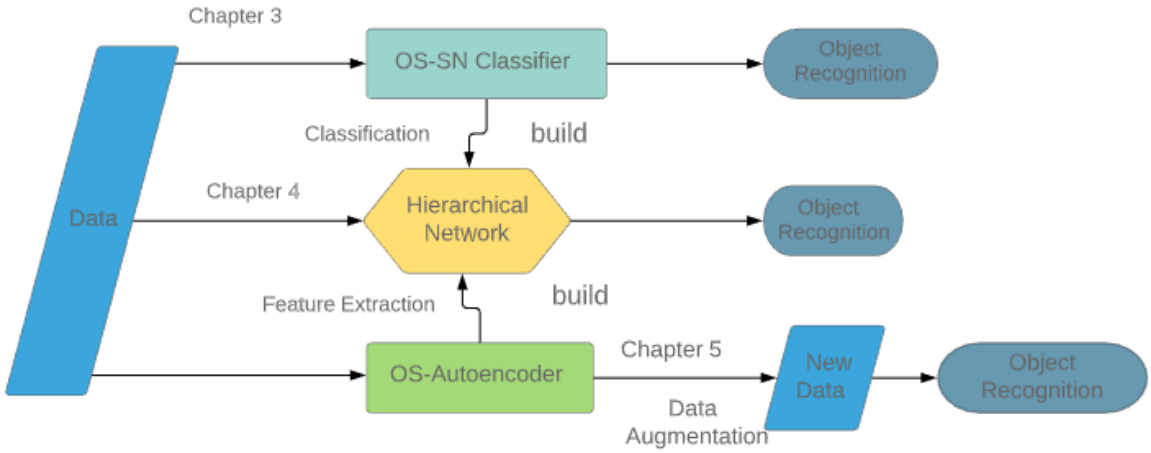

Figure 1.2: Overview of proposed networks and construction.

autoencoder works well in the field of feature extraction. For that reason, in Chapter 5, we extend this autoencoder to generate new data and deploy it to solve the over-fitting problem. We test our proposed model in both image and tabular datasets and compare the result with some data augmentation techniques.

1.3 Contribution

This thesis focuses on the improvement in the field of object classification. We depict the overview in Figure 5.3, to provide the abstraction of our work and how we modify our research work as a whole to build an efficient one. Therefore, through our experiments, this paper has made the following contributions:

- A classifier with online sequential learning proposed that it can optimize the usage of hidden nodes and calculate the input weight instead of using random value
- A multi-layer network with an online sequential learning strategy is buid which is a combination of feature extractor and the classifier. Non-iterative strategy is exploited to build like BP based algorithm can process data batch-wise, but there is no need to configure any learning parameters such as learning rate, number of learning epochs, stopping criteria, and other predefined parameters.

- We propose a generative model by utilizing our proposed autoencoder. Experimental results show that a DCNN model with augmented data via our proposed algorithm acquire higher classification accuracy rather than same DCNN model itself or other data augmentation techniques. For instance, Resnet [33] combined with our method achieves 94.68% image recognition accuracy from CIFAR-10 [57] augmented dataset with only five epochs.

1.4 Organization of Thesis

This section was all about introduction and rest of the thesis proceeds as follows,
Chapter II gives detailed insight into the background for the various methods and algorithms analyzed in this thesis.
Chapter III introduced an Online Sequential Classifier with Sub-network Node for object classification.
Chapter IV introduced an Online Sequential Autoencoder(OS-AE) to extract deep features for dimension reduction and image reconstruction. Moreover, we extend Chapter III to build a multi layer network that can classify the object based on the extracted feature.
Chapter V is an extension of chapter III, and we utilize features extraction quality of OS-AE to generate both image and tabular data.
Chapter VI is the last in this thesis, which summarizes the whole work in this thesis and further explaining the prospects of the research.

Chapter 2

Background and Related Work

2.1 Background

Neural Networks have been a hot topic since their first formulation during the 40s. After their introduction, many usages have discovered in the field of machine learning and artificial intelligence. A standard neural network constructs with many connected and straightforward processors named neurons, some non-linear activation passes through the neurons to activate. Input neurons get activated through sensors come from input data; sequentially, other neurons get activated through weighted connections from previously active neurons. A optimized weighted connection minimizes the cost function and trigger network to exhibit desired behavior. Many successive non-linear layer based models were popular in the era of the 60s to 70s. An efficient gradient descent method called backpropagation (BP) developed and applied to

train these models. However, BP-based training of NNs had been trying in practice because of less computation power. In 1980, Neocognitron [23] was the first deep artificial NN that sequentially assimilate the neuro-physiological insights of the visual cortex and response to specific properties of visual sensory inputs, such as the orientation of edges. It introduced convolutional NNs (CNNs or convnets), where the receptive field of a convolutional unit (basically rectangular) with a given weight vector (a filter) shifted step by step across a 2-dimensional array of input values (pixels of an image). It is quite similar to modern, contest-winning, feed-forward, gradient-based Deep CNN networks such as Alexnet [58], Resnet [33], and so on by alternating convolutional and down-sampling layers.

Nevertheless, Fukushima used WTA-based unsupervised learning rules [25] instead of backpropagation to set the weights in Neocognitron. For downsampling purposes, he altered Max-Pooling (MP) by Spatial Averaging. In the new millennium, DL combinations of BP based CNNs and MP able to provide outstanding performance because of the availability of multi-processor Graphics Cards Unit (GPU). As GPUs are widely popular for video games and to mitigate the up-growing demand, the competitive market had reduced hardware prices. GPUs accelerate matrix and vector multiplications and speed up the learning ability of NN. As a result, deep NNs have finally gained wide-spread attention and contributed many efficient alternative machine learning algorithms such as kernel machines [91], many feed-forward neural networks(FNNs). Most FNN applications focused on FNNs with few hidden layers as additional layers often did not provide any kind of practical benefits. Many practitioners state that a single layer feed-forward neural network(SLFN) with enough hidden units can approximate any multivariate continuous function with arbitrary accuracy [56]. Over the past two decades, SLFNs have become an exciting topic for many researchers because of their universal approximation capability [9]. It revolutionized the machine learning technique and played a vital role in the field of both regression and classification-related problems. An SLFN builds upon one input layer to receive input from external environments, a single hidden layer, and one output layer to send network output to external environments. We can say that for N arbitrary distinct samples (x_i, t_i) , the network output is

$$\mathbf{f}_L(\mathbf{x}) = \sum_{i=1}^{L} \boldsymbol{\beta}_i h(\mathbf{a}_i \cdot \mathbf{x}_j + b_i) = \sum_{i=1}^{L} \mathbf{H}_i \cdot \boldsymbol{\beta}_i, j = 1, \ldots, N \tag{2.1}$$

where $h(\cdot)$ denotes an activation function, (a_i, b_i) denotes the input weight and bias of i^{th} hidden node , and β_i is the i^{th} output weight between the hidden node and the output nodes. A neural network gives small squared error when the weight of the nodes are small as well [3]. Because, the generalization performance largely depends on the weights rather than number of nodes. if activation function $h(\cdot)$ is invertible, $\mathbf{a}_i$ and $\boldsymbol{\beta}$ would be the smallest norm and provide the smallest training error. According to the neural network theories, SLFNs work as universal approximators whenever (a, b, β) parameters are adjusted [104]. Back-propagation works behind of training SLFNs with additive hidden nodes. Stochastic gradient descent BP (SGBP) [49] is one of the main variants of BP for the batch learning process. The goal is to minimize the cost functions through gradient descent in the parameter space of NN by adapting control parameters (weights). Sometimes, NN may suffer from vanishing or exploding gradients because of backpropagated error signals with standard activation functions. As a result, the gradient would either shrink rapidly or grow out of bounds. For solving these problems, numerous approaches proposed to gain steepest descent through BP. Such as Least-squares methods (Gauss-Newton, Levenberg-Marquardt) [26, 72] and quasi-Newton methods [83] . However, they are computationally expensive for large NNs. For increasing the learning speed, ad-hoc constants added [18] to the slope of the activation function in BP. Moreover, momentum was introduced to determine the direction of gradient [84]. Some algorithms approached to control BP step size for adapting a global learning rate [61] instead of computing individual learning rates for each weight [48]. In BP based online learning, each weight's learning rate inversely proportional to the empirical standard deviation of its local gradient [73]. As a result, stochastic weight fluctuates. For minimizing this fluctuation, sequential learning algorithms have become an integral part of training feed-forward networks. What is sequential learning? It is a neural procedure associated with finding out about the best possible ordering of events [10]. According to Ritter et al. [81], "The order in which material is presented can strongly influence what is learned, how fast performance increases, and sometimes even whether the material is learned at all". Resource Allocation Network (RAN) [76] and its extensions [69, 103] are sequential learning algorithm and gained popularity because of it's fastest learning speed. However, it handles data one by one instead of chunk by chunk (block of data). Moreover, it can only work with either additive or radial basis function (RBF) types of hidden nodes [44]. Huang et al. [64] proposed an online sequential learning algorithm that process data chunk by chunk. However, it faces problem in choosing

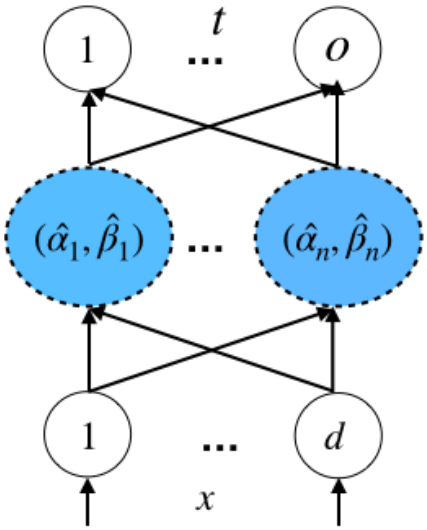

Figure 2.1: Network architecture of a Sub-network based classifier.

appropriate number of hidden nodes. Yang et al. proposed a non-iterative learning algorithm consisting of a hidden node within a sub-network, generated by calculation and provide competitive accuracy [100].

2.1.1 Fully-Connected Classifier with Sub-network Nodes

Network can grow sub-network nodes that form by pulling back residual error to hidden layer. It forms through a single hidden node or several nodes and focused on reaching the smallest training error as well as the smallest norm between output weight and hidden nodes [2, 3].

According to Bidirectional ELM [36], this residual error can be pulled back to the network, which will help to select a minimal number of neurons or sub-network itself to get higher classification accuracy. N numbers of distinct samples of $(x_i, t_i{}^N)$ and a sigmoid or sine activation function h , the input weight and output weight of n^{th} subnetwork would be

$$\hat{\boldsymbol{\alpha}_n} = h^{-1}(u(\mathbf{e}_{n-1})) \cdot \mathbf{x}^T(\mathbf{I}_{d\times d}/c + \mathbf{x}\mathbf{x}^T)^{-1}, \tag{2.2}$$

$$\hat{\boldsymbol{\beta}}_n = \mathbf{H}^{\dagger} g^{-1}(\mathbf{e}_{n-1}), \tag{2.3}$$

where $h^{-1}(\cdot)$ has been used as the inverse function of $h(\cdot)$; u is used as a normalized

function which processes the input and target data by mapping from original range to (0,1]; This functions applied to the residual network error of the previous subnetwork; $\mathbf{x}^T(\mathbf{I}/c + \mathbf{x}\mathbf{x}^T)^{-1} = \mathbf{x}^{-1}$ is the Moore-Penrose generalization inverse of training samples. Meanwhile, $\mathbf{H}^\dagger = (\mathbf{H}^T\mathbf{H} + (\mathbf{I}_{m\times m})/c)^{-1}\mathbf{H}^T$ can be used in Eq. 2.3 to get the output weight of that sub-network. The residual error of n^{th} sub-network can be defined by Eq.2.5

$$\mathbf{e}_n = \mathbf{t} - \mathbf{H} \cdot \boldsymbol{\beta}, \tag{2.4}$$

Huang et al. [64] proved that non-iterative learning issue can be solved by online sequential training. Instead of sending the entire dataset, data will be presented chunk by chunk with varying or fixed lengths of the chunks and the output weights of which constantly updated by adopting a Recursive Least Squares (RLS) algorithm. For that reason, it can deliver better performance than the other iterative learning algorithms and successfully applied in the field of system modeling and object classification, such as online nonlinear system identification [85], consumer sentiments prediction [89], and time series prediction [31]. However, it may still suffer from some issue of instability due to randomized input weight, and as a result, generalization performance may degrade if the number of hidden nodes in SLFNs has not set appropriately [30, 46].

From the last few decades, data mining is started from all kinds of different actions happening around the world, such as weather updates, medical records, communication system to build a data warehouse that machines can use to learn. For learning the intricate structure of these high-dimensional data and the object classification accuracy, the neural network has utilized by contributing some benchmark hierarchical networks, which work as a pioneer in the deep learning domain. Such as : Deep Belief Network [38], recurrent neural network [71], multi-layer neural network [14, 94, 95] non-iterative hierarchical network [52, 102] and so on [33, 66, 67, 93]. These networks work well at discovering intricate structures in high-dimensional data as they are efficient to extract meaningful features and map these features for classification. What are the features? Generally speaking, features are a low-dimensional representation of training or testing objects, which can provide insights into the raw high-dimensional input data. The quality of the features determines whether the subsequent classification and recognition will get a good result. The feature extraction techniques designed to reduce dimensional by choosing informative and non-redundant feature subset from a large number of components in data. For that reason, processing high

dimensional data requires a large amount of memory and computation power [34]. Based on the availability of label information, feature extraction techniques developed in a supervised, unsupervised, or semi-supervised manner. Supervised methods evaluate the correlation between feature and class labels, based on which the features are selected. It includes linear discrimination analysis (LDA) [16], neighborhood components analysis (NCA) [28], Isometric Projection [7].
Unsupervised feature extraction usually focuses on mapping function to find the best subspace from the geometrical structure of the data space. It includes principal component analysis (PCA) [50], information network embedding [90]. Many machine learning practitioners believe that unsupervised feature extraction methods involving neural networks can achieve high performance rather than traditional pattern recognition methods because deep learning method automatically learns features from big data. There are many ways to extract features based on neural networks, such as extracting features through convolutional neural networks (CNN) or using unsupervised learning models, autoencoders, etc. Autoencoders [92] are a special variant of neural networks, involving an unsupervised learning algorithm that compresses the input into a lower-dimensional and extracts prominent features efficiently. Moreover, they can reconstruct the output from that lower-dimensional representation. A brief description has given below

2.1.2 Autoencoder

Autoencoder is a backpropagation based algorithm through which real data provide as input [37]. Suppose the input vector x is first mapped to a hidden representation where z = f(x) and passed through the encoding layer, construct with one or more layers of non-linearity. The hidden representation z is afterward passed through the decoder to generate output $\hat{x} = g(z)$, which parametric the decoder function g. Similar to the encoder, the decoder network built on multiple layers of non-linearity. This process needs several iterations to adjust its weight and bias to learn deep features with the reduced numbers of hidden nodes, and based on these features; it can reconstruct whole input as an output. The goal of the autoencoder is to minimize the following cost function:

$$L(x, g(f(x))) = ||\hat{x} - x||^2, \tag{2.5}$$

Eq. 2.5 utilized to reduce the error between the inputs and the reconstructed outputs of the network and gradually decreased until it reaches below a threshold, which consumes much time. For reducing the time consumption rate, a non-iterative multi-layer autoencoder introduced [51, 99]. However, hidden nodes used in the encoding layer are randomly generated that can deteriorate useful features. Instead of using several random layers, a multi-layer autoencoder introduced, [99], where only the encoding layer weight has been generated randomly, based on which the decoding layer weight has calculated.

2.1.3 Non-iterative multi layer Autoencoder

Non-iterative multi-layer autoencoder learns prominent features based on singular values and significantly faster than the existing deep networks [51]. Instead of using several random layers, input data mapped into L dimensional feature space. Though the encoding layer weight has generated randomly, decoding layer weight has calculated. A brief description of the non-iterative multi-layer autoencoder algorithm explained in the following four steps.

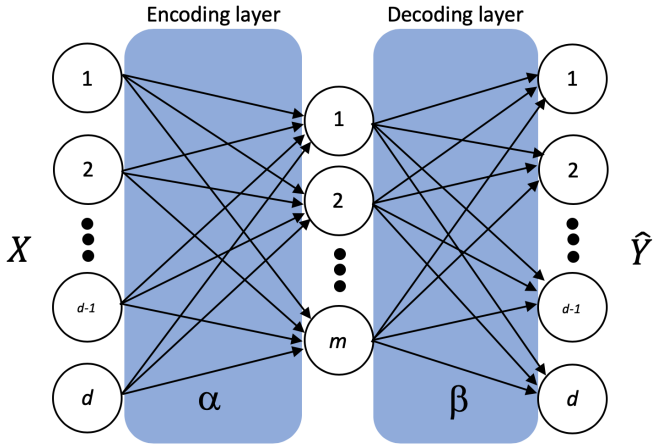

Figure 2.2: The abstracted architecture of a multi layer autoencoder.

Step-1: Randomly initialize the encoding layer weights α and biases b using an orthogonal random process. The orthogonal random process will ensure the random weights and biases satisfy the conditions in Eq. 2.6

$$\boldsymbol{\alpha}^T\boldsymbol{\alpha} = \boldsymbol{I}_{m\times m}, b^T b = 1, \tag{2.6}$$

where $I_{m\times m}$ represents an $m \times m$ identity matrix. After initializing the encoding layer, obtain the encoding layer output H through Eq. 2.7.

$$\mathbf{H} = g(X\boldsymbol{\alpha} + b), \tag{2.7}$$

Step-2: Calculate the decoding layer weights β through Eq. 2.8

$$\boldsymbol{\beta} = \mathbf{H}^{\dagger} g^{-1}(\mathbf{Y}), \tag{2.8}$$

where $g^{-1}(\cdot)$ is the inverse function of $g(\cdot)$. This inverse function helps $\boldsymbol{\beta}$ to be the smallest norm among all the least-squares solutions. For instance, if $g(\cdot) = sin(\cdot)$, then $g^{-1}(\cdot) = arcsin(\cdot)$; if $g(\cdot)$ is sigmoid function, then $g^{-1}(\cdot) = -log(1/(\cdot) - 1)$. It is worth noting that, the pseudo inverse $\mathbf{H}^{\dagger}$ [79] is calculated by Eq. 2.9.

$$\mathbf{H}^{\dagger} = (\mathbf{H}^T\mathbf{H} + (\mathbf{I}_{m\times m})/c)^{-1}\mathbf{H}^T, \tag{2.9}$$

Then, obtain b regarding Eq. 2.10

$$b = rmse(\mathbf{H}\boldsymbol{\beta} - g^{-1}(\mathbf{Y})), \tag{2.10}$$

where $rmse(\cdot)$ is the root mean squared error. The Singular Value Decomposition (SVD) of Eq. 2.8 can be written as

$$\mathbf{H}\boldsymbol{\beta} = \sum_{i=1}^{N} \mathbf{u}_i \frac{\mathbf{d}_i^2}{\mathbf{d}_i^2 + C} \mathbf{u}_i^T \mathbf{X}, \tag{2.11}$$

where $\mathbf{u}$ are the eigenvectors of $\mathbf{H}\mathbf{H}^T$ and d are the singular values of H and SVD of input data X. That is reason why H works as the projected feature space of input data. For that reason, output weight $\boldsymbol{\beta}$ of this autoencoder can learn feature from input data. However, it is a non-iterative process, and there is no way to upgrade the learning process based on the error between the inputs and the reconstructed outputs of the network. Moreover, decoding layer weight is calculated based on encoding layer weight which generated randomly. As a result, it decreases the efficiency of autoencoder by capturing anonymous information rather than relevant ones.

2.2 Generative Model

The generative Model is a theory that involves any kind of data distribution using unsupervised learning, and it has gained tremendous popularity within a few years. Generative models aim to learn existing data distribution from the training set and generate new data points by adding variations. It is quite hard to learn exact data distribution, either implicitly or explicitly. Neural network learns true data distribution efficiently and model the learned distribution by integrating new ideas. Among the generative models, Variational Autoencoders (VAE) and Generative Adversarial Networks (GAN) are two most efficient approaches using neural networks in an unsupervised fashion. A brief discussion has given below.

2.2.1 Variational Autoencoder

In machine learning, dimension reduction plays a key role by choosing attributes from features that describe the dominant portion of data and utilize in many situations where low dimensional data is required such as data transmission, Data Storage, heavy computation. This reduction implies either by selection (choose prominent features from existing) or by extraction (combine old features to create a meaningful minimal amount of new features). The ideal dimension reduction approach is to find the best encoder/decoder pair that can keep the maximum information while encoding and generate a minimum of reconstruction error when decoding from a designated dataset. Variational autoencoders (VAE) [55, 80] are probabilistic encoder/decoder pair that uses a stochastic encoder to learn the probability distribution $q(z \mid x)$ from training data and pair it with a generative network that maximizes the log-likelihood $\log p(x \mid z)$ of training data .

$$\log(p(x)) > \mathbb{E}_q(z \mid x)[\log(p(x \mid z))] - KL(q(z \mid x)||p(x)),$$

KL is the Kullback–Leibler divergence between two distributions. Equation 2.2.1 aims to maximize the log-likelihood of our data distribution by adding a regularization term given by $KL(q(z \mid x)||p(x))$. VAE is trying to minimize the lower bound of $\log(P(X))$ the KL-divergence term is less than 0. This KL-divergence is similar to maximize $\mathbb{E}_q(z \mid x)$ as a maximum likelihood estimation and performed as a decoder.

2.2.2 Generative Adversarial Networks

The Generative Adversarial Networks (GAN) [29] are built based on a min-max adversarial game between two neural networks to find the Nash equilibrium point . One is a generative model, G, and another is discriminative model, D. A generator model G uses a function G(z) to capture the data distribution from sample z, and a discriminator model D computes the probability that a sample came from the data distribution rather than generative model distribution. Basically, the Generator focuses on generating realistic images and then send to the Discriminator to decide whether the image is fake or real.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log(D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))], \qquad (2.12)$$

The Equation 2.12 depicts that if input of Discriminator comes from true data distribution, then the output of D(x) should be 1 for maximizing the above function w.r.t D. On the other hand, if the images are generated by the Generator then D(G(z)) should be 1 to minimize the objective function w.r.t G. In a nutshell, the network trains to maximize parameters of Discriminator using Gradient Ascent and minimize the same parameters of Generator using Gradient Descent.
One of the popular models of GAN involving Convolutional Neural Network is DCGAN [77], which stands for Deep Convolutional Generative Adversarial Networks. This network takes input as a noise of random numbers simulated as uniform distribution and outputs an image of the desired shape. The network builds upon many convolutional, deconvolutional, and fully connected layers to map the input noise to the desired output image. This network trains using mini-batch stochastic gradient descent tuned with hyperparameters. Though the training the network is time-consuming and require enormous computation power, generators create realistic vector arithmetic pattern using which images manipulates creatively.

2.3 Conclusion

Deep learning plays a vital role in fast-growing machine learning technology. The accelerated use of deep learning algorithms in diverse fields demonstrates the importance of this technology and the movement towards future advancement. Besides, it is essential to note that the hierarchy of layers is the main factor in creating a productive application of profound learning [11]. Back propagation-based learning has

become a paradigm to train hierarchical neural networks as it is quite efficient to get optimal weight. However, the training consumes excessive time than required, which has been a significant dilemma for many applications. Various kinds of approaches proposed to upgrade the back-propagation algorithm to get optimal gradient descent are not efficient enough to reduce time complexity. Moreover, the latest data indicates that network depth is critical because deeper NN's classification findings are more reliable than shallow ones. However, unrestricted network depth carries tremendous computing costs with little efficiency boost. According to a recent survey [86], a quest for solution-computing, perturbation-resistant, low-complexity NNs that can be represented by a few bits of information that reduce time complexity and boost network performance in both supervised and unsupervised learning. So, we propose an event-driven, resource-efficient, and quantized hierarchical network that can solve the high variance of gradient descent problem and provide excellent generalization performance without consuming lots of time. Moreover, we compare our model with other hierarchical networks with the same dataset to compare our model efficiency.

Chapter 3

Online Sequential Single Layer Classifier with Sub-Network Nodes

3.1 Abstract

The non-iterative pseudo-inverse matrix-based neural network training algorithm can achieve prominent generalization performance and, most importantly, much faster than many other popular machine learning algorithms, including the iterative learning based neural network. Over the last couple of decades, several variations of the non-iterative neural network training algorithms proposed to improve the performance in learning patterns from the data, such as: extreme learning machine (ELM) [43], random vector functional link network (RVFL) [74], Moore-Penrose Inverse [79]. The

hidden nodes in these types of NN generated randomly, and output weights are determined analytically. The random feature mapping is the key factor in measuring the stability of these types of NN, which depends on hidden nodes. However, there is no proper way to choose an optimal number of hidden nodes and activation functions to ensure the high quality feature. Moreover, one needs enormous computing resources to train the neural network as it does not support iterative training in nature. In this chapter, we exploit a non-iterative learning strategy with online sequential for single layer network with sub-network nodes that can optimize number of hidden nodes. The experimental results show that the proposed approach achieves satisfactory classification accuracy on many classic machine learning benchmark datasets with inadequate time consumption.

3.2 Introduction

From the past few years, the neural network has become an essential part of the machine learning family by contributing some benchmark methods such as autoencoder [92], Deep Belief Network [38], recurrent neural network [71], non-iterative hierarchical network [102] and so on [33, 66, 67, 93] for learning the intricate structure of high-dimensional data and object classification. The training time of these neural networks is generally time-consuming, which has been a significant bottleneck for many applications. As input transferred through higher or same dimensional space in each layer, the number of optimization parameters increases rapidly. This rapid increase is primarily the result of the underlying stochastic gradient method used. As a result, NN requires a lot of computation power and time to train the network. In fact, after consuming a tremendous amount of time, the output error of the network will stop or reduce slowly. For optimizing the parameters, a sequential training algorithm conserves information from input and train it sequentially. In [53], Kim proved through experiments that sequential training achieved a lower validation loss than the full training. This experimental result drives as a motivation to work with sequential training. For validating that claim, an experiment has been conducted on full training and our proposed sequential learning algorithm to compare the validation loss, which visualized in Figure 3.1. From this experiment , we can see that the validation error of sequential learning decreases rapidly with learning epochs. In [22], a brief review has been provided about sequential learning and explained a problem faced by it named “catastrophic forgetting” (CF). It is a consequence of input data

distribution overlapping [21]. For that reason, it disturbs the subsequent learning by eliminating information about previously learned behavior. New learning changes the weights of previously learned inputs and generates the wrong outputs. As a result, the error of the old information increases catastrophically after adding new learning, and forgetting occurs. Pseudorehearsal is an efficient way to reduce CF with the benefit of relearning [82]. It restricts the changes in previously learned function so that new learning only influences local changes by keeping the rest of function unaffected. Huang et al. [64] proposed pseudorehearsal based SLFN by utilizing Moore-Penrose Inverse [79], which can reduce the training time of neural networks a thousand times and delivers better performance than some conventional methods. It builds upon one input layer which can receive the stimuli from external environments, single-hidden layer with some nodes, and one output layer that generates output to forward it to external environments. However, the hidden layer generated randomly with infinite numbers of hidden nodes, and its input weights randomly generated [42]. Various studies [1, 8, 64] show that an appropriate number of neurons gained by some optimization methods work well rather than heaps of hidden nodes. Instead of using random hidden nodes along with randomized input weight, Yang et al. proposed a non-iterative learning algorithm consisting of a hidden node within a sub-network, generated by calculation [100]. This subnetwork hidden nodes can be grown by itself in the process of pulling back the residual network error to hidden layers. As it is a non-iterative learning algorithm, one needs to load the whole huge dataset into the computer memory is infeasible. Driven by this fact, a motivation arises: **can we use online sequential learning to build our classifier for obtaining better performance?**

In particular, this chapter has the following contributions:

1. Our online sequential classifier with m sub-network nodes provides a similar or much better generalization performance without maintaining a large number of hidden nodes. It reflects in the experimental result, which shows a significant increase in training accuracy within less time.

2. Generalization performance of this network is not sensitive to regularization parameter C. C is a pre-defined constant and $range \in \{2^{-10}, 2^{-9}, ..., 2^{9}, 2^{10}\}$. For that reason, users need to follow trial-and-error method to get the optimal value of C. Any random value of C will not affect the generalization performance of OS-SN in the learning process.

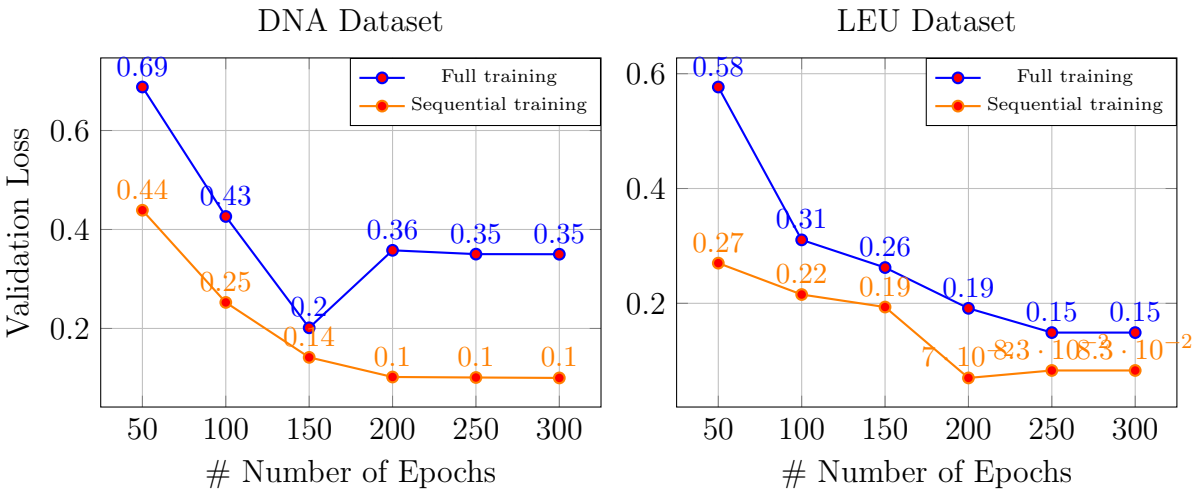

Figure 3.1: Comparison of Validation errors for the full training and the sequential training of DNA and LEU dataset where the x- and y-axis show the number of Epochs and average validation loss, respectively.

Table 3.1: Mathematical Notations.

Notation	**Definition**	**Shape**
α	hidden layer (encoding layer) weights	$d \times m$
b_α	hidden layer (encoding layer) biases	$1 \times m$
β	output layer (decoding layer) weights	$m \times o$
b_β	decoding layer biases	$1 \times m$
$mask$	neuron embedding layer	$d \times m$
X	input data	$n \times d$
$\hat{X}$	reconstructed input data	$n \times d$
Y	target data	$n \times o$
$\hat{Y}$	neural network output	$n \times o$
H	hidden layer output (encoding)	$n \times m$
$g(\cdot)$	activation function	N/A
c	a constant	1
d	input dimension	1
m	number of neurons in hidden layer	1
o	output dimension	1
n	number of samples	1
L	number of subnetworks	1
$\hat{\alpha_n}$	input weight of nth subnetwork	$d \times o$
β_n	output weight of nth subnetwork	$o \times o$
$h(\cdot)$	activation function	N/A
e_n	residual error of current subnetwork	$n \times o$

3.3 Method

The proposed method is designed for building an online sequential version of single layer classifier with sub-network nodes where the residual network error feed back into the network[100]. We define some mathematical notations in Table 3.1 for future use in this paper. In Table 3.1, $g(\cdot)$, c, d, m, o, and n are activation function which is the element-wise operation and other notations are all matrices.

3.3.1 Online Sequential Classifier with Sub-network Node

We would split the data along with target label data with n size and send it to the network for training. Firstly, we would use (x_0, t_0) chunk of data for the initial training of the network. Afterward, e_i represents the residual network error and $(\hat{\boldsymbol{\alpha}}_i, \hat{\boldsymbol{\beta}}_i)$ defines the input weight and output weight, which update in every iteration. For the first training batch $\boldsymbol{\alpha}_0$ (initial training phase), according to Eq. 2.2 and pseudo-inverse of the training sample, we get Eq. 3.1

$$\boldsymbol{\alpha}(0) = \mathbf{x}_0^{-1} \cdot h^{-1}(\mathbf{e}_0) = (\mathbf{x}_0^T\mathbf{x}_0 + (I_{d\times d})/c)^{-1}\mathbf{x}_0^T h^{-1}(\mathbf{e}_0), \tag{3.1}$$

We would use $\mathbf{m}_0$ to represent $\mathbf{x}_0^T\mathbf{x}_0 + (I_{d\times d})/c$. Now we can write Eq. 3.1 in following way:

$$\boldsymbol{\alpha}(0) = \mathbf{m}_0^{-1}\mathbf{x}_0^T h^{-1}(\mathbf{e}_0), \tag{3.2}$$

After the initial training, we would update the inputweight in a sequential manner with next batch of training samples $(\mathbf{x}_i, t_i)$. We combine $\mathbf{x}_0$ and $\mathbf{x}_1$ together as well as their corresponding residual error $\mathbf{e}_0$ and $\mathbf{e}_1$, theoretically, we can get

$$\hat{\boldsymbol{\alpha}_1} = \mathbf{m}_1^{-1} \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \end{bmatrix}^T \begin{bmatrix} h^{-1}(\mathbf{e}_0) \\ h^{-1}(\mathbf{e}_1) \end{bmatrix}, \tag{3.3}$$

where

$$\begin{aligned} \mathbf{m}_1 &= (I_{d\times d})/c + \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \end{bmatrix}, \\ &= (I_{d\times d})/c + \mathbf{x}_0^T\mathbf{x}_0 + \mathbf{x}_1^T\mathbf{x}_1, \\ &= \mathbf{m}_0 + \mathbf{x}_1^T\mathbf{x}_1, \end{aligned} \tag{3.4}$$

and

$$\begin{aligned}\begin{bmatrix}\mathbf{x}_0\\ \mathbf{x}_1\end{bmatrix}^T\begin{bmatrix}h^{-1}(\mathbf{e}_0)\\ h^{-1}(\mathbf{e}_1)\end{bmatrix} &= \mathbf{x}_0^T h^{-1}(\mathbf{e}_0)+\mathbf{x}_1^T h^{-1}(\mathbf{e}_1),\\ &= \mathbf{m}_0\mathbf{m}_0^{-1}\mathbf{x}_0^T h^{-1}(\mathbf{e}_0)+\mathbf{x}_1^T h^{-1}(\mathbf{e}_1),\\ &= \mathbf{m}_0\boldsymbol{\alpha}_{(0)}+\mathbf{x}_1^T h^{-1}(\mathbf{e}_1),\\ &= (\mathbf{m}_1-\mathbf{x}_1^T\mathbf{x}_1)\boldsymbol{\alpha}_{(0)}+\mathbf{x}_1^T h^{-1}(\mathbf{e}_1),\\ &= \mathbf{m}_1\boldsymbol{\alpha}_{(0)}-\mathbf{x}_1^T\mathbf{x}_1\boldsymbol{\alpha}_{(0)}+\mathbf{x}_1^T h^{-1}(\mathbf{e}_1),\end{aligned} \tag{3.5}$$

According to Eq. 3.3, Eq. 3.4, and Eq. 3.5, we derive

$$\begin{aligned}\hat{\boldsymbol{\alpha}}_1 &= \mathbf{m}_1^{-1}\left[\mathbf{m}_1\boldsymbol{\alpha}_{(0)}-\mathbf{x}_1^T\mathbf{x}_1\boldsymbol{\alpha}_{(0)}+\mathbf{x}_1^T h^{-1}(\mathbf{e}_1)\right],\\ &= \boldsymbol{\alpha}_{(0)}-\mathbf{m}_1^{-1}\mathbf{x}_1^T\mathbf{x}_1\boldsymbol{\alpha}_{(0)}+\mathbf{m}_1^{-1}\mathbf{x}_1^T h^{-1}(\mathbf{e}_1),\\ &= \boldsymbol{\alpha}_{(0)}+\mathbf{m}_1^{-1}\mathbf{x}_1^T\left[h^{-1}(\mathbf{e}_1)-\mathbf{e}_1\boldsymbol{\alpha}_{(0)}\right],\end{aligned} \tag{3.6}$$

We can generalize Eq. 3.6 to

$$\boldsymbol{\alpha}_{(\hat{i+1})} = \boldsymbol{\alpha}_{(i)}+\mathbf{m}_{i+1}^{-1}\mathbf{x}_{i+1}^T\left[h^{-1}(\mathbf{e}_{i+1})-\mathbf{x}_{i+1}\boldsymbol{\alpha}_{(i)}\right], \tag{3.7}$$

Instead of calculating inputweight $\hat{\alpha_n}$ for each chunk of data, we can use the Eq. 3.7 as a previous knowledge of inputweight to update the weight of new chunk of data. We would train our OS-Subnetwork in the following manner:

Algorithm 1 Subnetwork training algorithm

Result: Trained $\hat{\boldsymbol{\alpha}_n}$, $\hat{\boldsymbol{\beta}}_n$,
Consider $(\mathbf{x}_i, {t_i}^N)$ as training dataset $L = 1$ and $e = t$
while $L < L_{max}$ **do**
  calculate ($\hat{\boldsymbol{\alpha}_L}$,$\hat{\boldsymbol{\beta}}_L$) with Eq. 3.2 and 4.1
  calculate $\mathbf{e}_L$ with Eq. 2.5;
  $L = L + 1$;
end

Suppose, for initial training is ($\mathbf{x}_{init}$,$\mathbf{t}_{init}$) and remaining data would consider for sequential training ($\mathbf{x}_{seq}$,$\mathbf{t}_{seq}$). If total numbers of training epochs is $Total\_Epochs$ and batch size for sequential data is $BATCH\_SIZE$.

Algorithm 2 OS-Subnetwork training algorithm

Result: Update $\boldsymbol{\alpha}_L$ sequentially and calculate the corresponding $\boldsymbol{\beta}_L$

Split the dataset for initial and sequential training

$epoch \leftarrow 0$;

For($\mathbf{x}_{init}$,$\mathbf{t}_{init}$, we obtain $\boldsymbol{\alpha}_L$ and $\boldsymbol{\beta}_L$ through Algorithm 1 for each subnetwork

while $epoch < Total\_Epochs$ **do**

  $l \leftarrow 0$

  ; **while** $l < length(x_{seq})$ **do**

    if $l + BATCH\_SIZE \leq length(x_{seq})$ **then**

      $\mathbf{x}_{batch} \leftarrow \mathbf{x}_{seq}[l : l + BATCH\_SIZE]$;

      $\mathbf{t}_{batch} \leftarrow \mathbf{t}_{seq}[l : l + BATCH\_SIZE]$;

    else

      $\mathbf{x}_{batch} \leftarrow \mathbf{x}_{seq}[l :]$;

      $\mathbf{t}_{batch} \leftarrow \mathbf{t}_{seq}[1 :]$;

    end

    $l \leftarrow l + BATCH\_SIZE$;

    Update sequentially $\boldsymbol{\alpha}_L$ through Eq. 3.7 and calculate $\boldsymbol{\beta}_L$ for ($\mathbf{x}_{batch}$,$\mathbf{t}_{batch}$) following 1;

  end

  $epoch \leftarrow epoch + 1$;

end

3.4 Experimental Results

We choose dataset from a diverse environment to measure the efficiency of our proposed classifier. For that reason, we would describe the dataset and compare our proposed classifier with other classifiers in this section,. Moreover, we use the mean classification accuracy as one of the evaluation metrics to test the performance.

3.4.1 Datasets

For image data classification, we select six databases which described in Table 3.3 two datasets related to numeric values are mnist and USPS. Afterward, Ollivetti for

Table 3.2: Tabular Datasets.

Dataset	**#Features**	**#Train**	**#Test**	**#Category**
Mushroom	256	7291	2007	2
Acoustic	51	40000	58000	3
Hill Valley	101	606	606	2
Protein	357	17766	6621	2
Duke	7129	29	15	2
Leukemia	7129	38	34	2
DNA	180	1046	1186	3
Credit Card Fraud Detection(CCFD)	30	21100	64807	2

Table 3.3: Image Datasets.

Dataset	**Training Image**	**Testing Image**	**Category**
USPS	7291	2007	10
Mnist	60000	10000	10
Olivetti Face	200	200	40
Satimage	4435	2000	6
Cifar10	50000	10000	10
Cifar100	50000	10000	100
Scene15	2250	1610	15
Caltech101	6000	3144	102
Caltech265	20560	10047	257

face dataset, Cifar10 and Cifar100 dataset are related to objects and Scene15 [20] are based on Category scene.

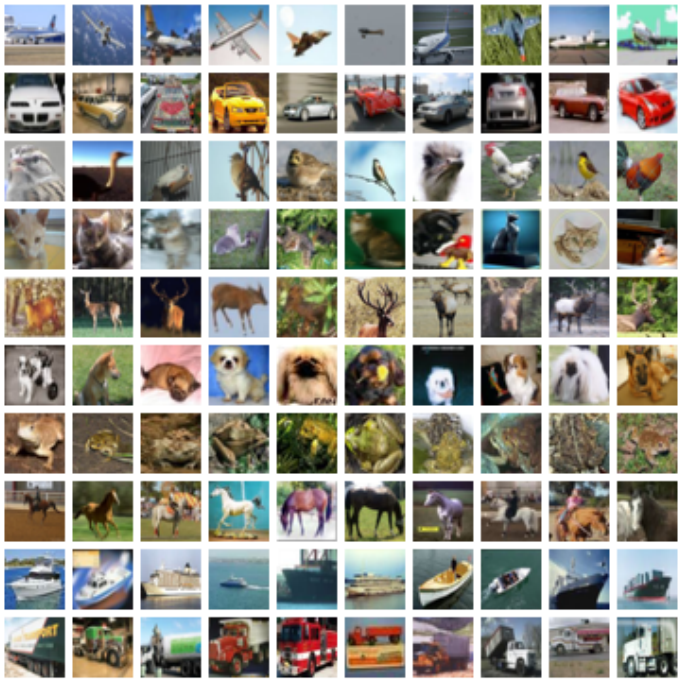

Figure 3.2: Contents of Encoding Layer Weight of Olivetti Face Dataset From Top to Bottom with the Dimension of 100, 40 and 10 respectively

mnist/USPS datasets both are hand-written dataset where mnist has a training set of 60,000 examples and a test set of 10,000 examples. The digits have centered in a fixed-size, which is 28x28 image. USPS dataset are 16X16 grayscale pixels were 7291 for training and 2007 for testing.

Olivetti Face dataset has ten different images of each 40 distinct persons. The images were taken by varying facial expressions (such as open or closed eyes, smiling or not smiling), and facial details (glasses or no glasses). We considered five images of each person in total of 200 images for training and rest for testing purposes.

Cifar10/100 datasets contain a natural colored image with 32x32 pixels. Cifar10 consists of 60,000 color images in 10 categories, including airplane, bird, auto-mobile,cat, dog, frog, deer, ship, horse and truck. Cifar100 dataset also builds upon 60000 color images of 100 categories. Both datasets have been split where 50000 images for training and remaining for testing. In Cifar10 dataset 5000 data per class and cifar100 dataset with 500 data per class has been used.

Scene15 dataset is a dataset containing 4486 gray-scale images where 3860 images

Table 3.4: Performance Comparison on Classification Problems (Mean: Average Testing Accuracy Training Time)

Dataset	**ELM**			**SVM**		**RF**		**OS-SN**		
	Accuracy	**Time(s)**	**nodes**	**Accuracy**	**Time(s)**	**Accuracy**	**Time(s)**	**Accuracy**	**Time(s)**	**SN**
Hill	76.25%	0.1647	500	58.67%	0.1295	56.90%	8.59	**92.27%**	**0.011**	2
Mushrooms	88.91%	0.9147	500	89.17%	38.62	52.90%	8.59	**89.27%**	**0.11**	2
Mnist	91.06%	8.46	500	80.58%	478.59	**96.78%**	403.59	85.80%	**0.68**	3
USPS	**93.54%**	2.08	500	94.65%	146.49	**93.79**%	148.78	86.25%	**1.83**	2
Duke	79.32%	0.84	500	86.36%	**0.159**	79.58%	20.89	**97.05%**	0.44	3
Leu	77.06%	9.46	500	83.58%	2.39	58.58%	15.78	**89.08%**	**0.81**	3
DNA	91.25%	0.215	500	92.90%	0.46	**93.52%**	4.42	92.58%	**0.031**	3
Protein	67.09%	5.15	500	51.18%	253.46	68.23%	222.59	**75.18%**	**0.119**	3
CCDF	97.98%	96.78	1000	91.13%	600.87	92.69%	567.36	**98.56%**	**2.39**	3

Table 3.5: Performance Comparison on Image Classification Problems with Imagenet pre-trained Vgg16 deep features.

Dataset	Vgg16	SVM		OS-SN	
	Accuracy	Accuracy	Time(sec)	Accuracy	Time(sec)
Caltech101	92.1	88.2%	112.42	**89.52%**	**4.45**
Caltech256	73.2	75.17%	14.12	**78.22%**	**1.36**
Scene15	92.4	86.63%	18.26	**92.53%**	**1.07**
CIFAR10	94.21	**95.49%**	103.94	94.88%	**2.05**

came from 15 -category scenes. For our experiments, we randomly select 150 images per category as training data and remaining for testing data.

Caltech101 dataset contains 9144 images of 102 types of object categories. Each category holds a various range of images from 31 to 800. For our experiment, we train 15 on 15 and 30 random samples per category and use rest for testing purpose.

Caltech256 dataset contains 30607 images of 257 object categories. Each category holds atleast 80 images per category. Compared to Caltech101 dataset, this dataset contains large variations in object location and size.

3.4.2 Image Classification

In Table 3.4 a performance comparison has visualized among popular ELM, Support Vector Machine(SVM), Random Forest(RF), and our proposed OS-SN algorithm. It indicates that our algorithm provides more accuracy compared with opponent methods. To demonstrate it properly, we consider one medium-sized and another large dataset, such as Mushroom and Protein datasets. For the Mushroom dataset, our OS-SN algorithm provides higher accuracy in 9 times, 350 times and 80 times faster than ELM, SVM, and RF, respectively. On the other hand, our OS-SN algorithm provides higher accuracy in 50 times, 2600 times, and 2000 times faster than ELM, SVM and RF, respectively. Though the OS-SN algorithm performs well in the Duke dataset, it consumes maximum time rather than other algorithms. Duke is a small but high dimensional (7129) dataset. For training this dataset through OS-SN, we split it for initial and sequential training. Moreover, we split the data into three batches. That's why it takes time. To reduce the time consumption, we provide the data in one batch for sequential training. Though it takes less time, testing accuracy degrades to 94%.

Our proposed hierarchical network achieves higher categorization accuracy in com-

plex image datasets. Such as : Scene15, Cifar10, Caltech101 and Caltech256. The experiments conducted in a workstation with 128 GB memory and one Geforce 1080 TI GPU. We use 100,10,10 and 30 images of per class for training, respectively. The remaining parts utilizes for testing purposes. We perform all experiments by converting the color images into grayscale. Moreover, we have repeated all experiments 10 times by changing the permutation of whole training, and test images. We also add the state of art Vgg16 accuracy to check performance differences. The average classification rates recorded for each run. Initially, we use Vgg16 4096-dimensional features tuned with Imagenet pre-trained weight as an initial parameter and with end-to-end training on the target dataset to extract complex features. Afterward, we associate our network with that features to accumulate similar patterns of data together and classify their category. We compared our network with the SVM classifier trained by pre-trained deep features of VGG16. In Table 3.5, we can notice moderate accuracy within less time of our proposed network.

3.5 Conclusion

In this chapter, we proposed an online sequential classifier for object recognition. According to the experimental results, the following conclusions are: (1) it generates sub-network hidden nodes by pulling back neural network residual error to the hidden layers. (2) our network with m hidden nodes can achieve similar or better generalization performance than other learning methods with hundreds of hidden nodes. (3) It could significantly reduce the training time compared to other conventional learning methods including BP, SVM, and ELM. (4) Instead of using randomized input weights, we approach a new way where weights would be configured by calculation.

Chapter 4

Online Sequential Learning with Non-iterative Strategy for Multiple Layer Neural Network

---

4.1 Abstract

In this chapter, we are going to propose an online sequential multiple layer neural network where we build the first portion with an online sequential autoencoder to extract subspace features and map into the sub-network node for classification. A multi-layer autoencoder is efficient for feature extraction as well as plays a vital role in dimension reduction and image reconstruction. The experimental results show that

the proposed approach achieves satisfactory classification accuracy on many classic machine learning benchmark dataset with extremely low time consumption.

4.2 Introduction

The performance of machine learning algorithms is highly dependent on the quality of data representation (or features). For that reason, researchers are deploying machine learning algorithms for designing pipelines that can transform high dimensional data into meaningful feature descriptors that can support machine learning algorithms for object classification. Feature selection and extraction techniques designs to reduce dimension as it significantly increases the time and space requirements for processing the data [34]. The process of selecting the feature subset divides into two categories: hand-crafted features and machine learning-based features. The primitive one includes a spatial pyramidal feature (SPF) [60], and scale-invariant feature transform (SIFT) [68]. The latter one mainly refers to the feature extraction techniques, involving supervised, unsupervised, or semi-supervised algorithms, based on the kind of label information provided. Supervised feature extraction methods typically focus on the correlation between feature and class labels, and as a result, it requires a large amount of labeled training data. From the past few years, some unsupervised feature extraction techniques gained popularity for projecting the data points from the dimensions of maximum variances and map it into a low dimension such as Stochastic Neighbor Embedding (SNE) [39], t-Distributed Stochastic Neighbor Embedding (t-SNE) [70], and deep neural network-based autoencoders. They work well in dimension reduction, data visualization, and signal processing. However, a significant problem among the methods mentioned above is consuming excessive training time for back-propagation based learning. To reduce the time complexity faced by backpropagation, Yang et al. proposed a non-iterative learning multi-hidden layer autoencoder [99] to learn the deep features which work well rather than other non-iterative algorithms. If we retrospect on the previous studies [4, 37], they state that a multi-layer neural network with iterative learning performs well in the field of feature representation. Because, iterative based learning converges the particular value of weight that for which the loss is minimum. This point is called minima for the loss function. However, this process is slow and requires to be cautious in setting the learning rate so that network can reach near to the global minimum. On the other hand, non-iterative algorithms support online sequential for training only a single layer network. If we

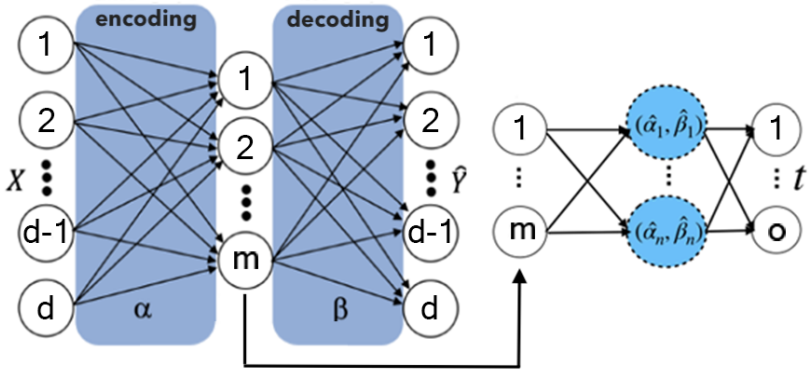

Figure 4.1: Proposed framework including feature extractor and classifier.

need to train the multi-layer networks with a non-iterative learning strategy, we need to use a significant portion of memory. When we have a big dataset, the existed methods all fail to train a such big one. From the previous chapter we noticed that an online sequential network could deliver better performance than the other iterative learning algorithms. Driven by this fact, a motivation arises: if we introduce online sequential learning in non-iterative autoencoder and concatenate with our proposed classifier, can we increase the efficiency of the network to obtain better performance? In particular, this chapter has the following contributions:

1. An online sequential autoencoder has been proposed with a non-iterative strategy that can process data chunk-by-chunk, but there is no iterative tuning required inside the hidden layer to configure any predefined parameters [12]. Instead of using randomized input and output weight, it analytically determines proper value of weight for which the loss would be minimum based on sequentially arrived data. Thus, it reduces computational workloads dramatically and uses less memory.

2. An online sequential version of a multi-layer neural network has been proposed which is build upon a feature extractor and a classifier (see Figure 4.1), which can process large dataset without possessing a significant proportion of memory. We use OS-AE as a feature extractor and OS-SN for classification.

3. We evaluated the approach we proposed in terms of accuracy and time consumption. The experiment's results show that our method delivers much better performance and faster training speed than most of the other iterative and non-iterative network.

4.3 Method

The proposed method is an online sequential version of a multi-layer neural network where training data will go through autoencoder to extract deep features, and based on these target features, sub-network nodes are going to be trained iteratively to provide better performance with fewer hidden nodes. In Figure 4.1, we visualized how our network process the training data may arrive chunk-by-chunk to the Autoencoder and reduce the dimension of input data (d) to its target value (m). Based on these target features, sub-network nodes are going to be trained iteratively to provide better performance with fewer hidden nodes.

4.3.1 Online Sequential Autoencoder

In this section, we give the mathematical derivation and a detailed description of the proposed algorithm.

We use $\mathbf{X}_i$ to represent the i^{th} training batch of samples (in autoencoder, $Y_i = X_i$). Accordingly, $\mathbf{H}_i$ represents the encoding layer output regarding the input $\mathbf{X}_i$. Here we follow the mathematical equation of OS-ELM [64]. The batch size is n. We use $\alpha^{(i)}$, $\boldsymbol{\beta}^{(i)}$ to represent the encoding weights and decoding weights updated after the training on the i^{th} training batch. For the first training batch H_0 (initial training phase), according to Eq. 2.8 and Eq. 2.9, we get Eq. 4.1

$$\boldsymbol{\beta}^{(0)} = \mathbf{H}_0^{\dagger} g^{-1}(\mathbf{X}_0) = (\mathbf{H}_0^T \mathbf{H}_0 + (I_{m\times m})/c)^{-1} \mathbf{H}_0^T g^{-1}(\mathbf{X}_0), \tag{4.1}$$

We use $\mathbf{K}_0$ to represent $\mathbf{H}_0^T \mathbf{H}_0 + (\mathbf{I}_{m\times m})/c$. Start from the arrival of the second training batch of samples $\mathbf{X}_1$ is the sequential training phase. If we combine $\mathbf{X}_0$ and $\mathbf{X}_1$ together, theoretically, we can get

$$\boldsymbol{\beta}^{(1)} = \mathbf{K}_1^{-1} \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{g}^{-1}(\mathbf{X}_0) \\ \mathbf{g}^{-1}(\mathbf{X}_1) \end{bmatrix}, \tag{4.2}$$

where

$$\begin{aligned}\mathbf{K}_1 &= (\mathbf{I}_{m\times m})/c + \begin{bmatrix}\mathbf{H}_0\\ \mathbf{H}_1\end{bmatrix}^T \begin{bmatrix}\mathbf{H}_0\\ \mathbf{H}_1\end{bmatrix},\\ &= (\mathbf{I}_{m\times m})/c + \mathbf{H}_0^T\mathbf{H}_0 + \mathbf{H}_1^T\mathbf{H}_1,\\ &= \mathbf{K}_0 + \mathbf{H}_1^T\mathbf{H}_1,\end{aligned} \tag{4.3}$$

and

$$\begin{aligned}\begin{bmatrix}\mathbf{H}_0\\ \mathbf{H}_1\end{bmatrix}^T \begin{bmatrix}g^{-1}(\mathbf{X}_0),\\ g^{-1}(\mathbf{X}_1)\end{bmatrix} &= \mathbf{H}_0^T g^{-1}(\mathbf{X}_0) + \mathbf{H}_1^T g^{-1}(\mathbf{X}_1),\\ &= \mathbf{K}_0\mathbf{K}_0^{-1}\mathbf{H}_0^T g^{-1}(\mathbf{X}_0) + \mathbf{H}_1^T g^{-1}(\mathbf{X}_1),\\ &= \mathbf{K}_0\boldsymbol{\beta}^{(0)} + \mathbf{H}_1^T g^{-1}(\mathbf{X}_1),\\ &= (\mathbf{K}_1 - \mathbf{H}_1^T\mathbf{H}_1)\boldsymbol{\beta}^{(0)} + \mathbf{H}_1^T g^{-1}(\mathbf{X}_1),\\ &= \mathbf{K}_1\beta^{(0)} - \mathbf{H}_1^T\mathbf{H}_1\boldsymbol{\beta}^{(0)} + \mathbf{H}_1^T g^{-1}(\mathbf{X}_1),\end{aligned} \tag{4.4}$$

According to Eq. 4.2, Eq. 4.3, and Eq. 4.4, we derive

$$\begin{aligned}\boldsymbol{\beta}^{(1)} &= \mathbf{K}_1^{-1}\left[\mathbf{K}_1\boldsymbol{\beta}^{(0)} - \mathbf{H}_1^T\mathbf{H}_1\boldsymbol{\beta}^{(0)} + \mathbf{H}_1^T g^{-1}(\mathbf{X}_1)\right]\\ &= \boldsymbol{\beta}^{(0)} - \mathbf{K}_1^{-1}\mathbf{H}_1^T\mathbf{H}_1\boldsymbol{\beta}^{(0)} + \mathbf{K}_1^{-1}\mathbf{H}_1^T g^{-1}(\mathbf{X}_1),\\ &= \boldsymbol{\beta}^{(0)} + \mathbf{K}_1^{-1}\mathbf{H}_1^T\left[g^{-1}(\mathbf{X}_1) - \mathbf{H}_1\boldsymbol{\beta}^{(0)}\right],\end{aligned} \tag{4.5}$$

We can generalize Eq. 4.5 for upcoming any chunk of data and continuously update output weight until all data has been utilized by Eq. 4.6

$$\boldsymbol{\beta}^{(i+1)} = \boldsymbol{\beta}^{(i)} + \mathbf{K}_{i+1}^{-1}\mathbf{H}_{k+1}^T\left[g^{-1}(\mathbf{X}_{i+1}) - \mathbf{H}_{i+1}\boldsymbol{\beta}^{(i)}\right], \tag{4.6}$$

In OS-ELM, encoding layer weights would generate randomly for the next chunk of data, and as a result, the network has to train again. In Figure 5.4, we visualized the encoding and decoding layer weight of OS-ELM that triggered to copy the decoding layer weight $\boldsymbol{\beta}^i$ to encoding layer weight $\alpha^{(i+1)}$ in our proposed algorithm, which reduce time consumption as well as accuracy.

$$\boldsymbol{\alpha}^{(i+1)} = (\boldsymbol{\beta})^T, \tag{4.7}$$

Because $\boldsymbol{\beta}^{(i+1)}$ depends on both $\boldsymbol{\beta}^{(i)}$ and X_{i+1}, one can learn $\boldsymbol{\beta}$ without losing the

"previous knowledge". Assume the batch of samples for initial training is X_{init}; the batch of samples for sequential training is X_{seq}; the total number of training epochs is $Total\_Epochs$; the sequential training batch size is $BATCH\_SIZE$. We define the proposed algorithm in Algorithm 3. Once the training finished, one can encode the data X through $H = g(X\alpha)$.

Algorithm 3 OS-Autoencoder training algorithm

Result: Trained α, $\boldsymbol{\beta}$, b_α, and $b_{\boldsymbol{\beta}}$

randomly initialize α according to Eq. 2.6

$epoch \leftarrow 0$;

given X_{init} obtain $\boldsymbol{\beta}$ through Eq. 4.1;

while $epoch < Total\_Epochs$ **do**

  $l \leftarrow 0$;

  while $l < length(X_{seq})$ **do**

    $\alpha \leftarrow \boldsymbol{\beta}^T$;

    if $l + BATCH\_SIZE \leq length(X_{seq})$ **then**

      $X_{batch} \leftarrow X_{seq}[l : l + BATCH\_SIZE]$;

    else

      $X_{batch} \leftarrow X_{seq}[l :]$;

    end

    $l \leftarrow l + BATCH\_SIZE$;

    according to X_{batch}, obtain $\boldsymbol{\beta}$ through Eq. 4.6 ;

  end

  $epoch \leftarrow epoch + 1$;

end

The learning process used in this algorithm includes continuous update of the weights results in error between the inputs and the reconstructed outputs of the network are gradually decreased until it reach to its threshold.

The performance of the proposed algorithm depends on both the number of hidden neurons m (encoding dimension) and the pre-defined constant c. We train the network on Caltech101 dataset with different settings of $m \in \{10, 20, 30, ..., 200\}$ and $c \in \{2^{-10}, 2^{-9}, ..., 2^9, 2^{10}\}$, the result is shown in Fig. 4.2. The proper way of selecting the

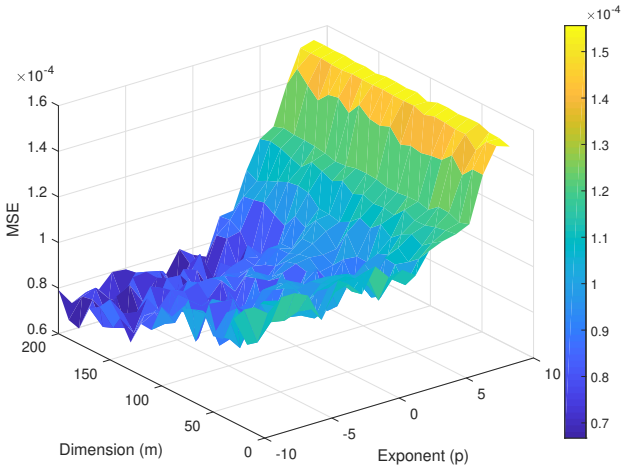

Figure 4.2: The mean squared error (MSE) on testing dataset with different m and c settings ($c = 2^P$).

optimal value of c is chosen by trial-and-error method [45].

4.4 Experimental Results

In this section, We would describe the dataset and compare our proposed algorithm with other algorithms. To test the performance, we first train some autoencoders to reduce the dimension of the original dataset, then train a classifier on the encoded dataset and use the mean classification accuracy as one of the evaluation metrics. To test the efficiency of our algorithm, we compared our network with other algorithm are in the following order:

1. Our proposed Algorithm;
2. Autoencoder [99];
3. t-distributed SNE (t-SNE) [19];
4. Stochastic neighbor embedding (SNE)[36];
5. Linear Graph Embedding (LGE)[65];
6. Deep Autoencoder (DAE) [37];

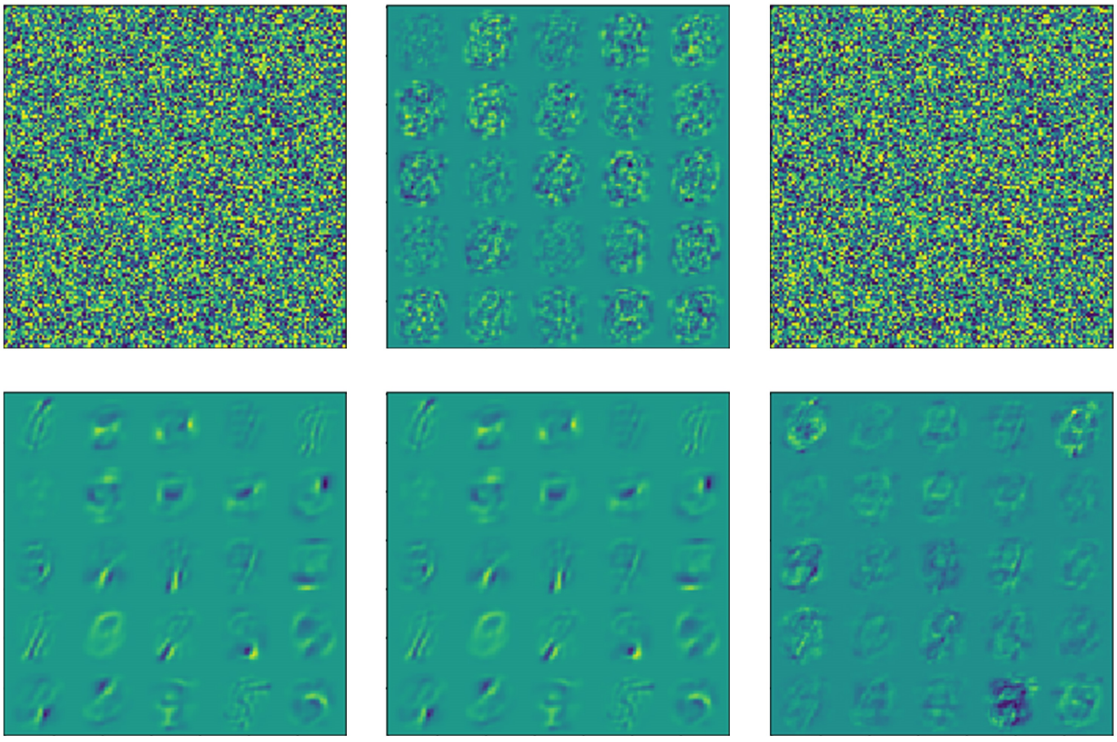

Figure 4.3: Visualized encoding layer weights (first column), decoding layer weights (middle column), and the difference between encoding layer weights and encoding layer weights (last column). Each image contains the reshaped weights of the first 25 neurons. The first row represents OS-ELM; the second row represents our proposed algorithm.

Table 4.1: Performance Comparison (Mean: Average Testing Accuracy Training Time) of Dimension Reduction

Dataset	**Dimension**	**DAE**		**ML-AE**		**AE**		**OS-AE**	
		Accuracy	**Time(s)**	**Accuracy**	**Time(s)**	**Accuracy**	**Time(s)**	**Accuracy**	**Time(s)**
Hill	$101 \rightarrow 2$	51.25%	0.82	88.78%	0.69	49.08%	0.27	**51.65**%	**0.11**
Satimage	$36 \rightarrow 26$	**90.23%**	3.33	73.98%	0.97	86.00%	1.45	83.27%	**0.9**
USPS	$256 \rightarrow 10$	91.05%	7.88	89.96%	1.87	**92.08**%	2.78	87.79%	**1.14**
Duke	$7129 \rightarrow 2$	50.64%	0.89	48.87%	3.72	61.55%	0.28	**62.89**%	**0.118**
Leu	$7129 \rightarrow 10$	60.29%	0.57	60.96%	6.87	**76.68**%	0.67	71.89%	**0.114**
DNA	$180 \rightarrow 2$	60.54%	1.93	56.65%	0.42	73.48%	0.16	**78.87**%	**0.026**
Protein	$357 \rightarrow 2$	**55.29**%	28.45	46.56%	4.98	49.24%	4.08	47.00%	**3.45**
Olive Face	$4096 \rightarrow 40$	78%	2.88	84.48%	1.75	**92.08**	0.52	90.87%	**0.126**
Acoustic	$50 \rightarrow 2$	45.65%	60.32	69.98%	0.78	**64.10**%	1.75	56.87%	**0.36**

Table 4.2: Performance Comparison (Mean: Average Testing Accuracy Training Time) of Dimension Reduction

Dataset	**Dimension**	**SNE**		**LGE**		**t-SNE**		**OS-AE**	
		Accuracy	**Time(s)**	**Accuracy**	**Time(s)**	**Accuracy**	**Time(s)**	**Accuracy**	**Time (s)**
Hill	$101 \rightarrow 2$	51.25%	223.48	52.88%	**0.09**	51.23%	61.23	**51.65**%	0.11
Satimage	$36 \rightarrow 26$	89.75%	3823.67	82.98%	332.67	**89.23**%	691.21	83.27%	**0.9**
USPS	$256 \rightarrow 10$	93.75%	1827.80	87.96%	6.87	**95.28**%	2991.78	87.79%	**1.14**
Duke	$7129 \rightarrow 2$	51.74%	2.798	48.37%	0.72	**73.33**%	1.88	62.89%	**0.118**
Leu	$7129 \rightarrow 10$	71.75%	24.45	47.96%	0.87	**79.68**%	4.98	71.89%	**0.114**
DNA	$180 \rightarrow 2$	61.74%	702.78	61.65%	0.32	59.48%	216.75	**78.87**%	**0.026**
Protein	$357 \rightarrow 2$	44.25%	61997.45	46.96%	288.98	**48.68**%	17724.98	47.00%	**3.45**
Olive Face	$4096 \rightarrow 40$	79.84%	12.78	80.25%	0.48	69.48%	216.75	**90.87**%	**0.126**
Acoustic	$50 \rightarrow 2$	56.65%	428.32	**59.48**%	216.75	un-define	infinite	56.87%	**0.36**
CCFD	$30 \rightarrow 15$	un-define	infinite	89.28%	427.66	un-define	infinite	**98.88**%	**4.66**

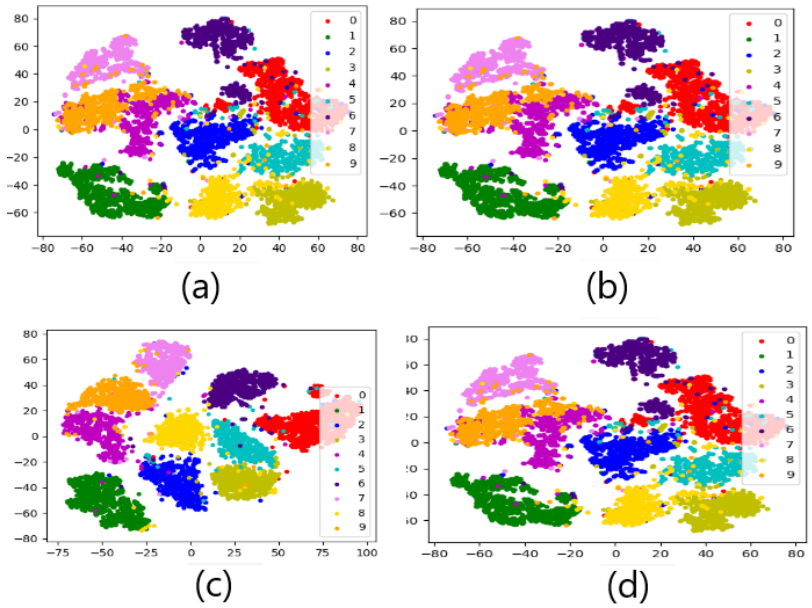

Figure 4.4: Visualizing the Performance Comparison of Features of USPS Dataset (a) Deep Autoencoder, (b) t-SNE, (c) PCA and (d) OS-Autoencoder respectively

7. Unsupervised Mulilayer Autoencoder with subnetwork nodes(ML-AE)[101].

The algorithms mentioned here are used for dimension reduction and ELM with 1000 hidden nodes has been used to classify from encoded dimension. The codes used for t-SNE, SNE, LGE are downloaded from internet.

4.4.1 Unsupervised Features Learning:

In Table 4.2, a performance comparison has been visualized among SNE, LGE,t-SNE, and our proposed algorithm. Furthermore, we carry out these experiments in Table 4.1 to compare our single-layer network with other multilayer neural network algorithms, such as Deep Autoencoder, MLNN-SN and Autoencoder. It indicates that our OS-Autoencoder can learn optimal features more accurately within less time compared with opponent methods. Let consider the Protein dataset (large datasets with medium dimension) and Leu dataset (medium samples with medium dimensions). Firstly in the Protein dataset, a significant increase in training speed has been noticed. Our approach is 20000, 80 and 6000 times faster than SNE, LGE and t-SNE. Secondly, in the Leu dataset, the training speed of our approach is about 200,8 and 40 times faster than SNE, LGE, and t-SNE.

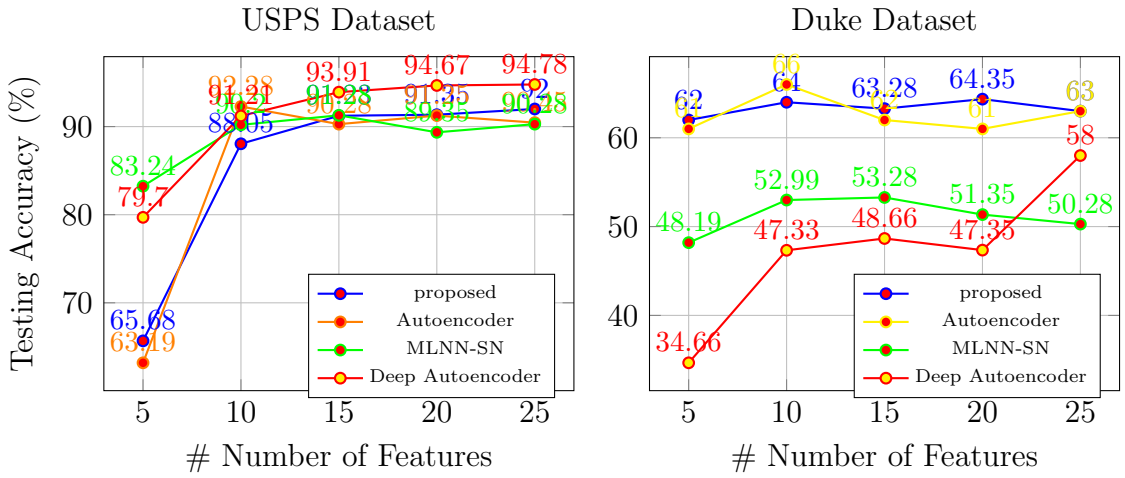

Figure 4.5: Generalization performance comparison on USPS and Duke dataset based on Autoencoder and our proposed method where the x- and y-axis show the number of features and average testing accuracy, respectively. Result on (a) USPS, (b) Duke.

Os-Autoencoder learns optimal features based on the number of feature selection. To examine those deep features, we visualize the encoding layer weights based on the number of features. In figure 4.6, we experiment with these interesting method on olivetti face dataset. The original dimension of the dataset is 4096, and we tested it by reducing the dimension into 100, 40, and 10. Our proposed autoencoder learn different features based on the reduced dimension so that optimal features accumulate together into the weights with the deeper compression of dimension.
In Fig 5.6 we want to visualize how the original high dimensional data acts after passing through different dimension reduction algorithm such as : deep autoencoder, tsne, pca and OS-Autoencoder on Usps dataset. The actual dimension of the dataset is 256 and we reduced it to 50. According to Fig 5.6 our OS-autoencoder clustered and distributed more precisely based on their features rather than other algorithms.

4.4.2 Image Reconstruction

To compare the image reconstruct quality of our algorithm, we compare with deep autoencoder, Deep Belief Network(DBN) and our proposed algorithm. We applied both training algorithms on the same neural network architecture (single hidden layer with 256 neurons) and trained the neural networks on the MNIST dataset and CIFAR10 [57] dataset.

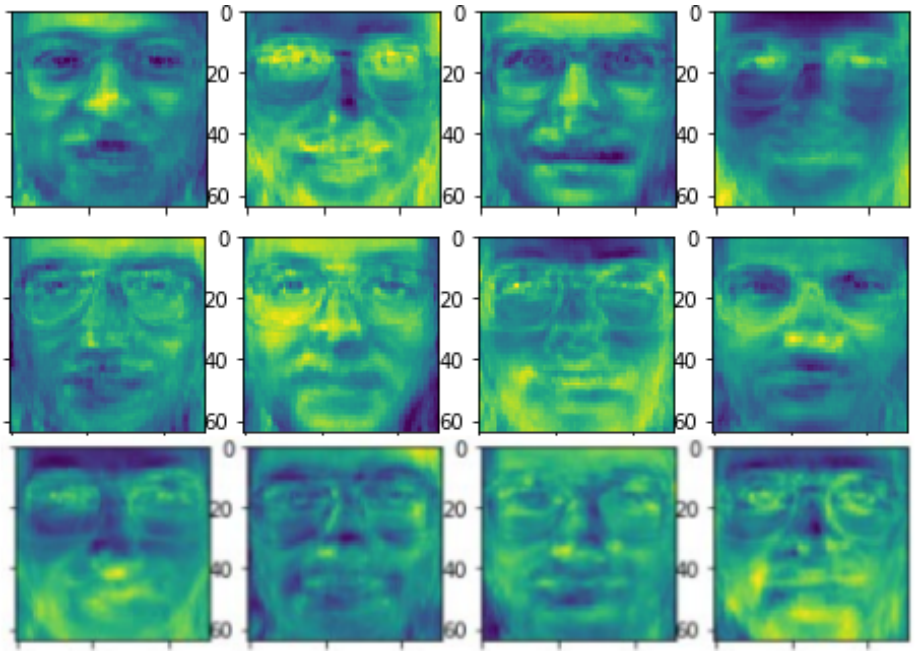

Figure 4.6: Contents of Encoding Layer Weight of Olivetti Face Dataset From Top to Bottom with the Dimension of 100, 40 and 10 respectively

For the MNIST dataset, the input dimension is $28 \times 28 = 784$. We trained deep autoencoder and OS-Autoencoder 30 epochs on MNIST dataset separately. We used two layers of belief network by stacking two Restricted Boltzmann Machine to train the Mnist dataset. Fig. 5.10 depicts that the quality of image reconstruction is much clear, which perceive that our proposed autoencoder can able to learn prominent features rather than other multi-layer autoencoders.

For the CIFAR10 dataset, we first converted the images from three channels RGB color space to one channel grayscale images. Therefore, the input dimension is $32 \times 32 = 1024$. We trained deep autoencoder and OS-Autoencoder 200 epochs on CIFAR10 dataset separately. The qualitative comparisons are shown in Fig. 5.11. The image reconstruction performance of our proposed OS-Autoencoder is better than deep autoencoder under the same experimental setting.

After the last training iteration on MNIST dataset, we visualized the encoding layer (hidden layer) weights, decoding layer (output layer) weights, and the difference of encoding layer weights and decoding layer weights (see Fig. 5.4). We can see that the encoding layer of deep autoencoder is random since only the decoding layer was trained. Although we also only trained the decoding layer, the encoding layer of our OS-Autoencoder contains some visual patterns because we copied the weights

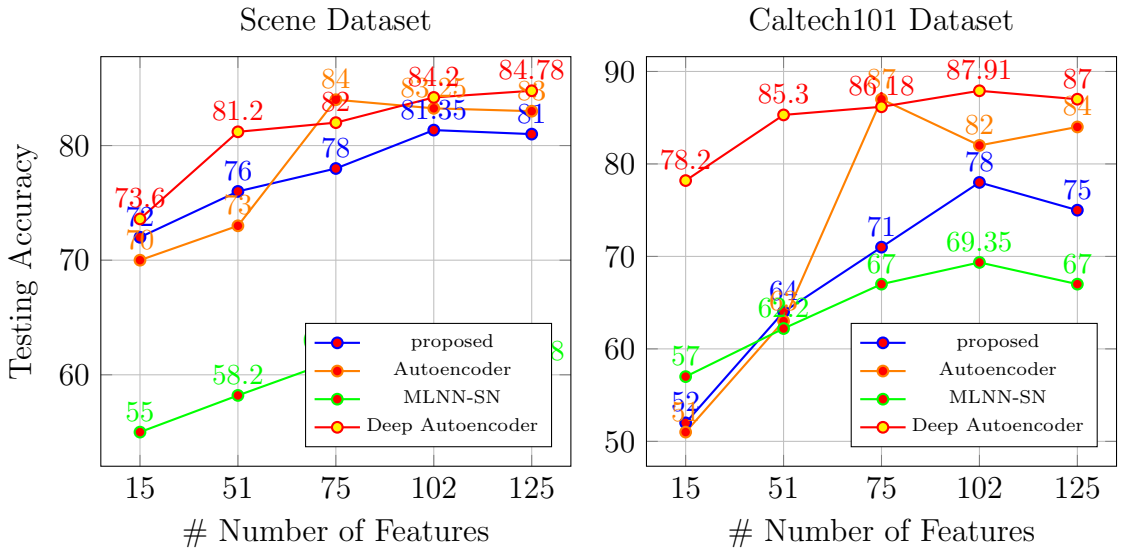

Figure 4.7: Generalization performance comparison on deep features (gathered by VGG16) of Scene15 and caltech101 dataset based on Autoencoder and our proposed method where the x- and y-axis show the number of features and average testing accuracy, respectively. Result on (a) Scene15, (b) Caltech 101.

of the decoding layer back to the encoding layer after each training step. Because we did not copy the weights after the last training step, there are some differences between the encoding layer weights and decoding layer weights. The learned patterns in the decoding layer of our OS-Autoencoder are more evident than deep autoencoder.

4.5 Conclusion and Future Work

In this chapter, we proposed an online sequential autoencoder that works well in the field of dimension reduction and feature extraction. Instead of using randomized input weights, we copy-back upgraded decoding layer weight for each iteration and reach the steepest descent in the value of the loss function iteratively without configuring the learning rate. Compared to other algorithms, our proposed method provides higher classification accuracy as well as faster training speed. In the future, we will extend the proposed algorithm to a hierarchical network for dimension reduction: to allow the network learns abstracted deep features.

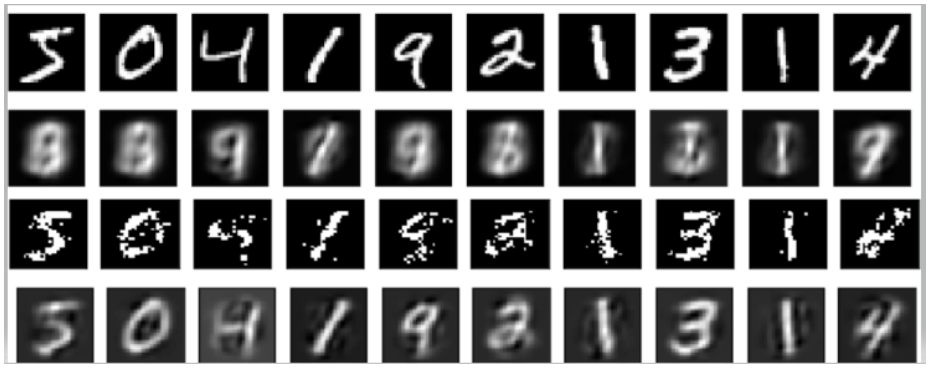

Figure 4.8: The qualitative comparisons of small image reconstruction performance of Deep Autoencoder, Deep Belief Network(DBN) and our proposed algorithm. The first three rows from top to bottom: some images randomly sampled from MNIST training set; the corresponding Deep Autoencoder reconstructed images; the corresponding DBN reconstructed images and OS-Autoencoder reconstructed images.

Figure 4.9: The qualitative comparisons of small image reconstruction performance of deep autoencoder and our proposed algorithm. From top to bottom: some images randomly sampled from CIFAR10 testing set; the corresponding deep autoencoder reconstructed images; the corresponding OS-Autoencoder reconstructed images.

Chapter 5

Extension of Online Sequential Autoencoder for Data Augmentation

5.1 Abstract

In recent years, iterative learning based generative models have gained popularity due to some incredible contribution in the field of Data Augmentation. Though a well-designed generative model can produce highly realistic data, it suffers from slow

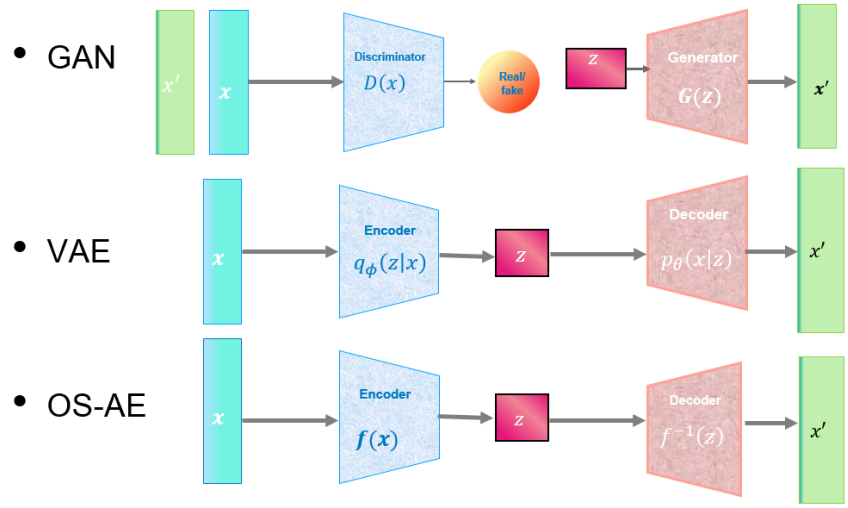

Figure 5.1: An abstracted comparison of input data mapping through the subspace among the generative models.

convergence and needs hours to train. In the previous chapter, we introduced a novel feature extraction method named OS-Autoencoder, which achieve prominent generalized performance in learning patterns from data and reconstruct data efficiently within a few epochs. In this chapter, we extend Online Sequential Autoencoder so that it can work as a generative model for manipulating images as well as tabular dataset and provide satisfactory evidence that augmented data can increase classification accuracy.

5.2 Introduction

Most of the real-world classification problems face some degree of class imbalance because of not having an equal proportion of data to make an unbiased decision. Class imbalance describes a dataset with a skewed ratio of majority to minority samples. If the proportion of data between different classes is small, then most of the machine learning or statistical algorithms perform well, but as this difference grows, these algorithms tend to assume most of the features are coming from the majority class. This imbalance can be reduced simply by augmenting classes either by oversampling the data points of the minority class or undersampling the instances of the majority class. Though theoretically, this does not allow classifiers to get biased toward

one class; these approaches are not efficient enough. Over-sampling introduces the likelihood of overfitting since it duplicates the minority class instances. Similarly, undersampling the majority can be biased by leaving out essential instances that hold deep features to differentiate two classes. Chawla et al. proposed a sophisticated algorithm named Synthetic Minority Over-sampling Technique (SMOTE) [5] and the extension of Borderline-SMOTE [32] to generate synthetic data by considering nearest neighbor example which achieved remarkable improvements over other sampling techniques. This algorithm can induce class overlapping, and as a result, it creates additional noise whenever it works with high dimensional data. Previous studies [4, 37] state that multi-layer neural network can manipulate high dimensional data efficiently, which motivate us to introduce neural network in the field of class imbalance of tabular data.

On the other hand, neural networks have become an integral part of image classification and revolutionized the performance by contributing autoencoders, convolutional neural networks and so on [24, 87]. However, they demand a large number of labeled training datasets to reduce overfitting problems [63], which is difficult to meet in practical applications. Therefore, some regularization technologies gained popularity to alleviate this problem such as dropout [88], batch normalization [47], transfer learning[78], semi-supervised learning [54] and data augmentation [75]. In [59], Hinton et al. mentioned the importance of data augmentation in order to train deep neural networks and reduce the generalization error. Unfortunately, data augmentation is an art, as it involves many choices, and an inappropriate choice can obscure the path to get the optimal solution. The most common augmentation technique for image data is category-free transformation methods to generate new samples from available ones. These methods include Flipping(flip image horizontally or vertically), Rotation (rotate an image in random orientation), Cropping(Crop part of the image in specific resolution), Color Jittering(change brightness and contrast), and Noise(add random disruption, basically gaussian noise) . However, these schemes are not informative enough, and augmented samples sometimes lead to no effect or create a detrimental effect on the accuracy, as well as the robustness of classifiers. Engstrom et al. [17] showed that basic transformations such as rotations or flipping could easily reduce accuracy by deep CNN models. For instance, the random transformations reduced the accuracy of MNIST by 26%, CIFAR10 [57] by 82%, and ImageNet (Top 1) [13] by 28%. On the other hand, with the adaptation of deep neural network, Generative Adversarial Networks(GANs)[29] and Variational Autoencoders (VAEs) [55] models

have drawn increasing attention for learning structure and patterns of input images and generating coherent and detailed images in unsupervised manner [62]. Basically, GANs are constructed with a generative model G and a discriminator model D. In Fig 5.1, the latent space of particular data distribution is mapped through G to generate data in the training process, and D is engaged in discriminating between real and synthesized instances produced by G.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log(D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))], \qquad (5.1)$$

The discriminator network fights against the generator network's backpropagation and provides the loss function to update gradient descent [105]. However, if the discriminator gets too successful, the generator gradient vanishes and suffer non-convergence. For that reason, selecting hyperparameter is highly sensitive, as the model parameter oscillate. As a result, it is time-consuming and performs at a high computation cost. For instance, GANs data augmentation technique experience on for something as simple as 28x28 grayscale MNIST digits takes around 10K epochs in Google Colab GPU environment to produce indistinguishable data from original ones. VAEs are another form of generative models nearly the same as traditional bottleneck autoencoders that follow variational bayesian learning to extract distribution from data. In Fig 5.1 we can see an encoder takes the latent vector from the input, and a decoder reconstructs the original variables from the latent space to generate unique images that have similar characteristics. A disadvantage of VAEs is that, because of the injected noise and imperfect reconstruction, and with the standard decoder (with factorized output distribution), the generated samples are much more blurred than those coming from GANs. Moreover, it suffers from high-variance because gradients bounces around in varying directions. Thus we motivated to ask: **Can we solve the gradient variance problems faced by the generative model?**

In Chapter 4, we propose an autoencoder named online sequential autoencoder, which has emerged to alleviate this problem faced by backpropagation. Experimental results prove that it extracts deep features efficiently by utilizing Moore-Penrose Inverse (see 5.1) and deliver better performance than the other iterative learning algorithms. Moreover, image reconstruction quality of this autoencoder is quite transparent because output weight can hold prominent feature precisely. It motivates us to add variation in output weight to analyze the changes in OS-AE re-construct data?

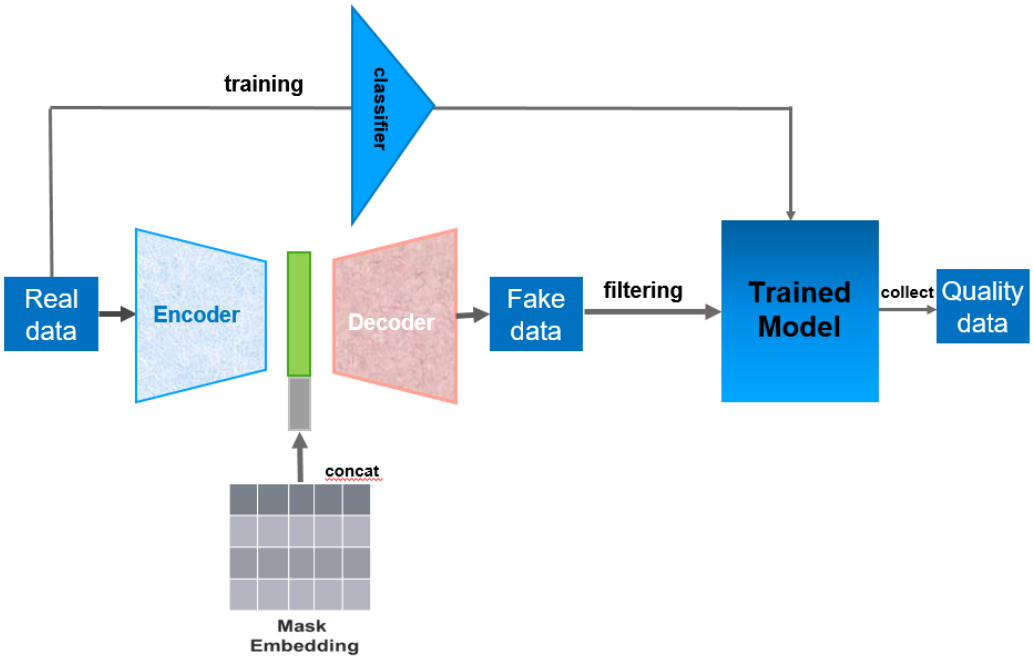

Figure 5.2: Structure of our proposed method in two layers. In the first layer, the d-dimensional inputs X map into an m-dimensional space. The number of hidden nodes m would be concatenated with a mask to generate fake data, which will be filtered by a trained model, with real data, to produce quality output.

In particular, this chapter has the following contributions:

1. A novel data augmentation technique is proposed using online sequential autoencoder, which can augment not only image dataset but also tabular dataset within fewer epochs .

2. Higher Quality Data. Experimental results show that the DCNN model with augmented data via our proposed algorithm acquire higher classification accuracy rather than same DCNN model itself or other data augmentation techniques. For instance, Resnet [33] combined with our method achieves 94.68% accuracy from CIFAR-10 [57] augmented datasets with only five epochs.

5.3 Method

The proposed method involves an online sequential autoencoder to process training data for extracting deep features and, based on these features, construct new data iteratively. It translates an image tensor of size height×width×color channels down into a vector size nx1, in the field of feature space augmentation. For that reason, the same architecture can be utilized for both image and tabular data augmentation. In Figure 5.2, we visualized how our network process the training data may arrive chunk-by-chunk to OS-AE and reduce the dimension of input data (d) into some of hidden nodes(m). A mask containing few random neurons will be concatenated with the extracted feature to learn prominent ones and will generate new data.

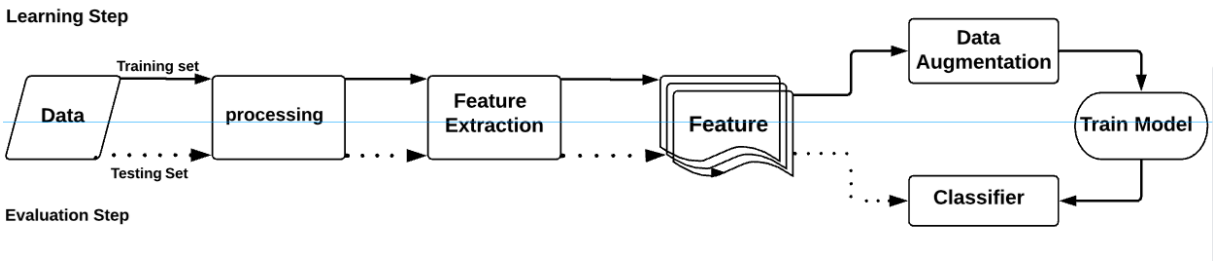

Figure 5.3: Work Flow of our proposed approach

5.3.1 Online Sequential Autoencoder

OS-AE is composed of an encoder that extracts subspace features from input variables and a decoder that reconstructs the original variables from the extracted feature by using pseudo-inverse. In chapter 4, we described the mathematical term to construct the autoencoder. According to Figure 5.4, OS-Autoencoder holds some visible patterns as weights of both encoding and decoding layers, which are updating continuously. Though VAE has the similar encoding and decoding process, learned patterns after the encoding layer is a random signal. As, the encoding layer weight is profuse enough, we can moderate that weight to utilize in data augmentation.

5.3.2 Reparameterization Weight

We noticed that the encoding weight of OS-AE bears significant information. If we enrich the extracted weight, we will generate new features. In Figure 5.3, we show the work-flow of data augmentation by utilizing feature extracted via OS-AE. To create

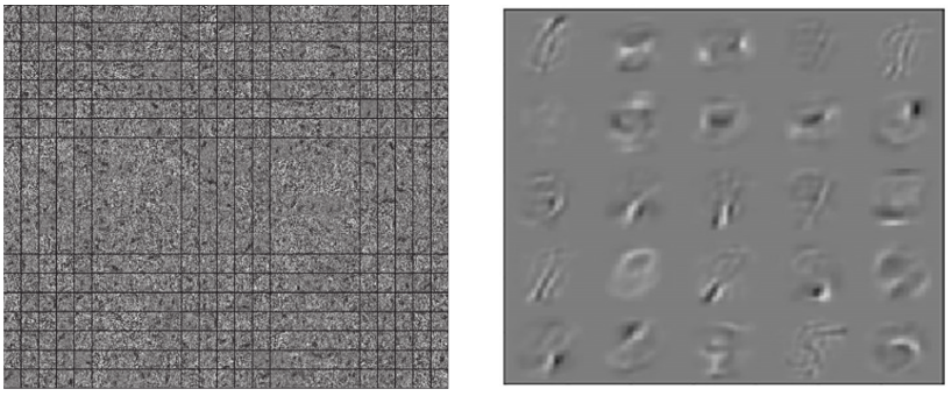

Figure 5.4: Visualized hidden layer learning information of (a)variational autoencoder [97] and (b) OS-Autoencoder on mnist dataset. VAE imposed some random signals, and OS-Autoencoder holds prominent features.

some variation in the extracted feature, we concatenate a mask to train the neurons to re-parameterize encoding layer weight. In chapter 4, Eq.4.6 is used to generate output weight. So, we are going to use that weight with some variant to train the neuron embedded in the mask. So,

$$\begin{aligned} \mathbf{M} &= \mathbf{M}\left[\mathbf{K}_{i+1}\boldsymbol{\beta}^{(\boldsymbol{i})} - \mathbf{H}_i^T\mathbf{H}_{i+1}\boldsymbol{\beta}^{(i)} + \mathbf{H}_{i+1}^T g^{-1}(\mathbf{X}_{i+1})\right], \\ &= \mathbf{M}\mathbf{K}_{i+1}\boldsymbol{\beta}^{(\boldsymbol{i})} - \mathbf{M}\mathbf{H}_i^T\mathbf{H}_{i+1}\boldsymbol{\beta}^{(i)} + \mathbf{M}\mathbf{H}_{i+1}^T g^{-1}(\mathbf{X}_{i+1}), \end{aligned} \tag{5.2}$$

We would utilize Eq. 5.2 and reshape this weight in the form of input data. As we embedded few neurons with the mask, we would take the average of that weight based on embedded neurons; we are going to train and subtract it from input data which will produce new data.

In chapter 4, Algorithm 3 has described the procedure of training OS-AE. Once the training finished, we will use $\boldsymbol{\beta}$ to train the mentioned neurons to generate new data through Algorithm 4.

Algorithm 4 OS-Autoencoder as a generative model

Result: Train new neuron to generate new data

while *neuron* < *number of neurons* **do**
    node ← *reshape*(*M*);
    node_weight ← *average*(*node*);
    neuron ← *neuron* + 1;
    end
    while *k* < *length(x_train)* **do**
        new_data ← *x_train*[*k*] + *node_weight*;
        node_weight ← *average*(*node*);
        k ← *k* + 1;
        end

The learning process used in this algorithm includes continuous update of the weights and gradually update neuron to generate new relevant data.

5.4 Experiments

In this section, We would describe the dataset and compare our proposed algorithm with other algorithms. We conduct our experiments in Keras with 32 GB of memory, Geforce 1080 8GPU and an I7-4470(3.4G) processor. To test the performance, we would use different algorithms for data augmentation in eight tabular and seven image dataset and compare mean classification accuracy as one of the evaluation metrics.

5.4.1 Comparison of Performance of Sampling Techniques and Our Method for Tabular Dataset

We compare OS-AE algorithm with random over-sampling(ROS), random under-sampling(RUS) and smote. To differentiate their efficiency, we use our OS-SN classifier with 2 sub-network nodes to obtain their classification accuracy. We have used imblearn library to apply to utilize mentioned algorithms.
In Table5.2, a performance comparison has been visualized among random over sampling, random under sampling, Smote and OS-AE. It indicates that our proposed algorithm can learn optimal features more accurately and reconstruct new data more efficiently rather than any sampling technique. Let consider the Hill dataset (small

Table 5.1: Performance Comparison on Classification Problems with Augmented Data

Dataset	**Dimension**	**No Aug**	**ROS**	**RUS**	**Smote**	**OS-AE**
Duke	7129	70.00	76.23	75.23	73.00	**80.20**
Hill	101	79.44	80.23	81.66	86.34	**90.25**
Protein	357	68.12	71.88	70.49	74.36	**75.87**
Mushroom	256	88.29	89.34	87.56	91.23	**96.15**
Leu	7129	78.56	79.75	84.47	77.82	**97.21**
Acoustic	51	65.29	67.66	66.35	**71.12**	70.02
DNA	180	91.25	90.34	90.55	92.02	**92.05**

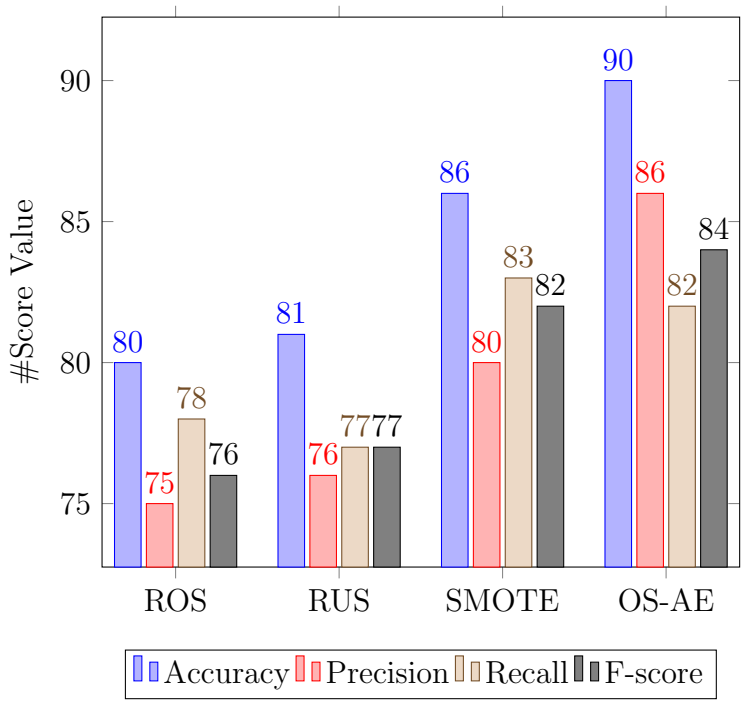

Figure 5.5: Generalization performance comparison on Hill-Valley Dataset based on sampling techniques and our proposed method where each bar represents Accuracy, Precision, Recall, and F-score respectively

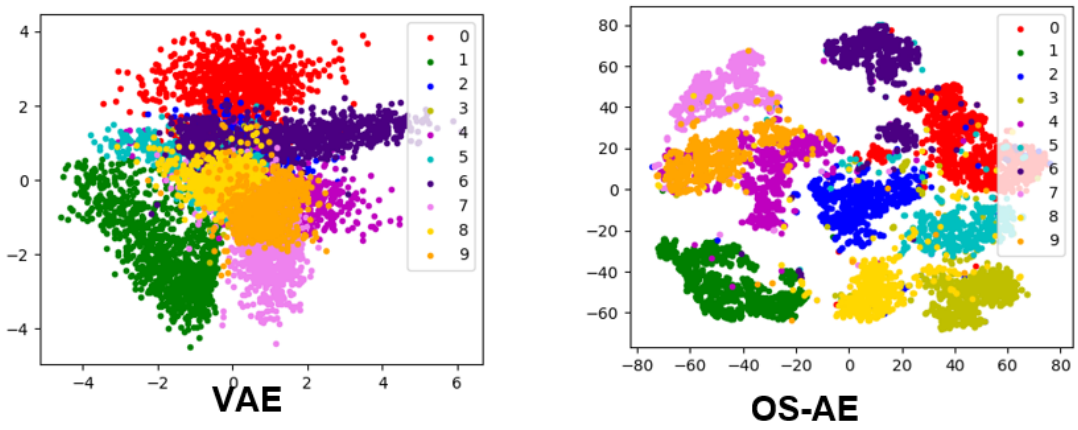

Figure 5.6: visualize the structure of encoding 784 dimensional MNIST dataset into latent space 2 by plotting each point with coloring by number it is [0,1,. . . ,9]

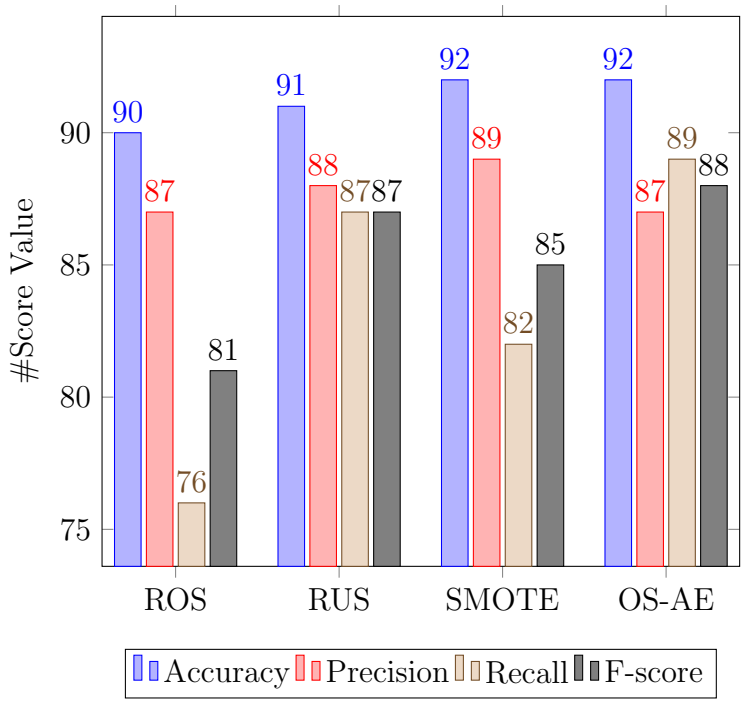

Figure 5.7: Generalization performance comparison on DNA Dataset based on sampling techniques and our proposed method where each bar represents Accuracy, Precision, Recall, and F-score respectively

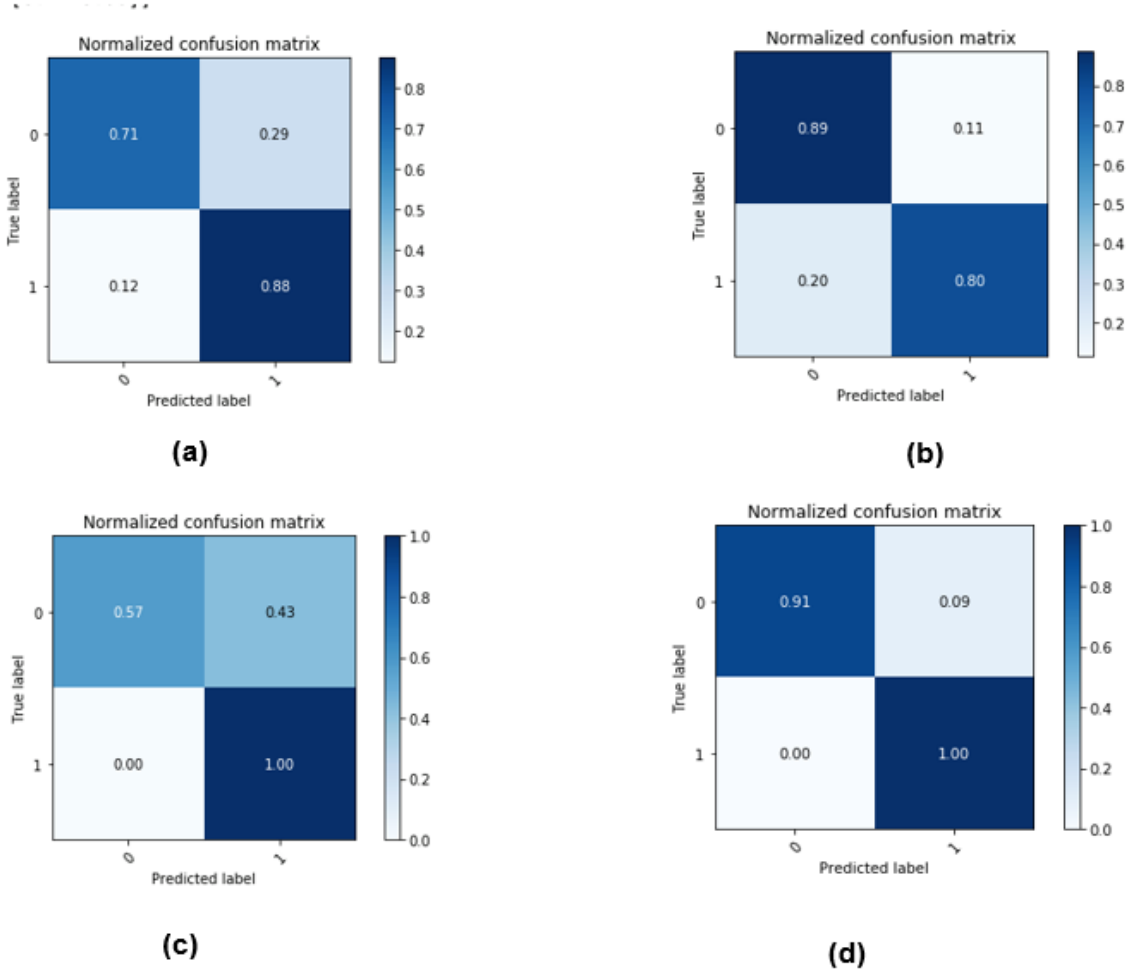

Figure 5.8: Comparison among Confusion Matrix of sampling techniques respectively (a)random over sampling, (b)random under sampling, (c) Smote and (d) OS-AE for Leu Dataset

datasets with less dimension) and Duke dataset (small samples with high dimensions). Firstly in Hill dataset, a significant increase in accuracy has been noticed. Our approach can able to generate 300 good quality unbiased data, which increased from 79% to 91% , whereas random over sampling and under sampling can able to increase tends to 1%. Though smote acquired kind of similar accuracy as our proposed algorithm, it did not perform well in high dimensional Duke dataset. It increased accuracy from 70% to 73%, whereas OS-AE can able to increase accuracy by 10% by generating 23 new data instance. For that reason, we can say that reconstruct new data based on optimal features learned by OS-AE perform much better rather than sampling majority or minority classes.

As it is hard to evaluate a model based on the only accuracy, we compare precision,recall and F-measure of all sampling techniques as well as our proposed algorithm on a low dimensional medium-sized dataset (Hill) and high dimensional small-sized dataset (Duke) in Figure 5.7. The quality of proposed method can be verified based on F-score, which is an average of precision and recall. We noticed synthetic data generated by OS-AE provides high precision and recall than other methods.

5.4.2 Image Data Augmentation

From Table 3.3, we can see that USPS, Mnist and Olivetti face are small dataset. For that reason, we use ELM with 1000 hidden nodes as a classifier to train these dataset. On the other hand, we use ResNet50[33] as a classifier for training Scene15 and Cifar10/100 dataset. Moreover, we process all dataset into grayscale from color and convert the image tensor from 3D vector to 2D vector. All the experiments are repeated ten times by randomizing selected training and testing images, and the average of per-class recognition rate has been recorded. We visualize the encoded feature structure of mnist dataset in Figure 5.6 to compare how data have been clustered based on their feature for both VAE and OS-AE algorithm. We noticed that each class has been clustered separately based on their features. However, OS-autoencoder performed well rather than VAE. Why this is happening? Because VAE is constructed based on probabilistic model and as a result some features of different class has been overlapped. On the other hand, OS-AE generates new data based on their prominent features, and as a result each class has been distinguished without any interference.

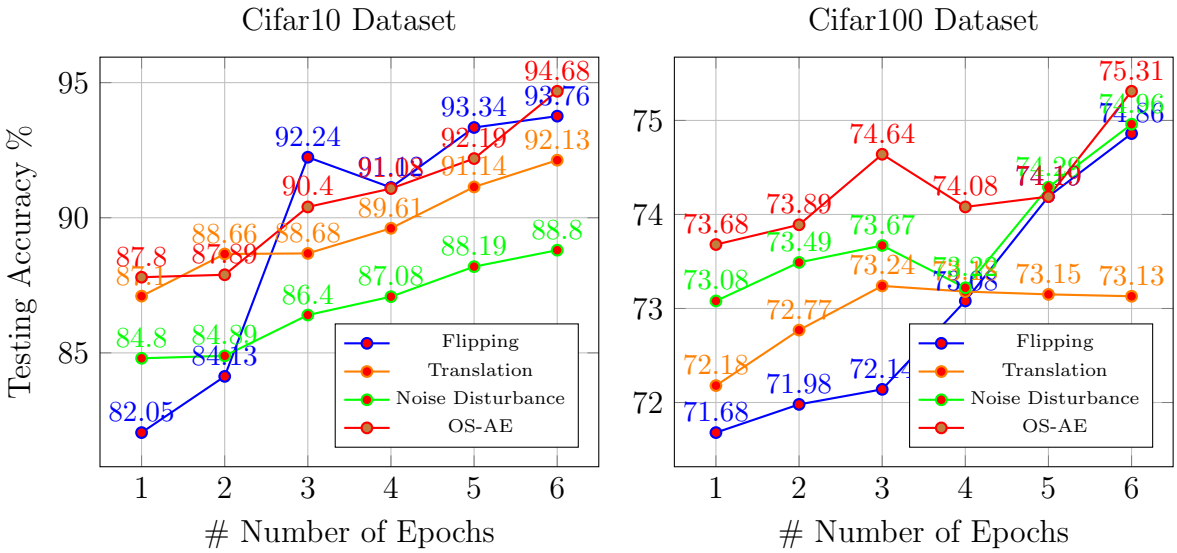

Figure 5.9: Generalization performance comparison on deep features (gathered by VGG16) of Cifar10 and Cifar100 dataset based on image augmentation technique and our proposed method where the x- and y-axis show the number of epochs and average testing accuracy, respectively. Result on (a) Cifar10, (b) Cifar100.

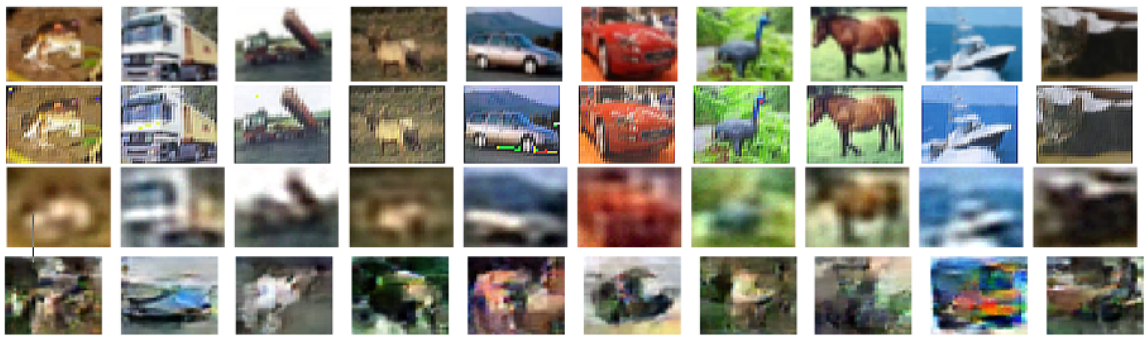

Figure 5.10: The qualitative comparisons of image reconstruction performance of VAE, DCGAN and our proposed algorithm. The first three rows from top to bottom: some images randomly sampled from Cifar10 training set; the corresponding OS-Autoencoder generated images; the corresponding VAE reconstructed images and DCGAN generated images.

Table 5.2: Performance Comparison on Classification Problems with Augmented Data

Dataset	**No Aug**	**Flipping**	**Translation**	**Rotation**	**Noise**	**Combination**	**OS-AE**
Olivetti face	92.05	93.78	93.23	93.64	83.67	93.21	**97.89**
Mnist	91.06	92.14	90.39	94.32	91.34	**94.78**	91.25
Usps	92.18	93.12	91.59	88.97	95.36	94.61	**96.17**
Cifar10	93.08	**94.71**	93.14	88.36	94.03	93.01	94.68
Cifar100	74.29	74.86	73.13	71.78	74.96	74.61	**75.31**
Scene15	87.08	88.23	84.54	86.75	88.02	87.88	**89.26**

Figure 5.11: The qualitative comparisons of data augmentation performance of VAE, DCGAN, and our proposed algorithm. The first three rows from top to bottom: some images randomly sampled from Cifar100 training set; the corresponding OS-Autoencoder generated images; the corresponding VAE reconstructed images and DCGAN [77] generated images.

Comparison of performance of Scene15, Cifar10/100

The result showed in Table 5.2 indicate that the DCNN model with augmented dataset with different methods significantly increment classification accuracy whereas OS-AE generated images to acquire the highest value. we first converted the images from three channels RGB color space to one channel grayscale. Therefore, input dimension is 32x32x3=3072. To extract the deep feature of vgg16 pre-trained weight, we add a layer of 3072 dimension with vgg16 as an initial parameter and send that feature through our OS-AE, which can create more enriched and new dataset. For Cifar 10/100 dataset, top accuracy gained by Resnet-50 without augmentation is 93.08% and 74.29%, which has been increased by OS-AE that is 94.68% and 75.31%. In Figure 5.9, we visualize how our proposed method acquire the highest accuracy compare to other traditional methods within six epochs in cifar10 and Cifar100 dataset.

5.4.3 Image Reconstruction

To compare the image generation quality of our algorithm, we compare with VAE, DCGAN and our proposed algorithm. We trained the neural networks on CIFAR10 and CIFAR100 [57] dataset.

VAE encodes the input data as a distribution over the latent space. It ensures that the latent space has excellent properties that enable the generative process. We

have used Cifar10 and Cifar100 as input and have utilize 500 epochs to generate images that needed at-least 30 minutes. In figure 5.10 and 5.11, we can see after 500 epochs the generated images are blurry, and it is hard to identify the object features. Convolutional neural networks help to find spatial correlation in input data, and that is the reason why DCGAN[77] would be a better alternative of GAN for image/video data. In figure 5.10 and 5.11, we can see some realistic and high-resolution visual content where patterns are fake. For that reason, images do not hold any notable pattern through which we can use those images for classification. The training is performed over 400 epochs and needs huge computation power. Comparing to VAE and DCGAN, OA-AE has used only 10 epochs on Cifar10/100 dataset separately, and the quality of the image is much precise and can reconstruct prominent features.

5.5 Conclusion

In this Chapter, we proposed a novel estimator of autoencoder that can efficiently generate both image and tabular data and optimized using online sequential learning. Our proposed method can learn convenient representations of features and generate new data which play a vital role to solve overfitting problem. The theoretical advantages are reflected in experimental results by proving that our proposed method can generate quality data within few seconds compared to other generative models.

Chapter 6

Conclusion & Future Work

6.1 Overview

Iterative based learning has become a paradigm for training a hierarchical neural network to increase the efficiency of the model but required long training time for slow convergence of the network. To reduce the time complexity faced by the hierarchical network, we introduced online sequential learning with the benefit of a non-iterative strategy. Moreover, we extend the network to build a generative model that can create some vibrant new data.

6.2 Future Work

For high dimensional imagenet datasets Such as Cifar10, Cifar100, and Caltech101, we have used a DCNN model tuned with Imagenet pre-trained weight as an initial parameter and with end-to-end training to extract complex features. We can improve these models by substituting gradient descent with online sequential learning. In [98] Yang et al. introduced a non-iterative learning strategy to retrain neurons of fully connected (FC) layers of DCNN, which provide better performance than the same

network with its original BP based training. It motivates us to introduce online sequential learning in the DCNN model so that our hierarchical network would be more efficient and powerful.
Furthermore, our proposed method with OS-AE can learn convenient representations of features most of the time, as well as faster training speed compared to other generative models. There are still some forms of modification is acquired to get realistic images like GAN. Variational auto-encoder outputs can be improved by forcing generative adversarial training mechanism [41] to generate less blurry images. In the future, we will focus on deep feature consistent principle [40] to learn how to embed GANs into our network. Additionally, we can send our OS-AE deep feature instead of noise vector inputs to GANs utilizing Bidirectional GANs [15]. As our model works well in both tabular and image data, it inspires us to extend this framework to other domains such as video (frame generation) and audio (speech synthesis).

6.3 Conclusion

In this thesis, we present an online sequential hierarchical network scheme with a non-iterative strategy for image recognition. We also extend this network for performing as a generative model. The network is approached from three main ways:

- Prominent features extracted from input data and map these features for classification following batch-by-batch learning.
- Low-dimensional subspace features fused by different operators;
- Without substituting iterative learning, we solve the gradient descent problem faced by BP.
- Instead of using randomized input weights, we can approach a classifier where weights would be configured by calculation and reach to the steepest descent iteratively without configuring the learning rate. Moreover, it does not need extra computation overload and provides excellent accuracy in less time.
- We optimized the usage of hidden nodes by substituting with sub-network node, which minimizes the training time significantly.
- Features extracted through OS-AE as a low-dimensional subspace features and fused it with a variance to generate new data.

Furthermore, this thesis indicates that our network functions as a feature extractor, a classifier and a generative model. The experimental results show that our network performs better than other relevant state-of-the-art methods.

Bibliography

[1] Z. Bai, G. Huang, D. Wang, H. Wang, and M. B. Westover. Sparse extreme learning machine for classification. *IEEE Transactions on Cybernetics*, 44(10):1858–1870, Oct 2014.

[2] P. L. Bartlett. For valid generalization, the size of the weights is more important than the size of the network. In *Proceedings of the 9th International Conference on Neural Information Processing Systems*, NIPS'96, pages 134–140, Cambridge, MA, USA, 1996. MIT Press.

[3] P. L. Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory*, 44(2):525–536, March 1998.

[4] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, 2007.

[5] K. W. Bowyer, N. V. Chawla, L. O. Hall, and W. P. Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *CoRR*, abs/1106.1813, 2011.

[6] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[7] D. Cai, X. He, and J. Han. Isometric projection. In *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 1*, AAAI'07, page 528–533. AAAI Press, 2007.

[8] J. Cao, Y. Zhao, X. Lai, T. Chen, N. Liu, B. Mirza, and Z. Lin. Landmark recognition via sparse representation. 07 2015.

[9] Cheng Xiang, S. Q. Ding, and Tong Heng Lee. Geometrical interpretation and architecture selection of mlp. *IEEE Transactions on Neural Networks*, 16(1):84–96, 2005.

[10] C. M. Conway. *Sequential Learning*, pages 3047–3050. Springer US, Boston, MA, 2012.

[11] S. Dargan, M. Kumar, M. R. Ayyagari, and G. Kumar. A survey of deep learning and its applications: A new paradigm to machine learning. *Archives of Computational Methods in Engineering*, pages 1–22, 2019.

[12] C. Deng, G.-B. Huang, J. Xu, and J. Tang. Extreme learning machines: new trends and applications. *Science China Information Sciences*, 58:1–16, 02 2015.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[14] S. Ding, N. Zhang, X. Xu, L. Guo, and J. Zhang. Deep extreme learning machine and its application in eeg classification. *Mathematical Problems in Engineering*, 2015, 2015.

[15] J. Donahue, P. Krähenbühl, and T. Darrell. Adversarial feature learning. *CoRR*, abs/1605.09782, 2016.

[16] M. Dorfer, R. Kelz, and G. Widmer. Deep linear discriminant analysis, 2015.

[17] L. Engstrom, D. Tsipras, L. Schmidt, and A. Madry. A rotation and a translation suffice: Fooling cnns with simple transformations. *ArXiv*, abs/1712.02779, 2017.

[18] S. E. Fahlman. An empirical study of learning speed in backpropagation networks. Technical Report CMU-CS-88-162, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1988.

[19] X. Fang, Z. Tie, Y. Guan, and S. Rao. Quasi-cluster centers clustering algorithm based on potential entropy and t-distributed stochastic neighbor embedding. *Soft Computing*, May 2018.

[20] L. Fei-Fei and P. Perona. A bayesian hierarchical model for learning natural scene categories. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 524–531. IEEE, 2005.

[21] R. French. Semi-distributed representations and catastrophic forgetting in connectionist networks. *CONNECTION SCIENCE*, 4:365–377, 01 1992.

[22] R. French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3:128–135, 05 1999.

[23] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

[24] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[25] K. Fukushima. Training multi-layered neural network neocognitron. *Neural Networks*, 40:18–31, 2013.

[26] C. F. Gauss, C. H. Davis, and M. of America Project. *Theory of the motion of the heavenly bodies moving about the sun in conic sections a translation of Gauss's "Theoria motus." With an appendix.* Boston,Little, Brown and company,. https://www.biodiversitylibrary.org/bibliography/19023.

[27] T. Ghosh. Quicknet: Maximizing efficiency and efficacy in deep architectures. 01 2017.

[28] J. Goldberger, G. E. Hinton, S. T. Roweis, and R. R. Salakhutdinov. Neighbourhood components analysis. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 513–520. MIT Press, 2005.

[29] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[30] W. Guo, T. Xu, K. Tang, et al. Online sequential extreme learning machine with generalized regularization and forgetting mechanism. *Control Decis*, 32:247–254, 2017.

[31] W. Guo, T. Xu, K. Tang, J. Yu, and S. Chen. Online sequential extreme learning machine with generalized regularization and adaptive forgetting factor for time-varying system prediction. 2018.

[32] H. Han, W. Wang, and B. Mao. Borderline-smote: A new over-sampling method in imbalanced data sets learning. In *ICIC*, 2005.

[33] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[34] X. He, M. Ji, C. Zhang, and H. Bao. A variance minimization criterion to feature selection using laplacian regularization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(10):2013–2025, 2011.

[35] G. Hinton, L. Deng, D. Yu, G. Dahl, A. rahman Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *Signal Processing Magazine*, 2012.

[36] G. Hinton and S. Roweis. Stochastic neighbor embedding. 15, 06 2003.

[37] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504 – 507, 2006.

[38] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.

[39] G. E. Hinton and S. T. Roweis. Stochastic neighbor embedding. In *Advances in neural information processing systems*, pages 857–864, 2003.

[40] X. Hou, L. Shen, K. Sun, and G. Qiu. Deep feature consistent variational autoencoder. *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1133–1141, 2016.

[41] X. Hou, K. Sun, L. Shen, and G. Qiu. Improving variational autoencoder with deep feature consistent and generative adversarial training. *CoRR*, abs/1906.01984, 2019.

[42] G. Huang, S. Song, J. N. D. Gupta, and C. Wu. Semi-supervised and unsupervised extreme learning machines. *IEEE Transactions on Cybernetics*, 44:2405–2417, 2014.

[43] G.-B. Huang, L. Chen, C. K. Siew, et al. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Trans. Neural Networks*, 17(4):879–892, 2006.

[44] G.-B. Huang, P. Saratchandran, and N. Sundararajan. An efficient sequential learning algorithm for growing and pruning rbf (gap-rbf) networks. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, 34:2284–92, 01 2005.

[45] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang. Extreme learning machine for regression and multiclass classification. *Trans. Sys. Man Cyber. Part B*, 42(2):513–529, Apr. 2012.

[46] H. T. Huynh and Y. Won. Regularized online sequential learning algorithm for single-hidden layer feedforward neural networks. *Pattern Recognition Letters*, 32(14):1930–1935, 2011.

[47] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[48] R. A. Jacobs. Increased rates of convergence through learning rate adaptation. Technical report, USA, 1987.

[49] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 315–323. Curran Associates, Inc., 2013.

[50] I. Jolliffe. *Principal Component Analysis*. Springer Verlag, 1986.

[51] L. Kasun, H. Zhou, G.-B. Huang, and C.-M. Vong. Representational learning with elms for big data. *IEEE Intelligent Systems*, 28:31–34, 11 2013.

[52] L. L. C. Kasun, H. Zhou, G.-B. Huang, and C. M. Vong. Representational learning with elms for big data.(2013). 2013.

[53] J. Kim. Sequential training algorithm for neural networks. *CoRR*, abs/1905.07490, 2019.

[54] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3581–3589, 2014.

[55] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2013. cite arxiv:1312.6114.

[56] A. K. Kolmogorov. On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSSR*, 114:369–373, 1957.

[57] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[58] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[59] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[60] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. 2006.

[61] Y. LeCun, P. Y. Simard, and B. Pearlmutter. Automatic learning rate maximization by on-line estimation of the hessian's eigenvectors. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 156–163. Morgan-Kaufmann, 1993.

[62] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. P. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802, 2016.

[63] J. Lemley, S. Bazrafkan, and P. Corcoran. Smart augmentation learning an optimal data augmentation strategy. *Ieee Access*, 5:5858–5869, 2017.

[64] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan. A fast and accurate online sequential learning algorithm for feedforward networks. *IEEE Transactions on neural networks*, 17(6):1411–1423, 2006.

[65] B. Liu, S.-X. Xia, F.-R. Meng, and Y. Zhou. Extreme spectral regression for efficient regularized subspace learning. *Neurocomputing*, 149:171–179, 2015.

[66] H. Liu, H. Liu, F. Sun, and B. Fang. Kernel regularized nonlinear dictionary learning for sparse coding. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(4):766–775, April 2019.

[67] H. Liu, F. Sun, B. Fang, and D. Guo. Cross-modal zero-shot-learning for tactile object recognition. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pages 1–9, 2018.

[68] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004.

[69] Y. Lu, N. Sundararajan, and P. Saratchandran. Performance evaluation of a sequential minimal radial basis function (rbf) neural network learning algorithm. *IEEE transactions on neural networks*, 9 2:308–18, 1998.

[70] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[71] Ming Liang and Xiaolin Hu. Recurrent convolutional neural network for object recognition. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3367–3375, June 2015.

[72] M. Naveen, S. Jayaraman, V. Ramanath, and S. Chaudhuri. Modified levenberg marquardt algorithm for inverse problems. In K. Deb, A. Bhattacharya,

N. Chakraborti, P. Chakroborty, S. Das, J. Dutta, S. K. Gupta, A. Jain, V. Aggarwal, J. Branke, S. J. Louis, and K. C. Tan, editors, *Simulated Evolution and Learning*, pages 623–632, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[73] R. Neuneier and H. G. Zimmermann. *How to Train Neural Networks*, pages 373–423. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[74] Y.-H. Pao and Y. Takefuji. Functional-link net computing: theory, system architecture, and functionalities. *Computer*, 25(5):76–79, 1992.

[75] L. Perez and J. Wang. The effectiveness of data augmentation in image classification using deep learning. 12 2017.

[76] J. Platt. A resource-allocating network for function interpolation. *Neural Comput.*, 3(2):213–225, June 1991.

[77] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015. cite arxiv:1511.06434Comment: Under review as a conference paper at ICLR 2016.

[78] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng. Self-taught learning: transfer learning from unlabeled data. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 759–766, New York, NY, USA, 2007. ACM.

[79] C. R. Rao. *Generalized inverse of matrices and its applications.* Number 04; QA263, R3. 1971.

[80] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In E. P. Xing and T. Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1278–1286, Bejing, China, 22–24 Jun 2014. PMLR.

[81] F. Ritter, J. Nerb, E. Lehtinen, and T. O'Shea. *In Order to Learn: How the Sequence of Topics Influences Learning.* Oxford Series on Cognitive Models and Architectures. Oxford University Press, 2007.

[82] A. Robins. Sequential learning in neural networks: A review and a discussion of pseudorehearsal based methods. *Intell. Data Anal.*, 8(3):301–322, Aug. 2004.

[83] E. M. Rosen. A review of quasi-newton methods in nonlinear equation solving and unconstrained optimization. In *Proceedings of the 1966 21st National Conference*, ACM '66, page 37–41, New York, NY, USA, 1966. Association for Computing Machinery.

[84] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press, Cambridge, MA, USA, 1986.

[85] D. M. Salih, S. B. M. Noor, M. H. Merhaban, and R. M. Kamil. Wavelet network: Online sequential extreme learning machine for nonlinear dynamic systems identification. *Adv. in Artif. Intell.*, 2015, Jan. 2015.

[86] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[87] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[88] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[89] Z.-L. Sun, T.-M. Choi, K.-F. Au, and Y. Yu. Sales forecasting using extreme learning machine with applications in fashion retailing. *Decision Support Systems*, 46:411–419, 12 2008.

[90] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, page 1067–1077, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee.

[91] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, Berlin, Heidelberg, 1995.

[92] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, Dec. 2010.

[93] P. Wei, Y. Ke, and C. K. Goh. Feature analysis of marginalized stacked denoising autoenconder for unsupervised domain adaptation. *IEEE Transactions on Neural Networks and Learning Systems*, 30(5):1321–1334, May 2019.

[94] D. Xiao, B. Li, and Y. Mao. A multiple hidden layers extreme learning machine method and its application. *Mathematical Problems in Engineering*, 2017, 2017.

[95] D. Xiao, B. Li, and S. Zhang. An online sequential multiple hidden layers extreme learning machine method with forgetting mechanism. *Chemometrics and Intelligent Laboratory Systems*, 176:126–133, 2018.

[96] H. Y. Xiong, B. Alipanahi, L. J. Lee, H. Bretschneider, D. Merico, R. K. C. Yuen, Y. Hua, S. Gueroussov, H. S. Najafabadi, T. R. Hughes, Q. Morris, Y. Barash, A. R. Krainer, N. Jojic, S. W. Scherer, B. J. Blencowe, and B. J. Frey. Rna splicing. the human splicing code reveals new insights into the genetic determinants of disease. *Science (New York, N.Y.)*, 347(6218):1254806, January 2015.

[97] Q. Xu, Z. Wu, Y. Yang, and L. Zhang. The difference learning of hidden layer between autoencoder and variational autoencoder. In *2017 29th Chinese Control And Decision Conference (CCDC)*, pages 4801–4804, May 2017.

[98] Y. Yang, J. Wu, X. Feng, and A. Thangarajah. Recomputation of dense layers for the performance improvement of dcnn. *IEEE transactions on pattern analysis and machine intelligence*, PP, 05 2019.

[99] Y. Yang, Q. J. Wu, and Y. Wang. Autoencoder with invertible functions for dimension reduction and image reconstruction. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(7):1065–1079, 2018.

[100] Y. Yang and Q. M. J. Wu. Extreme learning machine with subnetwork hidden nodes for regression and classification. *IEEE Transactions on Cybernetics*, 46(12):2885–2898, Dec 2016.

[101] Y. Yang and Q. M. J. Wu. Multilayer extreme learning machine with subnetwork nodes for representation learning. *IEEE Transactions on Cybernetics*, 46(11):2570–2583, Nov 2016.

[102] Y. Yang and Q. M. J. Wu. Features combined from hundreds of midlayers: Hierarchical networks with subnetwork nodes. *IEEE Transactions on Neural Networks and Learning Systems*, PP:1–13, 01 2019.

[103] L. Yingwei, N. Sundararajan, and P. Saratchandran. A sequential learning scheme for function approximation using minimal radial basis function neural networks. *Neural Comput.*, 9(2):461–478, Feb. 1997.

[104] R. Zhang, Y. Lan, G. Huang, and Z. Xu. Universal approximation of extreme learning machine with adaptive growth of hidden nodes. *IEEE Transactions on Neural Networks and Learning Systems*, 23(2):365–371, 2012.

[105] X. Zhang, Z. Wang, D. Liu, and Q. Ling. Dada: Deep adversarial data augmentation for extremely low data regime classification. abs/1809.00981, 2018.