Extreme learning machines for feature learning

Liyanaarachchi Lekamalage, Chamara Kasun

2017

Liyanaarachchi Lekamalage, C. K. (2017). Extreme learning machines for feature learning.
Doctoral thesis, Nanyang Technological University, Singapore.

<http://hdl.handle.net/10356/69620>

<https://doi.org/10.32657/10356/69620>

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

EXTREME LEARNING MACHINES FOR FEATURE LEARNING

LIYANAARACHCHI LEKAMALAGE CHAMARA KASUN

School of Electrical & Electronic Engineering

2016

EXTREME LEARNING MACHINES FOR FEATURE LEARNING

LIYANAARACHCHI LEKAMALAGE CHAMARA KASUN

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

2016

Statement of Originality

I hereby certify that the content of this thesis is the result of work done by me and has not been submitted for a higher degree to any other University or Institution.

.....

Date

.....

LIYANAARACHCHI LEKAMALAGE

CHAMARA KASUN

Acknowledgments

I would like to express my heartfelt appreciation to my supervisor Associate Professor Huang Guangbin; this thesis would not have been possible without your guidance. I would like to thank you for your illuminating discussions and also for providing opportunities such as proposal writing, conference arrangement to grow as a researcher.

I would like to thank my thesis advisory committee members Associate Professor Qing Song and Assistant Professor Wang Gang for providing valuable insights to improve my research. I would also thank Associate Professor Xudong Jang for providing datasets for my research work.

I would also like to thank Dr. Yang Yan and Dr. Bai Zuo for their time and provide valuable input to improve my research. I would like to thank my friends and my lab mates from Internet of Things Laboratory, BMW-HMI lab, and EEE-Delta joint lab on Internet of things for their valuable discussions. I would also like to thank the laboratory executive Mr. Yuen Sien Huan for helping in acquiring computer resources.

Finally, I would like to thank my parents for their support and understanding during my PhD.

Table of Contents

Acknowledgments	i
Summary	vi
List of Figures	xv
List of Tables	xvii
1 Introduction	1
1.1 Research Background	1
1.2 Motivation and Objectives	4
1.3 Major Contributions	6
1.4 Organization	8
2 Literature Review	9
2.1 Extreme Learning Machines	9
2.1.1 Extreme Learning Machine based Local Receptive Field	12
2.1.2 Removing Neurons Which Negatively Affect the Generalization Performance of Extreme Learning Machine	15
2.1.3 Deep Extreme Learning Machine	16

2.1.4	Extreme Learning Machine for Data Visualization	18
2.1.5	Extreme Learning Machine for Clustering	19
2.2	Feature Learning	20
2.2.1	Principal Component Analysis	21
2.2.2	Random Projection	22
2.2.3	Non-negative Matrix Factorization	23
2.2.4	Tied Weight Auto-Encoder	24
2.2.5	Restricted Boltzmann Machine (RBM)	26
2.3	Multi-Layer Neural Networks	27
2.4	Facial Point Detection	29
2.4.1	Supervised Descent Method	30
2.5	Summary	33
3	Extreme Learning Machine based Dimension Reduction	35
3.1	Extreme Learning Machine based Dimension Reduction	36
3.1.1	Linear Extreme Learning Machine Auto-Encoder	37
3.1.2	Non-linear Extreme Learning Machine Auto-Encoder	42
3.1.3	Linear Sparse Extreme Learning Machine Auto-Encoder	43
3.1.4	Non-linear Sparse Extreme Learning Machine Auto-Encoder	46
3.1.5	ELM-AE (Linear and Non-Linear) and SELM-AE (Linear and Non-Linear) Function as a Denoiser	47
3.2	Experiments	48
3.2.1	Benchmark Datasets	50
3.2.2	Visual Investigation of Features Learned by ELM-AE and SELM- AE	51

3.2.3	Performance Comparison of Different Algorithms in USPS, CIFAR-10, and NORB Datasets	54
3.3	Summary	62
4	Extreme Learning Machine based Multi-Layer Neural Network	73
4.1	Non-linear Extreme Learning Machine Auto-Encoder	73
4.2	Multi-Layer Extreme Learning Machine	75
4.3	C Parameter of Non-Linear ELM-AE Selects Important Features	76
4.4	Visual Investigation of Features Learned by Non-Linear ELM-AE . . .	77
4.5	Visual Investigation of Features Learned by a Two Layer ML-ELM . . .	77
4.6	Performance Evaluation	78
4.7	Summary	79
5	Extreme Learning Machine based Local Receptive Field Neural Network	85
5.1	Local Receptive Field Extreme Learning Machine Auto-Encoder . . .	85
5.1.1	Learning Convolution Layer Parameters of Local Receptive Field Extreme Learning Machine Auto-Encoder	86
5.1.2	Pooling Layer of Local Receptive Field Extreme Learning Machine Auto-Encoder	89
5.1.3	Learning Fully Connected Layer Weights of Local Receptive Field Extreme Learning Machine Auto-Encoder	90
5.1.4	Learning Output Weights of Local Receptive Field Extreme Learning Machine Auto-Encoders	90
5.2	Experiments	91
5.3	Summary	92

6	Extreme Learning Machine based Facial Point Detection	96
6.1	Extreme Learning Machine Supervised Descent Method	97
6.1.1	Creating response maps	98
6.2	Results of Extreme Learning Machine base Facial Point Detection Al- gorithm	100
6.3	Summary	111
7	Conclusions and Future Work	112
7.1	Conclusions	112
7.2	Future work	114
	Author's Publications	115
	Bibliography	116

Summary

Neural Networks (NN) map input data to desired output data in image processing, time series prediction and data analytics. The commonly used variant of NN is Single Layer Feed forward Neural network (SLFN) due to its simple network architecture and universal approximation capability. Traditionally the hidden layer and output layer weights of NN is learned by Back-Propagation (BP) algorithm which has problems such as convergence to a local minimum and slow learning rate.

In contrast to traditional SLFN, Extreme Learning Machine (ELM) is a “generalized” SLFN with a wide variety of hidden neurons types such as sigmoid, wavelets, radial bias functions, and Fourier series. Furthermore, the hidden layer parameters of ELM are randomly selected independent from input data, and only the output parameters are tuned.

Input data might contain noise and unwanted information, which might affect machine learning algorithms generalization capability. Hence the objective of feature learning is to remove these noise and unwanted information from input data. ELM has been successfully used in regression and classification problems but not in feature learning. Hence this thesis extends ELM for feature learning.

This thesis introduces an ELM based feature learning framework with linear hidden layer activation function referred to as linear Extreme Learning Machine Auto-Encoder (ELM-AE) and linear Sparse Extreme Learning Machine Auto-Encoder (SELM-AE); ELM-AE and SELM-AE with sigmoid hidden layer activation function referred to as non-linear ELM-AE and non-linear SELM-AE. For dimension reduction, linear ELM-

AE and linear SELM-AE, in theory, learn the between-cluster scatter. Hence performing dimension reduction with linear ELM-AE and linear SELM-AE will cluster data points based on Euclidean distance (i.e. if the Euclidean distance between two data points is small, then these data points are assigned to one cluster) and reduce the Euclidean distance between data points belonging to the same cluster. For dimension reduction the efficacy of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE is compared with PCA, Non-negative Matrix Factorization (NMF), and Tied weight Auto-Encoder (TAE) in terms of discriminative capability, sparsity, Normalized Mean Square Error (NMSE), and training time.

Experimentally it has been shown that multi-layer neural networks have better generalization capability than SLFN in image classification tasks and learn hierarchal features. Hierarchal features consist of multiple levels and the low level features represent edges of an image such as lines and curves while high level features represent parts of the image such as nose, eyes of a face. This thesis extends ELM to fully connected multi-layer neural networks by stacking non-linear ELM-AE output weights and referred as Multi-Layer Extreme Learning Machine (ML-ELM). The experimental results show that the training time of Multi-Layer ELM (ML-ELM) is significantly lower than other multi-layer neural networks such as Deep Boltzmann Machine (DBM) and Deep Belief Network (DBN), while the generalization capability is similar.

In contrast to fully connected multi-layer neural networks, Local receptive field networks performs better for image data. This thesis investigates an local receptive field network referred to as Local Receptive Field Extreme Learning Machine Auto-Encoder (LRF-ELMAE) which learns hidden layer parameters by non-linear ELM-AE and not by BP. Experimental results show that LRF-ELMAE training time is significantly lower than other local receptive field networks while achieving state-of-the-art results.

The objective of facial point detection is to detect eyes, nose, mouth, and face contour in the face image. Facial point information is used to determine head pose, facial expressions and face animation. The proposed ELM Facial Point Detection (ELM-FPD) consist of the three algorithms: 1) automatic face recognition algorithm such as Vi-

ola Jones algorithm locates the face in the image; 2) LRF-ELMAE estimates the facial points on the located faces using mean face shape; 3) Extreme Learning Machine Supervised Descent Method (ELM-SDM) improves the estimated facial points using low dimensional response maps. The experimental results show that ELM-FPD outperforms other multi-layer neural networks for facial point detection.

List of Figures

2.1	ELM additive type network randomly generates the hidden node parameters $(\mathbf{a}_i, b_i)$. ELM calculates the output of i -th hidden node for input $\mathbf{x}$ as: $h_i(\mathbf{x}) = g(\mathbf{a}_i, \mathbf{x}, b_i)$. ELM hidden layer activation function g is a non-linear piecewise continuous function.	10
2.2	ELM random connection types. a) In a fully connected architecture each neuron connects to the neurons in the previous layer. b) In local receptive field network architecture each neuron connects to some neurons in the previous layer.	12
2.3	Combinatorial node of ELM-LRF. Neurons of ELM-LRF can be combined using combinatorial node and enforces translation invariance. Commonly used combinatorial nodes are max pooling, average pooling and sum pooling.	13
2.4	ELM-LRF network architecture. ELM-LRF generates the first layer connections from a step probability function, ELM-LRF generates the first layer weights from a uniform random distribution, and ELM-LRF uses square root pooling function for the second layer. Linear regression network learns the output layer weights.	14
2.5	Deep ELM network structure with one hidden layer. The first hidden layer weights $\mathbf{A}^1$ is generated randomly. Equation (2.9) calculates the second hidden layer weights β^2 of Deep ELM.	17

- 2.6 Deep ELM network structure with three hidden layers. The first hidden layer weights $\mathbf{A}^1$ is generated randomly. Equation (2.9) calculates the second hidden layer weights β^2 of Deep ELM. Third hidden layer weights $\mathbf{A}^3$ is generated randomly. Equation (2.11) calculates the output layer weights of three hidden layer Deep ELM. 18
- 2.7 Deep ELM hidden layers can be combined to create a compact network structure. Deep ELM with three hidden layers can be combined to finally create a two hidden layer network. This is done by multiplying the linearly connected hidden layer weights ($\beta^2 \mathbf{A}^3$). Hence the final Deep ELM network has only non-linear hidden layers. 18
- 2.8 TAE network architecture. TAE uses the same weights β_{TAE} for the encoder stage and decoder stage. TAE learns features in three representations: 1) Compressed architecture; 2) Equal dimension representation; 3) Sparse architecture. 25
- 2.9 RBM network architecture. In a RBM the input nodes and hidden nodes connected, but there are no connection between input nodes or between hidden nodes. 26
- 2.10 DBN network architecture. (a) RBM learns the first hidden layer weights of DBN $\mathbf{W}^1$ by modeling input data $\mathbf{x}$. (b) RBM learns $(i+1)$ -th hidden layer weights of DBN $\mathbf{W}^{i+1}$ by modeling the output of i -th hidden layer $\mathbf{h}^i$. (c) Softmax classifier learns the output layer weights of DBN and BP fine tunes the whole network. 29
- 2.11 SAE network architecture. (a) TAE learns the SAE first hidden layer weights β^1 by modeling input data $\mathbf{x}$. (b) TAE learns the SAE $(i+1)$ -th hidden layer weights β^{i+1} by modeling i -th hidden layer $\mathbf{h}^i$. (c) Softmax classifier learns the last layer weights of SAE and BP fine tunes the whole network. 30

- 3.1 Differences between the proposed ELM based AE framework (linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE) and the original ELM are: 1) the output of linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE is equal to the input data $\mathbf{x}$; 2) linear ELM-AE and non-linear ELM-AE use orthogonal random hidden node parameters $g(\mathbf{a}_i, b_i)$, while linear SELM-AE and non-linear SELM-AE use sparse random hidden node parameters $g(\mathbf{a}_i, b_i)$. $h_i(\mathbf{x}) = g(\mathbf{a}_i, \mathbf{x}, b_i)$ is the output of i -th hidden node for input $\mathbf{x}$; 3) linear ELM-AE and linear SELM-AE use linear hidden layer activation function, while non-linear ELM-AE and SELM-AE use non-linear hidden layer activation function. 38
- 3.2 Lower dimension projection of linear ELM-AE $\mathbf{X}_{\text{IRIS}}\beta_{\text{AE}}^T$ for iris dataset, calculated by Equation (3.17). The spread of data points belonging to the same cluster is smaller than PCA and NMF projection. 51
- 3.3 Lower dimension projection of non-linear ELM-AE $\mathbf{X}_{\text{IRIS}}\beta_{\text{NAE}}^T$ for iris dataset, calculated by Equation (3.21). The spread of data points belonging to the same cluster is smaller than PCA and NMF projection. 52
- 3.4 Lower dimension projection of linear SELM-AE $\mathbf{X}_{\text{IRIS}}\beta_{\text{SAE}}^T$ for iris dataset, calculated by Equation (3.26). The spread of data points belonging to the same cluster is smaller than PCA and NMF projection. 52
- 3.5 Lower dimension projection of non-linear SELM-AE $\mathbf{X}_{\text{IRIS}}\beta_{\text{NSAE}}^T$ for iris dataset, calculated by Equation (3.30). The spread of data points belonging to the same cluster is smaller than PCA and NMF projection. 53
- 3.6 Lower dimension projection of PCA $\mathbf{X}_{\text{IRIS}}\mathbf{V}^T$ for iris dataset, calculated by Equation (2.14). The spread of data points belonging to the same cluster is larger than linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE projection. 53

- 3.7 Lower dimension projection of NMF $\mathbf{X}_{\text{IRIS}}\mathbf{W}^T$ for iris dataset, calculated by Equation (2.22). The spread of data points belonging to the same cluster is larger than linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE projection. 54
- 3.8 USPS dataset sparsity results of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, PCA, NMF, and TAE. Sparsity of linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE is larger than PCA but smaller than NMF and TAE. Hence linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE is moderately sparse. 66
- 3.9 USPS dataset sparsity results of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE. Linear SELM-AE is sparser than linear ELM-AE, non-linear ELM-AE and non-linear SELM-AE for large number of features. While linear ELM-AE is sparser than non-linear ELM-AE and non-linear SELM-AE for large number of features. 67
- 3.10 CIFAR-10 dataset sparsity results of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, PCA, NMF, and TAE. Sparsity of linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE is larger than PCA but smaller than NMF and TAE. Hence linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE is moderately sparse. 68
- 3.11 CIFAR-10 dataset sparsity results of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE. Linear SELM-AE is sparser than linear ELM-AE, non-linear ELM-AE and non-linear SELM-AE for large number of features. While linear ELM-AE is sparser than non-linear ELM-AE and non-linear SELM-AE for large number of features. 69

- 3.12 NORB dataset sparsity results of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, PCA, NMF, and TAE. Sparsity of linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE is larger than PCA but smaller than NMF and TAE. Hence linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE is moderately sparse. 70
- 3.13 NORB dataset sparsity results of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE. Linear SELM-AE is sparser than linear ELM-AE, non-linear ELM-AE and non-linear SELM-AE for large number of features. While linear ELM-AE is sparser than non-linear ELM-AE and non-linear SELM-AE for large number of features. 71
- 4.1 ML-ELM network structure. (a) Non-linear ELM-AE learns the 1-st layer weights β^1 of ML-ELM using input data $\mathbf{x}$. (b) Non-linear ELM-AE learns the i -th layer weights β^{i+1} of ML-ELM using i -th hidden layer output $\mathbf{h}^i$. (c) Regularized least square learns the ML-ELM output layer weights. 76
- 4.2 Features learned by non-linear ELM-AE resembles input data. The first 20 digits of 0 in this figure is created by taking all the digits 0 in the MNIST dataset as input data for a non-linear ELM-AE with 20 hidden neurons and reshaping the output weights of non-linear ELM-AE to $28 \times 28 \times 20$. The next 20 digits of 1 in this figure is created by taking all the digits of 1 in the MNIST dataset as input data for a non-linear ELM-AE with 20 hidden neurons and reshaping the output weights of non-linear ELM-AE to $28 \times 28 \times 20$ 78
- 4.3 1-st layer weights β_{NAE} of ELM-AE network 784-100-784. All the data of MNIST data is taken as input data for a non-linear ELM-AE with 100 hidden neurons and the output weights of non-linear ELM-AE is reshaped to $28 \times 28 \times 100$ 83

- 4.4 2-nd layer weights β_{NAE} of ELM-AE network 100-100-100 and 1-st layer weights β_{NAE} of ELM-AE 784-100-784. All the data of MNIST data is taken as input data for a non-linear ELM-AE with 100 hidden neurons. Then the output weights of the 1st non-linear ELM-AE is multiplied with input data and sigmoid activation function is applied to create a lower dimensional projection ($\text{sigmoid}(\mathbf{X}\beta_{NAE}^T)$). This lower dimension projection $\text{sigmoid}(\mathbf{X}\beta_{NAE}^T)$ is taken as input data for the second non-linear ELM-AE with 100 hidden neurons. The first non-linear ELM-AE output weights and second non-linear ELM-AE output weights are multiplied to create this figure. 84
- 5.1 LRF-ELMAE Structure. Non-linear ELM-AE learns the convolution layer weights. The pooling layer does not contain any learn able parameters and norm pooling operation is applied. Non-linear ELM-AE learns the fully connected layer weights and regularized least squares learns the output weights. 87
- 5.2 Generating the patch dataset $\mathbf{X}_p$. Randomly select patches of size $r \times r$ from the training data. Normalize these patches by removing the mean and dividing by the standard deviation to create the patch dataset. 88
- 6.1 Response Map Training of ELM-FPD. Get a image patch and generate HOG features for this image patch. For this HOG image patch generate the appropriate response map. If the image patch contains facial point in the middle the appropriate response map is shown in training sample 1. If the image patch contains facial point shifted to the right the appropriate response map is shown in training sample 2. Use the HOG features as input data and response map as output data to train a ELM. 98
- 6.2 ELM-FPD, CFAN, and SDM results of labeled face parts in the wild dataset. ELM-FPD outperforms CFAN and SDM for labeled face parts in the wild dataset. 102

6.3	ELM-FPD, CFAN, and SDM results of ibug dataset. ELM-FPD outperforms CFAN and SDM for ibug dataset.	103
6.4	ELM-FPD, CFAN, and SDM results of HELEN dataset. ELM-FPD outperforms SDM for HELEN dataset.	104
6.5	ELM-FPD, CFAN, and SDM results of Multiple dataset. ELM-FPD outperforms CFAN and SDM for multiple dataset.	105
6.6	ELM-FPD cumulative error plot of each iteration for labeled face parts in the wild dataset. Each iteration improves the performance of ELM-FPD.	106
6.7	ELM-FPD cumulative error plot of each iteration for ibug dataset. Each iteration improves the performance of ELM-FPD.	107
6.8	ELM-FPD cumulative error plot of each iteration for HELEN dataset. Each iteration improves the performance of ELM-FPD.	108
6.9	ELM-FPD cumulative error plot of each iteration for multiple dataset. Each iteration improves the performance of ELM-FPD.	109
6.10	Each row starting from top represents ELM-FPD output of images from labeled face parts in the wild, ibug, and helen datasets respectively. The five columns from left represents face images with occlusion, beard, pose, smile, and sunglasses respectively. The right hand side column with the red outline represents failure cases of ELM-FPD.	110

List of Tables

3.1	The parameter ranges used to determine the best parameters of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, TAE, PCA, and NMF using ELM classifier.	56
3.2	The parameter ranges used to determine the best parameters of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, TAE, PCA, and NMF using linear SVM classifier.	57
3.3	The parameter ranges used to determine the best parameters of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, TAE, PCA, and NMF using Gaussian Kernel ELM classifier.	58
3.4	The testing accuracy, training time, and NMSE of USPS, CIFAR-10, and NORB dataset of ELM-AE (linear and non-linear), SELM-AE (linear and non-linear), TAE, PCA, and NMF using ELM classifier.	63
3.5	The testing accuracy, training time, and NMSE of USPS, CIFAR-10, and NORB dataset of ELM-AE (linear and non-linear), SELM-AE (linear and non-linear), TAE, PCA, and NMF using linear SVM classifier.	64
3.6	The testing accuracy, training time, and NMSE of USPS, CIFAR-10, and NORB dataset of ELM-AE (linear and non-linear), SELM-AE (linear and non-linear), TAE, PCA, and NMF using Gaussian Kernel ELM classifier.	65

4.1	Performance comparison of ML-ELM with state-of-the-art Deep Networks.	82
5.1	LRF-ELMAE network parameters for MNIST and NORB dataset. . . .	93
5.2	Parameter range used to determine the best parameters for LRF-ELMAE.	94
5.3	Testing Accuracy for MNIST and NORB dataset.	95
6.1	Mean NMSE values of ELM-FPD, CFAN, and SDM for labeled face parts in the wild, ibug, and HELEN datasets.	111

List of Abbreviations

AE	Auto-Encoders.
BP	Back-Propagation.
CAE	Contractive Auto-Encoders.
CFAN	Coarse to Fine Auto-eNcoder.
CNN	Convolutional Neural Network.
DAE	Denoising Auto-Encoder.
DBN	Deep Belief Network.
ELM	Extreme Learning Machine.
ELM-AE	Extreme Learning Machine Auto-Encoder.
ELM-FPD	Extreme Learning Machine based Facial Point Detection.
ELM-LRF	Extreme Learning Machine based Local Receptive Field.
ELM-SDM	Extreme Learning Machine Supervised Descent method.
H-ELM	Hierarchical Extreme Learning Machine.
HOG	Histogram Oriented Gradients.

LRF-ELMAE	Local Receptive Field Extreme Learning Machine.
MDAE	Marginalized Denoising Auto-Encoder.
ML-ELM	Multi-Layer Extreme Learning Machine.
MSE	Mean Square Error.
NMF	Non-negative Matrix Multiplication.
NMSE	Normalized Mean Square Error.
PCA	Principal Component Analysis.
RBM	Restricted Boltzmann Machine.
RP	Random Projection.
SAE	Stacked Auto-Encoders.
SDAE	Stacked Denoising Auto-Encoder.
SDM	Supervised Descent Method.
SELM-AE	Sparse Extreme Learning Machine Auto-Encoder.
SLFN	Single Layer Feed forward Neural network.
SVD	Singular Value Decomposition.
SVM	Support Vector Machines.
TAE	Tied weight Auto-Encoder.

Chapter 1

Introduction

1.1 Research Background

Traditionally the relationship between output data and input data is explicitly described by an expert using a set of mathematical operations. In contrast, machine learning algorithms learn the relationship between output data and input data implicitly. Data might contain noise and unwanted information which might affect the generalization capability of machine learning algorithms. Hence before applying machine learning algorithms, important information or features of the data is extracted using feature learning algorithms. Feature learning algorithms learn features while reducing the dimension of data or increasing the dimension of data and categorized as holistic-based or parts-based algorithms. Holistic-based algorithms learn features with decreasing level of importance for all data samples and features with least importance are removed. Principal Component Analysis (PCA) [53, 92] is a holistic-based dimension reduction algorithm which learns features describing the variance information. Hence the first feature of PCA describes the largest variance of data. In contrast, parts-based algorithms learn features which represent parts of the data such as a nose from a face image. Furthermore, the level of importance of parts-based features is different for each data sample.

Non-negative Matrix Multiplication (NMF) [78] and Tied weight Auto-Encoder (TAE) [17] is a commonly used parts-based dimension reduction algorithm.

A commonly used machine learning algorithm is neural networks, and the basic building block of neural networks is the artificial neuron which apes the operation of a neuron in the human brain. Neural networks can be categorized to fully connected and local receptive field networks. Different fully connected neural networks architecture exists based on the connections between neurons such as: 1) Single Layer Feed forward Neural network (SLFN), have a single hidden layer with feed forward connections. Hence input layer connects to hidden layer, and hidden layer connects to output layer; 2) multi-layer neural networks, have multiple hidden layers with feed forward connections, hence input layer connects to first hidden layer, first hidden layer connects to second hidden layer and last hidden layer connects to output layer; 3) Recurrent neural networks, have multi-layer or single layer architecture with feed forward and backward connections. Hence input layer connects to the hidden layer; hidden layer connects to itself and an output layer.

SLFN is the simplest fully connected neural network architecture, and the generalization performance is better than other neural network architectures for some applications such as data analytics. However, an SLFN learns network parameters using Back-Propagation (BP) [99], which might converge to a local minimum and has slow training time. Extreme Learning Machine (ELM) [57, 59, 60, 61, 63, 64, 120] is an SLFN with random hidden layer parameters independent from the input data and not tuned; ELM only learns the output weight. Hence ELM training time is significantly lower than BP. Furthermore, as ELM minimizes both training error and norm of the output weights, according to Bartlett theory [12], ELM tends to achieve better generalization performance. Due to the simple network architecture and simple learning algorithm of ELM, it is used in many applications such as image object detection [13, 21, 23], time series analysis [1, 20, 112, 121], fault detection [85, 116], and remote sensing [91].

Multi-layer neural networks consist of multiple hidden layers, and each layer learns

features of the input data. The first layer of multi-layer neural networks learns abstract features such as lines and curves of images; higher layers learn parts of images such as a nose of a face image. Hinton et al. [50] introduced an effective multi-layer learning algorithm based on Restricted Boltzmann Machine (RBM) [47] and BP. Before this seminal work, multi-layer neural networks' generalization performance was significantly lower than SLFN. The multi-layer neural network learning algorithm consisted of a pre-training step which uses an unsupervised learning algorithm to find the initial hidden layer parameters and fine tune the multi-layer network using BP. In contrast, SLFN does not use a pre-training step and the hidden layer parameters learned by BP.

In contrast to fully connected neural networks, local receptive field network layers have partial connections and are motivated by the visual cortex [65]. These local receptive field networks [44, 71, 76, 81, 94, 104] perform better than fully connected neural networks in image processing tasks [75]. Three types of local receptive field layers are: 1) convolution layer which detects specific parts of the image such as lines and curves; 2) pooling layer which makes the image translation invariant; 3) fully connected layer which combines the output of the pooling layer. Typically the first layer of local receptive field networks is a convolution layer and the second layer is a pooling layer. Local receptive field networks add subsequent layers of convolution and pooling layers based upon cross-validation process, and the final layer of local receptive field network is a fully connected layer.

Feature learning and machine learning algorithms are used in facial point detection to find the location of eyes, mouth, nose, and face contour in a face image [26, 27, 28, 29]. These facial points are used to find head pose [97], facial expressions [105], face hallucination [111], face swapping [19], and face animation [114]. Detecting facial points is a non-linear problem due to occlusion, illumination, facial expressions, and different head poses and uses algorithms such as Supervised Descent Method (SDM) [115] to find facial points. SDM is a generalized version of Gauss-Newton method [115] for non-linear problems. Gauss-Newton method converges with few iterations due to quadratic learning speed. However, Gauss-Newton method requires the function

to twice differentiable and the Hessian of the function to be positive definite. In contrast, SDM can solve non-linear problems such as facial point detection problem which is not twice differentiable and Hessian is not positive. SDM is a linear single layer machine learning algorithm while Coarse to Fine Auto-eNcoder (CFAN) extends SDM to fully connected multi-layer neural network [119]. Hence generalization capability of CFAN is better than SDM.

1.2 Motivation and Objectives

Feature learning algorithms remove noise or unwanted information of input data and improve the generalization capability of machine learning algorithms. The commonly used PCA is a holistic-based dimension reduction feature learning algorithm which learns the variance information and removes features with low variance. The disadvantage of PCA is that it is unable to learn parts-based features of data (i.e. nose of a face image). In contrast NMF and TAE parts-based dimension reduction feature learning algorithm. However, the training time of NMF and TAE is significantly high. Hence it is important to investigate feature learning algorithms with small training time which can learn to an extend parts-based features and holistic-based features.

When the hidden layer neurons of TAE is smaller than the input layer neurons, TAE performs dimension reduction. It has been shown that TAE with linear hidden layer activation function learns the variance information of input data similar to PCA [10] and TAE with non-linear hidden layer activation function learns non-linear features of input data [17]. However, the training time of TAE is high as hidden layer and output layer parameters must be learned. Hence it is important to investigate Auto-Encoders (AE) [15, 16, 17, 24, 43, 70, 80, 95, 96, 109] with random hidden layer parameters which are not tuned.

ELM is a “generalized” SLFN with a wide variety of hidden layer activation functions. In contrast to traditional SLFN training algorithms such as BP, ELM hidden layer pa-

parameters are chosen randomly independent from the input data and not tuned. As only the output weights of ELM must be calculated the training speed of ELM significantly faster than BP. Huang et al. [57, 59, 60, 61, 63, 64, 120] have thoroughly investigated the classification and regression capability of ELM. However, the feature learning capability of ELM is not investigated.

Multi-layer neural network learning algorithm consist of two learning steps: 1) unsupervised feature learning algorithm such as TAE learns the initial hidden layer parameters of the multi-layer neural network; 2) BP learning algorithm fine tunes the hidden layer parameters [17, 50, 109]. As two algorithms are used in multi-layer neural networks the training time is significantly high. Hence it is important to investigate multi-layer neural network learning algorithms with one learning step.

Convolutional Neural Network (CNN) [75] and Extreme Learning Machine based Local Receptive Field (ELM-LRF) [58] are local receptive field networks. Hidden layer parameters of CNN is learned by BP algorithm while the hidden layer parameters of ELM-LRF is chosen randomly and not tuned. Hence the training time ELM-LRF is significantly lower than CNN and the generalization capability of ELM-LRF is higher than CNN for some datasets [9]. Hence it is important to investigate local receptive field networks with better generalization capability and low training time.

Facial point detection learns the location of eye, nose, mouth, and face contour which is represented by 68 facial points. Hence for facial point detection, only a limited amount of training data is available, due to high labeling costs. ELM has been shown to perform well for small and moderate size datasets. SDM and CFAN find facial points using Histogram Oriented Gradients (HOG) [33] values and fully connected neural networks. However finding facial points using HOG values is difficult as HOG values do not contain distance information of facial points. Furthermore, Local Receptive Field networks outperform fully connected neural networks for computer vision problems [75]. Hence this work investigates an ELM based facial point detection framework which resolves the shortcomings of SDM and CFAN.

1.3 Major Contributions

The contributions of the Ph.D. is summarized as follows.

ELM based feature learning algorithms referred to as Extreme Learning Machine Auto-Encoder (ELM-AE) and Sparse Extreme Learning Machine Auto-Encoder (SELM-AE) are introduced based on AE framework. This thesis investigates ELM-AE and SELM-AE with linear hidden layer activation function referred to as linear ELM-AE and linear SELM-AE respectively; ELM-AE and SELM-AE with sigmoid hidden layer activation function referred to as non-linear ELM-AE and non-linear SELM-AE. In contrast to TAE, the hidden layer parameters of linear and non-linear ELM-AE are chosen to be random and orthogonal while the hidden layer parameters of linear and non-linear SELM-AE are chosen to be sparse random. Hence the training time of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE is lower than TAE. In contrast to linear TAE which learns the variance information [10], this work shows that linear ELM-AE and linear SELM-AE, in theory, learns the between-cluster scatter. Hence reducing dimension using linear ELM-AE and non-linear ELM-AE reduces the Euclidean distance between data points belonging to the same cluster while removing features with low variance. In contrast, PCA and linear TAE performs dimension reduction by removing features with low variance. The experimental results show the efficacy of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, PCA, NMF, and non-linear TAE in terms of discriminative capability, sparsity, reconstruction cost and training time.

This thesis extends single layer ELM to a fully connected multi-layer neural network referred as Multi-Layer Extreme Learning Machine (ML-ELM). Non-linear ELM-AE learns the hidden layer parameters of ML-ELM and ridge regression learns the output layer weight of ML-ELM. In contrast to traditional multi-layer neural networks which consist of fine tuning learning step, ML-ELM hidden layer parameters should not be fine tuned using BP. The results show that ML-ELM generalization capability is similar to other multi-layer neural networks with small training time.

This thesis introduces an local receptive field network referred to as Local Receptive Field Extreme Learning Machine (LRF-ELMAE). Non-linear ELM-AE learns the hidden layer parameters of LRF-ELMAE and ridge regression learns the output layer weight of LRF-ELMAE. In contrast, traditional local receptive field networks such as CNN hidden layer parameters and output weight are learned by BP. Experimental results show that LRF-ELMAE achieves state-of-the-art results at a significantly lower training time.

In the final work an Extreme Learning Machine based Facial Point Detection (ELM-FPD) algorithm is introduced which finds facial points as follows: 1) automated face recognition algorithm such as Viola Jones algorithm [110] finds the face location in the image; 2) LRF-ELMAE estimates the facial points on the located faces using mean face shape; 3) Extreme Learning Machine Supervised Descent method (ELM-SDM) improves the estimated facial points using low dimensional response maps. ELM-FPD resolves the following shortcomings of SDM and CFAN: 1) in contrast to SDM and CFAN which uses fully connected neural networks, ELM-FPD uses local receptive field networks as local receptive field networks outperform fully connected neural networks for computer vision problems [75]; 2) in contrast to SDM and CFAN which uses HOG values, ELM-FPD uses response maps as response maps contain distance information of facial points which is important in finding facial points. Traditionally response maps are created by evaluating small image patches as a single output classification or regression problem. Which is time-consuming due to multiple evaluations of small image patches and discriminating small image patches as positive, and negative is difficult as some positive and negative small image patches might be similar due to the limited amount of pixel information. Hence the response maps are generated using a single large image patch as a multi-output regression problem. Experimental results show that ELM-FPD outperforms SDM and CFAN.

1.4 Organization

Thesis is organized as follows:

Chapter 2 presents the literature review of the following: 1) basic ELM and ELM-LRF; 2) PCA, Random Projection (RP), NMF, TAE, and RBM; 3) multi-layer neural networks; 4) facial point detection.

Chapter 3 introduces the ELM based feature learning algorithm referred as ELM-AE and SELM-AE.

Chapter 4 introduces ELM based multi-layer neural network ML-ELM.

Chapter 5 introduces non-linear ELM-AE based local receptive field neural network LRF-ELMAE.

Chapter 6 introduces ELM based facial point detection algorithm ELM-FPD.

Chapter 7 presents the conclusion and future work.

Chapter 2

Literature Review

2.1 Extreme Learning Machines

Back-Propagation (BP) [98] learning algorithm is a commonly used learning algorithm for Single Layer Feed forward Neural network (SLFN), which uses the gradient information to learn parameters. However, the training time of BP algorithm is large and might converge to a local minimum. Huang et al. [57, 59, 60, 61, 63, 64, 120] introduced Extreme Learning Machine (ELM) as a learning algorithm for SLFN and resolves the problems of BP algorithm. In contrast to BP algorithm which learns hidden layer and output layer parameters, ELM theories [57, 59, 60, 61, 63, 64, 120] shows that the hidden layer parameters can be chosen randomly independent of input data and not tuned. Hence only the output layer parameters of ELM must be learned. Huang et al. [59, 60, 62] extended ELM to “generalized” SLFN where the hidden layer activation function is non-linear piecewise continuous and not necessarily neuron alike. Figure 2.1 shows the ELM additive node network and ELM output is calculated as:

$$f_L(\mathbf{x}) = \sum_{i=1}^L \beta_i h_i(\mathbf{x}) = \mathbf{h}(\mathbf{x})\beta \quad (2.1)$$

Figure 2.1: ELM additive type network randomly generates the hidden node parameters $(\mathbf{a}_i, b_i)$. ELM calculates the output of i -th hidden node for input $\mathbf{x}$ as: $h_i(\mathbf{x}) = g(\mathbf{a}_i, \mathbf{x}, b_i)$. ELM hidden layer activation function g is a non-linear piecewise continuous function.

where L are the number of hidden neurons of ELM, $\beta = [\beta_1, \dots, \beta_L]^T$ is the output weights and $\mathbf{h}(\mathbf{x}) = [h_1(\mathbf{x}), \dots, h_L(\mathbf{x})]$ is hidden layer output of input $\mathbf{x} \in \mathbf{R}^d$ and $h_i(\mathbf{x})$ is the output of the i -th hidden neuron. For additive hidden node $h_i(\mathbf{x}) = g(\mathbf{a}_i, \mathbf{x}, b_i)$ where $g(\cdot)$ is piecewise non-linear activation function, $\mathbf{a}_i$ and b_i are fixed random parameters. Huang et al. [59, 60, 62] showed that ELM can use many types of activation functions such as sigmoid, hard-limit, Gaussian, multi-quadratic, Fourier series, and wavelet etc. For N training samples $\{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=1}^N$, ELM solves the following learning problem [57, 63]:

$$\text{Minimize: } \|\beta\|_p^{\sigma_1} + C \|\mathbf{H}\beta - \mathbf{T}\|_q^{\sigma_2} \quad (2.2)$$

where $\sigma_1 > 0$, $\sigma_2 > 0$, $p, q = 0, \frac{1}{2}, 1, 2, \dots, +\infty$, C is user given parameter, $\mathbf{T} = [\mathbf{t}_1, \dots, \mathbf{t}_N]^T$ is the target data and $\mathbf{H} = [\mathbf{h}(\mathbf{x}_1), \dots, \mathbf{h}(\mathbf{x}_N)]^T$ is the hidden layer output. ELM minimizes norm of the output weights and training error. Hence according to Barlett theory [12] minimizing the norm of the output weights improves the generalization capability. When C is infinitely large and $\sigma_1 = \sigma_2 = p = q = 2$, the solution to Equation (2.2) is:

$$\beta = \mathbf{H}^\dagger \mathbf{T} \quad (2.3)$$

where $\mathbf{H}^\dagger$ is the Moore-Penrose generalized inverse of $\mathbf{H}$.

When the number of samples are larger than or equal to the number of hidden neurons $N \geq L$ and $\sigma_1 = \sigma_2 = p = q = 2$, the solution to Equation (2.2) is [57, 63]:

$$\beta = \left(\frac{\mathbf{I}}{C} + \mathbf{H}^T \mathbf{H} \right)^{-1} \mathbf{H}^T \mathbf{T} \quad (2.4)$$

When the number of samples are smaller than the number of hidden neurons $N < L$ and $\sigma_1 = \sigma_2 = p = q = 2$, the solution to Equation (2.2) is:

$$\beta = \mathbf{H}^T \left(\frac{\mathbf{I}}{C} + \mathbf{H} \mathbf{H}^T \right)^{-1} \mathbf{T} \quad (2.5)$$

Support Vector Machines (SVM) [30] and its variants such as least square support vector machines [106] calculates output weights and output bias. Huang et al. [57, 63] showed that as ELM only calculates the output weights (See Equation (2.4) and Equation (2.5)) and ELM has fewer optimization constraints than SVM and its variants. Hence the training time of ELM is significantly lower than SVM and its variants. In contrast to SVM, ELM feature mapping $\mathbf{H}$ is non-parametric, and ELM provides a unified learning architecture for binary class classification, multi class classification and regression [57, 63]. The learning algorithm of ELM is as follows:

Given N training samples $\{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=1}^N$, activation function $g()$ and the number of hidden nodes L :

step 1) Randomly generate the input weight $\mathbf{A} \in \mathbf{R}^{d \times L}$ and bias $\mathbf{b} \in \mathbf{R}^{1 \times L}$;

step 2) Calculate the Hidden layer output $\mathbf{H}$;

step 3) If C is infinitely large calculate ELM output weight using Equation (2.3) else calculate ELM output weight using Equation (2.4) or Equation (2.5);

Algorithm 1: ELM learning algorithm

(a) fully connected architecture.

(b) Local receptive field network architecture.

Figure 2.2: ELM random connection types. a) In a fully connected architecture each neuron connects to the neurons in the previous layer. b) In local receptive field network architecture each neuron connects to some neurons in the previous layer.

2.1.1 Extreme Learning Machine based Local Receptive Field

Huang et al. [57, 59, 60, 61, 63, 64, 120] show that ELM networks can use random hidden layer parameters, random connections between layers, and combines hidden layer neurons. Basic ELM investigates an SLFN with random hidden layer parameters. Hence Extreme Learning Machine based Local Receptive Field (ELM-LRF) [9, 58] investigates multi-layer neural networks with random connections, random weights and combining hidden layer nodes. ELM-LRF random connections form dense connections for some input neurons and sparse connections to other input neurons as shown in Figure 2.2(b). Similar to local receptive field architecture of visual cortex [65], ELM-LRF processes a local region or local receptive field of input data. Furthermore, ELM theories have shown that hidden layer nodes could be combined using combinatorial node.

Figure 2.3: Combinatorial node of ELM-LRF. Neurons of ELM-LRF can be combined using combinatorial node and enforces translation invariance. Commonly used combinatorial nodes are max pooling, average pooling and sum pooling.

Figure 2.3 shows that three groups hidden layer nodes are grouped together by summation. The objective of a combinatorial node of ELM-LRF is to enforce translation invariance [102]. The commonly used translation invariance functions are max pooling, average pooling, and sum pooling. In contrast to other local receptive field networks [44, 71, 74, 76, 81, 94, 104] such as Convolutional Neural Networks (CNN) [74] which uses BP to learn the hidden layer parameters, ELM-LRF uses random hidden layer parameters independent from the input data and only the output layer weights is learned.

Huang et al. [58] showed that any random distribution could generate random connections for ELM-LRF, and combinatorial nodes can use different type of functions. However, ELM-LRF uses step probability distribution to generate the random connections and square-root pooling function for the combinatorial node as shown in Figure 2.4. The first layer of ELM-LRF convolves k orthogonal random filters $\mathbf{A} \in \mathbf{R}^{r^2 \times k}$ with input data to create k feature maps as shown in Figure 2.4. Where $r \times r$ is the size of the receptive field and $d \times d$ is the size of the input image. The size of each feature maps is

Figure 2.4: ELM-LRF network architecture. ELM-LRF generates the first layer connections from a step probability function, ELM-LRF generates the first layer weights from a uniform random distribution, and ELM-LRF uses square root pooling function for the second layer. Linear regression network learns the output layer weights.

$(d - r + 1) \times (d - r + 1)$ and the feature map is calculated as:

$$c_{i,j,k}(\mathbf{x}) = \sum_{m=1}^r \sum_{n=1}^r x_{i+m-1,j+m-1} \cdot \mathbf{a}_{k,m,n} \quad (2.6)$$

$$i, j = 1, \dots, (d - r + 1)$$

Second layer of ELM-LRF applies norm operation on k feature maps as shown in Figure 2.4 and calculated as:

$$h_{p,q,k} = \sqrt{\sum_{i=p-e/2}^{p+e/2} \sum_{j=q-e/2}^{q+e/2} c_{i,j,k}^2} \quad (2.7)$$

Last layer of ELM-LRF concatenates pool maps in the combinatorial layer to create matrix $\mathbf{H} \in \mathbf{R}^{N \times k \cdot (d-r+1)^2}$ and the output weights β is calculated based on the number of rows and columns of matrix $\mathbf{H}$. If the rows of matrix $\mathbf{H}$ is smaller than or equal to the columns ($N \leq k \cdot (d - r + 1)^2$) then the output weight layer weights β is calculated using Equation (2.4) else ($N > k \cdot (d - r + 1)^2$) using Equation (2.5).

2.1.2 Removing Neurons Which Negatively Affect the Generalization Performance of Extreme Learning Machine

Hidden layer parameters of ELM is randomly chosen and not tuned. Hence some neurons might negatively effect the generalization performance of ELM and these neurons are removed by: 1) incremental approaches; 2) pruning approaches; 3) dimension reduction approaches.

Huang et al. [62] introduced Incremental Extreme Learning Machine (I-ELM) which incrementally adds a random hidden neuron and calculates the output weight only for the added hidden neuron.

Yoan et al. [87] introduced a Optimally Pruned Extreme Learning Machine (OP-ELM) which prunes hidden neurons. OP-ELM first ranks the hidden neurons using Least An-

gle regression (LARS) algorithm [37] and removes hidden neurons using Leave One Out Cross validation method. Tikhonov regularized Optimally Pruned Extreme Learning Machine (TROP-ELM) is a extension of OP-ELM which uses Tikhonov regularization to penalize the weights of OP-ELM and improve generalization performance [88].

Hongming et al. [122] showed that unwanted hidden neurons of ELM can be removed by dimension reduction and vertically stacking ELM. This vertically stacked ELM was referred as Stacked ELM and algorithm is as follows: 1) calculate feature mapping $\mathbf{H} = \text{sigmoid}(\mathbf{XA} + \mathbf{b})$; 2 calculate output weights $\beta = (\frac{\mathbf{I}}{C} + \mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T \mathbf{T}$; 3) calculate the eigenvectors of β ; 4) use eigenvectors $\mathbf{V}$ to reduce the dimension of ELM feature mapping $\mathbf{H}^{\text{reduce}} = \mathbf{V}^T \mathbf{H}$; 5) calculate new ELM feature mapping $\mathbf{H}^2 = \text{sigmoid}(\mathbf{XA} + \mathbf{b})$; 6) concatenate the reduced ELM feature mapping with the new ELM feature mapping $\mathbf{H} = [\mathbf{H}^{\text{reduce}}, \mathbf{H}^2]$; 7) calculate output weights $\beta = (\frac{\mathbf{I}}{C} + \mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T \mathbf{T}$ for the concatenated ELM feature mapping; 7) go to step 3.

2.1.3 Deep Extreme Learning Machine

ELM requires large number of hidden layer parameters than neural networks with learned hidden layer parameters. In order to reduce the number of hidden layer parameters of ELM, Tissera et al. [107] introduced a multi-layer ELM referred as Deep ELM.

This section explains the procedure to create a two hidden layer Deep ELM. In order to create a two hidden layer Deep ELM, Deep ELM first creates a three hidden layer Deep ELM and then combine two hidden layers of the three hidden layer network to create a two hidden layer Deep ELM. Deep ELM learns weights in a layer wise manner where the first hidden layer output $\mathbf{H}^1$ is calculated as:

$$\mathbf{H}^1 = \text{sigmoid}(\mathbf{XA}^1); \quad (2.8)$$

where $\mathbf{A}^1 = [\mathbf{a}_1^1, \dots, \mathbf{a}_d^1]^T$ is generated randomly. The second hidden layer weights β^2

of Deep ELM is calculated as:

$$\beta^2 = \left(\frac{\mathbf{I}}{C^1} + \mathbf{H}^{1T} \mathbf{H}^1 \right)^{-1} \mathbf{H}^{1T} \mathbf{X} \quad (2.9)$$

Figure 2.6 shows the network architecture of Deep ELM with one hidden layer.

Figure 2.5: Deep ELM network structure with one hidden layer. The first hidden layer weights $\mathbf{A}^1$ is generated randomly. Equation (2.9) calculates the second hidden layer weights β^2 of Deep ELM.

The third hidden layer output $\mathbf{H}^3$ of Deep ELM is calculated as:

$$\mathbf{H}^3 = \text{sigmoid}(\mathbf{H}^1 \beta^2 \mathbf{A}^3); \quad (2.10)$$

where $\mathbf{A}^3 = [\mathbf{a}_1^3, \dots, \mathbf{a}_d^3]^T$ is generated randomly. The fourth hidden layer weights β^4 of Deep ELM is calculated as:

$$\beta^4 = \left(\frac{\mathbf{I}}{C^3} + \mathbf{H}^{3T} \mathbf{H}^3 \right)^{-1} \mathbf{H}^{3T} \mathbf{T} \quad (2.11)$$

Figure 2.6 shows the network architecture of Deep ELM with three hidden layers.

As the second and third hidden layer of a three layer Deep ELM is connected with linear hidden neurons the second and third hidden layer can be connected by multiplication $\beta^2 \mathbf{A}^3$. Multiple layers can be added and Figure 2.7 shows the complete Deep ELM network structure after combining the second and third hidden layers.

Even though Deep ELM requires less number of hidden layer parameters than ELM, it

Figure 2.6: Deep ELM network structure with three hidden layers. The first hidden layer weights $\mathbf{A}^1$ is generated randomly. Equation (2.9) calculates the second hidden layer weights β^2 of Deep ELM. Third hidden layer weights $\mathbf{A}^3$ is generated randomly. Equation (2.11) calculates the output layer weights of three hidden layer Deep ELM.

Figure 2.7: Deep ELM hidden layers can be combined to create a compact network structure. Deep ELM with three hidden layers can be combined to finally create a two hidden layer network. This is done by multiplying the linearly connected hidden layer weights ($\beta^2 \mathbf{A}^3$). Hence the final Deep ELM network has only non-linear hidden layers.

still requires large number of hidden layer parameters. To further reduce the number of hidden layer parameters of Deep ELM, Tissera et al. [108] introduced Back Propagation Deep ELM which uses BP to reduce the number of hidden layer parameters.

2.1.4 Extreme Learning Machine for Data Visualization

The main objective of data visualization to visualize high dimensional data in a 2 or 3 dimensional space. Anton et al. [4] has extended ELM to map high dimensional data to a visualization space (2 or 3 dimensional space) using an algorithm named ELMVIS. As the exact location of data points in the visualization space is not important, location of data points in the visualization space is randomly decided. ELMVIS algorithm then

calculates the mean square error between the high dimensional data and visualization space. Next the visualization space data points are randomly permuted and the mean square error between the high dimensional data and permuted visualization space is calculated. If the mean square error between the high dimensional data and permuted visualization space is smaller than the high dimensional data and visualization space, then the random permutation is kept otherwise discarded. ELMVIS algorithm performs these random permutations multiple times to find the appropriate visualization space. Anton et al. [5] also introduced a computationally efficient algorithm referred to as ELMVIS+ for data visualization using ELM.

2.1.5 Extreme Learning Machine for Clustering

The main objective of clustering algorithms is to find classes or clusters from input data with out using label information. ELM based clustering algorithms can be divided in to three categories: 1) learn clusters by maximizing the margin between clusters; 2) learn clusters using graph laplacian; 3) learn clusters by reformulating the clustering problem as a traveling sales man problem.

Zhang et al. [118] introduced a clustering algorithm referred to as Extreme Maximum Margin Clustering (EMMC) algorithm which find clusters by maximizing the margin between clusters. EMMC algorithm minimizes the following objective function: $\text{Minimize}_{\beta, \mathbf{T}} \|\mathbf{H}\beta - \mathbf{T}\|^2$. EMMC algorithm has to find both output weights β and clusters $\mathbf{T}$. Hence EMMC uses alternative minimization approach to learn output weights β and clusters $\mathbf{T}$ as: 1) find output weights β for fixed labels $\mathbf{T}$; 2) find output labels $\mathbf{T}$ for fixed output weights β . Huang et al. [54] showed that weight information of data points can be used to improve the clustering information of EMMC algorithm.

Huang et al. [56] introduced a clustering algorithm referred to as Un-Supervised Extreme Learning Machine (US-ELM) which uses graph laplacian to find clusters. In contrast to EMMC algorithm which has to find two unknown parameters, US-ELM has

to learn output weights β to find clusters. US-ELM solves the following generalized eigenvalue problem: $(\mathbf{I} + \lambda \mathbf{H}^T \mathbf{L} \mathbf{H})\beta = \gamma \mathbf{H}^T \mathbf{H} \beta$. where $\mathbf{L}$ is the graph laplacian of input data $\mathbf{X}$, γ is the generalize eigenvalue and λ is a user given parameter. Experimental results show that US-ELM outperforms other graph laplacian clustering algorithms. Peng et al. [93] showed that US-ELM only uses local information to find clusters and this might cause over fitting problems. Hence, Peng et al. [93] introduced a clustering algorithm referred to as Unsupervised Discriminative Extreme Learning Machine (UD-ELM) which uses both local and global information to find clusters.

Yoan et al. [86] introduced a clustering algorithm referred to as Self Organizing Maps Extreme Learning Machine (SOM-ELM). SOM-ELM formulates the clustering problem as a traveling sales man problem and solves it using heuristic optimization. In contrast to EMMC and US-ELM which maps input data to clusters, SOM-ELM maps clusters to input data (reversed formulation).

2.2 Feature Learning

The objective of feature learning algorithms is to learn important features while removing unwanted information or noise from input data by reducing the dimension of input data or increasing the dimension of input data. Lee et al. [78] categorizes feature learning algorithms as holistic-based or parts-based algorithms. Holistic-based algorithms such as Principal Component Analysis (PCA) [53, 92] and Random Projection (RP) [2, 68, 69] learn features which describe all the input data samples (i.e. global features), while parts-based algorithms such as Non-negative Matrix Factorization (NMF) [78] and Tied weight Auto-Encoder (TAE) [17] learn features which describe an individual part of a input data sample (i.e. local features). Subsequent sub sections describes PCA, RP, NMF, and TAE in detail.

2.2.1 Principal Component Analysis

The objective of PCA is to remove dimensions with low variance and retain dimensions of high variance in multi-dimensional input data, by projecting the input data with an orthogonal matrix [53, 92]. This orthogonal matrix consist of linearly independent vectors named as eigenvectors and the number of eigenvectors are always equal to the number of dimensions of the input data. PCA associates each eigenvector with a scalar value referred as eigenvalue which describes the amount of variance. For dimension reduction eigenvectors with low eigenvalue is removed and input data is projected with these reduced eigenvectors to create a projected low dimensional space ($d > L$) referred to as principal components. First principal component describes the highest variance, second principal component describes the second highest variance of input data, and so on. Hence PCA performs dimension reduction by removing principal components with small eigenvalue (low variance). PCA calculates the 1-st eigenvector $\mathbf{V}_1$ of input data $\mathbf{X}$ as:

$$\mathbf{V}_1 = \underset{\|\mathbf{V}_1\|=1}{\text{Maximize}} \left(\frac{\mathbf{V}_1^T (\mathbf{X} - \text{mean}(\mathbf{X}))^T (\mathbf{X} - \text{mean}(\mathbf{X})) \mathbf{V}_1}{\mathbf{V}_1^T \mathbf{V}_1} \right) \quad (2.12)$$

PCA calculates subsequent eigenvectors $\mathbf{V}_j$ of input data $\mathbf{X}$ as:

$$\begin{aligned} \hat{\mathbf{X}}_{j-1}^T &= \mathbf{X} - \sum_{i=1}^{j-1} \mathbf{X} \mathbf{V}_{j-1} \mathbf{V}_{j-1}^T \\ \mathbf{V}_j &= \underset{\|\mathbf{V}\|=1}{\text{Maximize}} \left(\frac{\mathbf{V}_j^T \hat{\mathbf{X}}_{j-1}^T \hat{\mathbf{X}}_{j-1} \mathbf{V}_j}{\mathbf{V}_j^T \mathbf{V}_j} \right) \end{aligned} \quad (2.13)$$

A prerequisite of applying PCA is that the input data $\mathbf{X}$ should be standardized to have zero mean. PCA creates principal components (projected lower dimensional space) of the input data $\mathbf{X}$ by projecting the input data $\mathbf{X}$ with the selected eigenvectors $\mathbf{V}_j$ as:

$$\mathbf{X}_{\text{proj}} = \mathbf{X} \mathbf{V}_j \quad (2.14)$$

2.2.2 Random Projection

Johnson-Lindenstrauss lemma [69] state that a linear mapping can reduce the dimension of input data while retaining the Euclidean distance and is the foundation for RP. Objective of Johnson-Lindenstrauss lemma is:

$$(1 - \varepsilon) \|\mathbf{x}_i - \mathbf{x}_j\|^2 \leq \|\mathbf{x}_i \mathbf{A} - \mathbf{x}_j \mathbf{A}\|^2 \leq (1 + \varepsilon) \|\mathbf{x}_i - \mathbf{x}_j\|^2 \quad (2.15)$$

where $0 < \varepsilon < 1$ and $\mathbf{A}$ is a random matrix. Hence the objective of RP is to reduce the dimension by projecting the input data along a random matrix. RP [2, 68, 69] shows that only certain types of random matrix satisfy Equation (2.15), and RP uses these random matrices for dimension reduction: 1) orthogonal random matrix [69]; 2) random matrix generated from a Gaussian distribution with mean zero and standard deviation one [68]; 3) sparse random matrix [2]. RP represents orthogonal random matrix $\mathbf{A}$ as [69]:

$$\mathbf{A}^T \mathbf{A} = \mathbf{I} \quad (2.16)$$

RP represents Gaussian random matrix as [68]:

$$\mathbf{A} \sim N(0, 1) \quad (2.17)$$

RP represents Sparse random matrix as [2]:

$$a_{ij} = 1/\sqrt{L} \begin{cases} +\sqrt{3} & p = 1/6 \\ 0 & p = 2/3 \\ -\sqrt{3} & p = 1/6 \end{cases} \quad (2.18)$$

Sparse random matrix contains three values ($+\sqrt{3}$, 0, and $-\sqrt{3}$) and p represents the ratio of coefficients in the sparse random matrix $\mathbf{A}$. Hence $\frac{2}{3}$ of the sparse random matrix coefficients are zero, $\frac{1}{6}$ of the sparse random matrix coefficients are $\sqrt{\frac{3}{L}}$, and $\frac{1}{6}$ of the sparse random matrix coefficients are $-\sqrt{\frac{3}{L}}$.

The time taken to generate sparse random weights are lower than Gaussian random weights and time taken to generate Gaussian random matrix is lower than the orthogonal random matrix. RP calculates projected lower dimensional space $\mathbf{X}_{\text{proj}}$ as:

$$\mathbf{X}_{\text{proj}} = \mathbf{XA} \quad (2.19)$$

2.2.3 Non-negative Matrix Factorization

Biological neurons have only positive firing rates, and the sign of biological neurons do not change from positive to negative or negative to positive in its lifetime [78]. However, feature learning algorithms can have negative firing rates, and the sign of neurons can change from positive to negative in the learning phase. NMF [78] resolves these shortcomings by constraining the learning problem to learn only positive weights or features, which will generate only positive sign neurons and have positive firing rates. As positive features constrain NMF to add features to generate input data, features contain parts of input data. NMF learns features while reducing the dimension of input data.

NMF learns two set of matrices referred as positive coefficient matrix $\mathbf{A}$ and positive basis matrix $\mathbf{W}$ from positive input matrix $\mathbf{X}$. NMF represents the square loss learning problem as:

$$\text{Minimize: } \|\mathbf{X} - \mathbf{AW}\|^2 \quad (2.20)$$

$$\text{Subject to: } \mathbf{A} \geq 0, \mathbf{W} \geq 0$$

NMF calculates the positive basis matrix $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_d]$ and the positive coefficient matrix $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_L]$ using multiplicative update rules iteratively, for k iterations as:

$$\begin{aligned} \mathbf{A}^{k+1} &= \mathbf{A}^k \frac{\mathbf{XW}^T}{\mathbf{A}^k \mathbf{W}^{k+1} (\mathbf{W}^{k+1})^T} \\ \mathbf{W}^{k+1} &= \mathbf{W}^k \frac{(\mathbf{A}^k)^T \mathbf{X}}{(\mathbf{A}^k)^T \mathbf{A}^k \mathbf{W}^k} \end{aligned} \quad (2.21)$$

Positive basis matrix $\mathbf{W}$ learns the most occurring parts of input data and the positive coefficient matrix $\mathbf{A}$ indicates how the positive basis matrix $\mathbf{W}$ is combined to generate the input data $\mathbf{X}$. The main disadvantage of NMF is that NMF converges to a local minimum [79]. NMF creates the projected lower dimensional space $\mathbf{X}_{\text{proj}}$ by projecting the input data $\mathbf{X}$ along the positive basis matrix $\mathbf{W}$ as:

$$\mathbf{X}_{\text{proj}} = \mathbf{X}\mathbf{W}^T \quad (2.22)$$

2.2.4 Tied Weight Auto-Encoder

Auto-Encoder (AE) [10, 15, 16, 17, 24, 43, 51, 70, 73, 80, 95, 96, 109] is feature learning algorithm which can learn features by either reducing the dimension of input data or increasing the dimension of input data. AE is a neural network with output data is equal to input data, and the objective of AE is to learn the intrinsic features of input data. AE can have multiple hidden layers, but the single hidden layer AE is the most common and this thesis focuses on single layer AE. TAE is a single layer AE with the same weights for input layer and output layer [17]. The input layer or encoder of TAE maps the input data to hidden layer representation, and the output layer or decoder maps the hidden layer representation to input data as shown in Figure 2.8. As shown in Figure 2.8 TAE learns features of input data in three architectures: 1) compressed architecture; 2) sparse architecture; 3) equal dimension architecture.

Compressed architecture

TAE learns features which reduce the dimension of input data by having a smaller number of hidden layer neurons than the input neurons.

Sparse architecture

TAE learns features which increase the dimension of input data by having a larger number of hidden layer neurons than the input neurons.

Equal dimension architecture

TAE learns features which do not change the dimension of input data by having the same number of hidden layer neurons as the input neurons.

Figure 2.8: TAE network architecture. TAE uses the same weights β_{TAE} for the encoder stage and decoder stage. TAE learns features in three representations: 1) Compressed architecture; 2) Equal dimension representation; 3) Sparse architecture.

TAE calculates the encoder output as:

$$\mathbf{Y} = g(\mathbf{X}\beta_{\text{TAE}} + \mathbf{b}) \quad (2.23)$$

where $g()$ is a non-linear activation function such as sigmoid activation function. TAE calculates the decoder stage output as:

$$\mathbf{Z} = g(\mathbf{Y}\beta_{\text{TAE}}^T + \mathbf{c}) \quad (2.24)$$

where $\mathbf{c} = [c_1, \dots, c_d]$ is the bias vector of the output nodes and $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_d]$ is the output of TAE. TAE decoder activation function could be a linear activation function or sigmoid activation function and minimizes the squared error objective given by $L_2(\mathbf{X}, \mathbf{Z}) = \|\mathbf{X} - \mathbf{Z}\|^2$. TAE performs dimension reduction by multiplying the input data $\mathbf{X}$ with the encoder stage weights β_{TAE} as:

$$\mathbf{X}_{\text{proj}} = \mathbf{X}\beta_{\text{TAE}} \quad (2.25)$$

2.2.5 Restricted Boltzmann Machine (RBM)

Similar to AE, Restricted Boltzmann Machine (RBM) [47] also learns features by either reducing the dimension of input data or increasing the dimension of input data. In contrast to PCA, RP, NMF, and TAE, RBM is an energy based model where each possible input has an attached energy function. RBM is a special case of Boltzmann Machine (BM) [3] where there are no connections between binary input nodes and binary hidden nodes as shown in Figure 2.9. Similar to TAE, RBM is a parts-based algorithm which learns non-linear relationships of input data. RBM calculates the probability as:

$$P(\mathbf{x}) = \sum_h \frac{e^{-Energy(\mathbf{x}, \mathbf{h})}}{Z} \quad (2.26)$$

$\mathbf{x} = [x_1 \cdots x_d]$ are the visible nodes, $\mathbf{h} = [h_1 \cdots h_L]$ are the hidden nodes of RBM and Z is the partition function given by $Z = \sum_{\mathbf{x}, \mathbf{h}} e^{-Energy(\mathbf{x}, \mathbf{h})}$. RBM calculates the energy as:

Figure 2.9: RBM network architecture. In a RBM the input nodes and hidden nodes connected, but there are no connection between input nodes or between hidden nodes.

$$Energy(\mathbf{x}, \mathbf{h}) = -\mathbf{b}^T \mathbf{x} - \mathbf{c}^T \mathbf{h} - \mathbf{h}^T \mathbf{W} \mathbf{x} \quad (2.27)$$

Where $\mathbf{b} = [b_1, \cdots, b_d]$ are the bias of the visible nodes, $\mathbf{c} = [c_1, \cdots, c_L]$ are the bias of the hidden nodes and $\mathbf{W} = [W_1, \cdots, W_L]$ are the weights between input nodes and hidden nodes. RBM calculates the marginal probability as:

$$P(\mathbf{x}) = \sum_h \frac{e^{-Energy(\mathbf{x}, \mathbf{h})}}{Z} \quad (2.28)$$

RBM calculates the gradient of the likelihood $P(\mathbf{x})$ for one data sample as:

$$\begin{aligned}
\frac{\partial}{\partial \theta} \log(P(\mathbf{x})) &= \frac{\partial}{\partial \theta} \log(P(\mathbf{x})) - \frac{\partial}{\partial \theta} \sum_{x,h} \log(Z) \\
\frac{\partial}{\partial \theta} \log(P(\mathbf{x})) &= \frac{1}{P(\mathbf{x})} \frac{\partial}{\partial \theta} P(\mathbf{x}) - \frac{1}{Z} \sum_{x,h} \frac{\partial}{\partial \theta} P(\mathbf{x}, \mathbf{h}) \\
\frac{\partial}{\partial \theta} \log(P(\mathbf{x})) &= \frac{1}{P(\mathbf{x})} \frac{\partial}{\partial \theta} \sum_h P(\mathbf{x}, \mathbf{h}) - \frac{1}{Z} \sum_{x,h} \frac{\partial}{\partial \theta} P(\mathbf{x}, \mathbf{h}) \\
\frac{\partial}{\partial \theta} \log(P(\mathbf{x})) &= \sum_h \frac{P(\mathbf{x}, \mathbf{h})}{P(\mathbf{v})} \frac{\partial}{\partial \theta} \log P(\mathbf{x}, \mathbf{h}) - \frac{1}{Z} \sum_{x,h} P(\mathbf{x}, \mathbf{h}) \frac{\partial}{\partial \theta} \log P(\mathbf{x}, \mathbf{h})
\end{aligned} \tag{2.29}$$

By using $\frac{P(\mathbf{x}, \mathbf{h})}{P(\mathbf{v})} = P(\frac{\mathbf{h}}{\mathbf{x}})$ and Equation (2.29):

$$\frac{\partial}{\partial \theta} \log(P(\mathbf{x})) = \sum_h P(\frac{\mathbf{h}}{\mathbf{x}}) \frac{\partial}{\partial \theta} \log P(\mathbf{x}, \mathbf{h}) - \frac{1}{Z} \sum_{x,h} P(\mathbf{x}, \mathbf{h}) \frac{\partial}{\partial \theta} \log P(\mathbf{x}, \mathbf{h}) \tag{2.30}$$

Equation (2.30) has two parts, and the first part can be solved analytically as $P(\frac{\mathbf{h}}{\mathbf{x}})$ is conditionally independent in an RBM. $P(\frac{\mathbf{h}}{\mathbf{x}}) = \text{sigmoid}(\mathbf{c} + \mathbf{W}\mathbf{x})$. The second part require an exponential number of summations (2^d where d is the dimension of the input data x). Hence the second part is solved using Gibbs sampler [6, 40]. As performing Gibbs sampling for large sample data is time-consuming, an approximation is performed by running Gibbs sampling for 1 step instead of $k = \infty$ steps. This approximation is commonly referred as Contrastive Divergence and empirically has shown to perform well [22].

2.3 Multi-Layer Neural Networks

BP algorithm learns the weights of neural networks based gradient information of the error and passes the gradient information from the last layer to the first layer in a backward manner. Traditionally multi-layer neural networks are trained in one step; which is to initialize the multi-layer neural network with random hidden layer weights and tune this random weights using BP. In this case, BP fails to learn useful hidden layer weights

as the gradient information vanish or explode [52].

In 2006 Hinton et al. [50] introduced a novel two-step method to train multi-layer neural networks. In the first step, an unsupervised learning algorithm learns the hidden layer weights of the multi-layer neural network. In the second step, BP is applied to the multi-layer neural network as a fine tuning step to improve the hidden layer weights. Deep Belief Network (DBN) [50] is a multi-layer neural network which uses RBM generative neural network for unsupervised learning, and Stacked Auto-Encoders (SAE) [17] is a multi-layer neural network which uses TAE discriminative neural network for unsupervised learning. DBN and SAE use BP to fine tune the hidden layer weights.

Figure 2.10 shows that in the first step, RBM learns the hidden layer weights of DBN. DBN learns first hidden layer weights of DBN $\mathbf{W}^1$ by an RBM which models input data $\mathbf{X}$, and DBN learns second hidden layer weights of DBN $\mathbf{W}^2$ by an RBM which models the first hidden layer $\mathbf{h}^1$. Hence, DBN learns i -th layer DBN weights $\mathbf{W}^i$ by an RBM which models $(i - 1)$ -th hidden layer $\mathbf{h}^{i-1}$. Hence RBM creates DBN by stacking weights. Stacking RBM is useful as it has shown to improve the lower bound of the previous model if the number of dimensions in the previous model is equal to the number of dimensions in the current model [48]. DBN learns last layer weights by a softmax classifier and BP fine tunes the whole network.

Figure 2.11 shows that in the first step, TAE learns the hidden layer weights of SAE. TAE learns the first hidden layer weights of SAE β^1 by a TAE which models input data $\mathbf{X}$, and TAE learns the second hidden layer weights β^2 of SAE by a TAE which models hidden layer $\mathbf{h}^1$. Where $\mathbf{h}^1$ is the output of the first TAE. Hence, TAE learns i -th layer SAE weights β^i of SAE by a TAE which models $(i - 1)$ -th hidden layer $\mathbf{h}^{i-1}$. TAE creates SAE by stacking weights. Softmax classifier learns the last layer weights of SAE and BP fine tunes the whole network.

Figure 2.10: DBN network architecture. (a) RBM learns the first hidden layer weights of DBN W^1 by modeling input data $\mathbf{x}$. (b) RBM learns $(i+1)$ -th hidden layer weights of DBN W^{i+1} by modeling the output of i -th hidden layer $\mathbf{h}^i$. (c) Softmax classifier learns the output layer weights of DBN and BP fine tunes the whole network.

2.4 Facial Point Detection

Facial point detection algorithms such as Active Shape Models [26, 28, 29], Active Appearance Models [27], Constrained Local Models [31, 32], and Supervised Descent Method (SDM) [115] aims to learn the location of eyes, mouth, nose and face contour in a face image. Facial points is used in applications such as head pose detection [97], facial expression detection [105], face animation [114], face swapping [19], and face hallucination [111]. Learning the facial points is a non-linear optimization problem and difficult due to occlusion, illumination, facial expressions, and different head poses.

Figure 2.11: SAE network architecture. (a) TAE learns the SAE first hidden layer weights β^1 by modeling input data $\mathbf{x}$. (b) TAE learns the SAE $(i+1)$ -th hidden layer weights β^{i+1} by modeling i -th hidden layer $\mathbf{h}^i$. (c) Softmax classifier learns the last layer weights of SAE and BP fine tunes the whole network.

Non-linear problems can be solved by Gauss-Newton method, but requires the optimization function to be twice differentiable and Hessian to be positive definite [115]. SDM resolves these shortcomings and finds facial points.

2.4.1 Supervised Descent Method

The objective of SDM is to minimize the following non-linear facial point detection problem:

$$f(\mathbf{t}_0) = ||h(\mathbf{t}_0) - h(\mathbf{t}_*)||_2^2 \quad (2.31)$$

where $h()$ is a feature extraction algorithm such as Histogram of Oriented Gradients (HOG) [33] or scale invariant feature transform [83] features, $\mathbf{t}_* \in \mathbf{R}^{2m}$ is ground truth

facial points and $\mathbf{t}_0 \in \mathbf{R}^{2m}$ is the estimated facial points. If $h()$ is HOG feature extraction algorithm, then $h(\mathbf{t}_*)$ is the HOG values around each facial point represented by ground truth facial points $\mathbf{t}_*$; while $h(\mathbf{t}_0)$ is the HOG values around each facial point represented by estimated facial points $\mathbf{t}_0$.

Solution to Equation (2.31) is found by differentiating Equation (2.31) with respect $\Delta\mathbf{t}$:

$$\frac{\partial f(\mathbf{t}_0)}{\partial \Delta\mathbf{t}} = 2 * (h(\mathbf{t}_0) - h(\mathbf{t}_*)) \frac{\partial (h(\mathbf{t}_0) - h(\mathbf{t}_*))}{\partial \Delta\mathbf{t}} \quad (2.32)$$

As $h(\mathbf{t}_*)$ is a constant Equation (2.32) can be simplified to:

$$\frac{\partial f(\mathbf{t}_0)}{\partial \Delta\mathbf{t}} = 2 * (h(\mathbf{t}_0) - h(\mathbf{t}_*)) \frac{\partial (h(\mathbf{t}_0))}{\partial \Delta\mathbf{t}} \quad (2.33)$$

Equation (2.33) is expressed as:

$$\mathbf{J}_f = 2 * (h(\mathbf{t}_0) - h(\mathbf{t}_*)) \mathbf{J}_h \quad (2.34)$$

As $h(\mathbf{t}_0)$ cannot be differentiated directly with respect to $\Delta\mathbf{t}$, the Taylor expansion of $f(\mathbf{t}_0)$ is differentiated with respect to $\Delta\mathbf{t}$:

$$\begin{aligned} f(\mathbf{t}_0) &\approx f(\mathbf{t}_0) + \mathbf{J}_f(\mathbf{t}_0) \Delta\mathbf{t} + \frac{1}{2} \Delta\mathbf{t}^T \mathbf{H}_f(\mathbf{t}_0) \Delta\mathbf{t} \\ \frac{\partial (f(\mathbf{t}_0))}{\partial \Delta\mathbf{t}} &= \mathbf{J}_f(\mathbf{t}_0) + \frac{1}{2} \Delta\mathbf{t}^T \mathbf{H}_f(\mathbf{t}_0) \end{aligned} \quad (2.35)$$

where $\mathbf{J}_f$ and $\mathbf{H}_f$ are the Jacobian and Hessian of function f evaluate at $\mathbf{t}_0$. By substituting $\frac{\partial f(\mathbf{t}_0)}{\partial \Delta\mathbf{t}} = 0$ to Equation (2.35):

$$\Delta\mathbf{t}^T = -2 * \mathbf{J}_f \mathbf{H}_f^{-1} \quad (2.36)$$

By substituting Equation (2.34) in equation (2.36):

$$\begin{aligned} \Delta\mathbf{t}^T &= -4(h(\mathbf{t}_0) - h(\mathbf{t}_*)) \mathbf{J}_h \mathbf{H}_f^{-1} \\ \Delta\mathbf{t}^T &= -4(h(\mathbf{t}_0)) \mathbf{J}_h \mathbf{H}_f^{-1} + 4(h(\mathbf{t}_*)) \mathbf{J}_h \mathbf{H}_f^{-1} \end{aligned} \quad (2.37)$$

As $h(\mathbf{t}_*)$ is a constant we can represent $4(h(\mathbf{t}_*))\mathbf{J}_h\mathbf{H}_f^{-1}$ as $\mathbf{b}_{\text{sdm}}$ and $-4\mathbf{J}_h\mathbf{H}_f^{-1}$ as β_{sdm} :

$$\Delta\mathbf{t}^T = (h(\mathbf{t}_0))\beta_{\text{sdm}} + \mathbf{b}_{\text{sdm}} \quad (2.38)$$

Equation (2.38) is the SDM solution to facial point detection problem and finds a local minimum. SDM requires couple of iterations to find a global minimum and calculated as:

$$\Delta\mathbf{t}_{\text{iter}}^T = (h(\mathbf{t}_{0_{\text{iter}}}))\beta_{\text{sdm}} + \mathbf{b}_{\text{sdm}} \quad (2.39)$$

where iter is the number of iterations, $\Delta\mathbf{t}_1 = 0$ and $\mathbf{t}_{0_{\text{iter}}} = \mathbf{t}_{0_{\text{iter}-1}} + \Delta\mathbf{t}_{\text{iter}-1}$. SDM finds facial points as follows: 1) automatic face recognition algorithm such as Viola Jones algorithm [110] finds the face locations; 2) mean training facial points estimates the facial points for the located faces; 3) linear regression network with learn-able parameters β_{sdm} and $\mathbf{b}_{\text{sdm}}$ improves the estimated facial points using HOG values.

SDM uses coarse to fine strategy to remove unwanted information [115], as unwanted information affects the generalization performance of machine learning algorithms. The objective of coarse to fine strategy is to remove unwanted information by reducing the search area of the image [119]. Reducing search area can be done by: 1) use same image resolution and reduce the patch size; 2) increase the resolution of the image for each iteration and use the same patch size. SDM uses the second method as increasing the image resolution increases the quality of the image (high-resolution images are clearer than low-resolution images). Hence coarse to fine SDM calculates approximated facial points from a higher resolution face image than the face image resolution used previous iteration as:

$$\Delta\mathbf{t}_{\text{iter}}^T = (h(\mathbf{t}_{0_{\text{iter}}}))\beta_{\text{sdm}_{\text{iter}}} + \mathbf{b}_{\text{sdm}_{\text{iter}}} \quad (2.40)$$

In contrast to SDM, Equation (2.40) shows that coarse to fine SDM requires a different linear regression network for each iteration; i.e., SDM requires ten linear regression

networks for ten iterations. However experimental results have shown coarse to fine SDM typically requires five iterations [115].

SDM is a linear regression network and might not be able to capture non-linear relationships. To resolve this disadvantage of SDM, Coarse to Fine Auto eNcoder (CFAN) [119] learns non-linear relationships using a fully connected multi-layer neural network as:

$$\Delta \mathbf{t}^T = g_{ae_{level}}(g_{ae_2}(g_{ae_1}(h(\mathbf{t}_0), \beta_{cfan_1}), \beta_{cfan_2}), \beta_{cfan_{level}}) \quad (2.41)$$

where CFAN $level$ -th layer non-linear activation function is given by $g_{ae_{level}}()$. CFAN finds facial points as follows: 1) automatic face recognition algorithm such as Viola Jones algorithm finds the face locations; 2) fully connected multi-layer neural network estimates the facial points for the located faces; 3) fully connected multi-layer neural network improves the estimated facial points using HOG values.

2.5 Summary

This chapter consist of four sections: 1) ELM; 2) Dimension Reduction; 3) Multi-layer Neural Networks; 4) Facial Point Detection. ELM is an efficient learning algorithm for SLFN, where a random distribution selects the hidden layer parameters independent from input data. In contrast to SVM, ELM has less amount of constraints, lower training time and generalization capability similar or better than SVM. In contrast to fully connected SLFN, local receptive field networks perform better on computer vision problems. ELM-LRF extends ELM to local receptive field networks by generating random hidden neurons, random hidden layer connections, and combining hidden layer nodes.

The objective of feature learning is to remove unwanted information of input data while preserving important information by reducing the dimension of input data or increasing the dimension of input data. There is two type of feature learning algorithms: 1)

holistic-based; 2) parts-based. PCA is a holistic-based algorithm which preserves the variance information of input data. RP is a holistic-based algorithm which preserves the Euclidean information of input data. NMF is a parts-based algorithm which preserves the most occurring parts of input data such as nose in a face image. TAE and RBM are parts-based algorithm which learns non-linear relationships in input data.

SAE and DBN multi-layer neural networks learn network parameters (hidden layer and output layer) in two steps. In the first step, an unsupervised feature learning algorithm learns the hidden layer parameters. In the second step, BP fine tunes the network parameters. Hence the training time of SAE and DBN is large.

The objective of facial point detection is to detect the location of eye, mouth nose, and face contour. Facial point detection is a non-linear optimization problem which might not be twice differentiable and have a non-positive definite Hessian due to: 1) illumination; 2) occlusion; 3) different head poses; 4) facial expressions. Gauss-Newton non-linear problem solver cannot find facial points, as it cannot solve non-linear optimization problems which are not twice differentiable and Hessian is not positive definite. SDM and CFAN algorithms resolve the shortcomings of Gauss-Newton algorithm and are capable of finding facial points.

Chapter 3

Extreme Learning Machine based Dimension Reduction

Huang et al. [57, 59, 60, 61, 63, 64, 120] have thoroughly investigated the regression and classification capability of ELM, while Gao et al. [55, 56] have investigated the clustering capability of ELM. Hence this thesis investigates feature learning capability of ELM using Extreme Learning Machine Auto-Encoder (ELM-AE) and Sparse Extreme Learning Machine Auto-Encoder (SELM-AE)¹ with a different type of hidden layer activation functions. ELM-AE and SELM-AE with linear hidden layer activation function are referred to as linear ELM-AE and linear SELM-AE respectively. ELM-AE and SELM-AE with sigmoid hidden layer activation function are referred to as non-linear ELM-AE and non-linear SELM-AE respectively. Linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE learns features while reducing the dimension of input data or increasing the dimension of input data or having the same dimension of input data. However, this section investigates the dimension reduction capability of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE. Experi-

¹This thesis investigates SELM-AE as experimental results have shown that learning sparse features is beneficial [15]. SELM-AE learns sparse features which contain few large coefficient values and many small coefficient values but is not sparse in the sense of generalization capability. Hence, SELM-AE cannot be compared with Sparse Extreme Learning Machine [8].

mental results show that linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE outperforms PCA, TAE, and NMF.

3.1 Extreme Learning Machine based Dimension Reduction

ELM-AE (linear and non-linear) and SELM-AE (linear and non-linear) uses orthogonal random weights and sparse random weights in the hidden layer respectively and uses linear regression to learn output weight. The output data of ELM-AE (linear and non-linear) and SELM-AE (linear and non-linear) is equal to input data. The motivation of using random weights in ELM-AE (linear and non-linear) and SELM-AE (linear and non-linear) is based on ELM universal approximation capability [59, 60, 61, 63, 64, 120]:

Lemma 3.1 Universal approximation capability [59, 60, 61, 63, 64, 120] *If the hidden layer activation function of SLFN is bounded non-constant piecewise continuous and if tuned parameters of the hidden layer can approximate any target continuous function $f(\mathbf{x}) \in C(K)$ where K is the compact subset in $\mathbb{R}^d$ and $C(K)$ represents the space of all continuous function in K , then any randomly generated function sequence $\{h_i(\mathbf{x})\}_{i=1}^L$ with appropriate output weights β can approximate any continuous target function with probability one $\lim_{L \rightarrow \infty} \|\sum_{i=1}^L \beta_i h_i(\mathbf{x}) - f(\mathbf{x})\| = 0$.*

Lemma 3.1 shows that a randomly generated function sequence $\{h_i(\mathbf{x})\}_{i=1}^L$ with appropriate output weights β can approximate any continuous function $f(\mathbf{x}) \in C(K)$, hence different objective functions of ELM is used to realize different type of tasks such as regression [64], classification [63], and clustering [55, 56]. ELM regression and ELM classification the objective function is given by Equation (2.2), while ELM clustering

objective function is:

$$\begin{aligned} & \underset{\beta}{\text{Minimize}} \quad \|\beta\|^2 + \lambda \text{Tr}(\beta^T \mathbf{H}^T \mathbf{L} \mathbf{H} \beta) \\ & \text{s.t.} \quad (\mathbf{H}\beta)^T \mathbf{H}\beta = \mathbf{I} \end{aligned} \quad (3.1)$$

where $\mathbf{L}$ is the graph Laplacian of input data $\mathbf{X}$ and λ is a user given parameter. Hence a different ELM objective function can realize feature learning.

The motivation of using orthogonal random weights and sparse random weights in ELM-AE (linear and non-linear) and SELM-AE (linear and -non-linear) is based on the mutual information maximization between input layer and hidden layer output principle of AE [109]. To this end, based on RP theories [2, 69] ELM-AE (linear and non-linear) uses orthogonal random hidden layer parameters and SELM-AE (linear and non-linear) uses sparse random hidden layer parameters to retain the Euclidean information between input data $\mathbf{X}$ and input data projection along hidden layer weights $\mathbf{XA}$. Mutual information is maximized between input data and input data projection along hidden layer weights in ELM-AE (linear and non-linear) and SELM-AE (linear and non-linear), but not between input data and hidden layer output. Hence ELM-AE (linear and non-linear) and SELM-AE (linear and non-linear) follows the mutual information maximization principle of AE loosely.

3.1.1 Linear Extreme Learning Machine Auto-Encoder

Linear ELM-AE calculates hidden layer parameters for dimension reduction as:

$$\begin{aligned} \mathbf{h}(\mathbf{x}) &= \mathbf{x}\mathbf{A} + \mathbf{b} = [h_1(\mathbf{x}), \dots, h_L(\mathbf{x})] \\ &= [\mathbf{a}_1 \cdot \mathbf{x} + b_1, \dots, \mathbf{a}_L \cdot \mathbf{x} + b_L] \\ \mathbf{A}^T \mathbf{A} &= \mathbf{I} \\ \mathbf{b}^T \mathbf{b} &= 1 \end{aligned} \quad (3.2)$$

Figure 3.1: Differences between the proposed ELM based AE framework (linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE) and the original ELM are: 1) the output of linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE is equal to the input data $\mathbf{x}$; 2) linear ELM-AE and non-linear ELM-AE use orthogonal random hidden node parameters $g(\mathbf{a}_i, b_i)$, while linear SELM-AE and non-linear SELM-AE use sparse random hidden node parameters $g(\mathbf{a}_i, b_i)$. $h_i(\mathbf{x}) = g(\mathbf{a}_i, \mathbf{x}, b_i)$ is the output of i -th hidden node for input $\mathbf{x}$; 3) linear ELM-AE and linear SELM-AE use linear hidden layer activation function, while non-linear ELM-AE and SELM-AE use non-linear hidden layer activation function.

where $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_L]$ is the orthogonal random weight matrix between the input nodes and hidden nodes and $\mathbf{b} = [b_1, \dots, b_L]$ is the orthogonal bias vector of the hidden nodes. In contrast to TAE, the hidden layer parameters (encoder stage parameters) of linear ELM-AE is not tuned.

Linear ELM-AE calculates output weights (decoder stage parameters) β_{AE} as:

$$\underset{\beta_{\text{AE}}}{\text{Minimize: }} \|\mathbf{H}\beta_{\text{AE}} - \mathbf{X}\|^2 \quad (3.3)$$

Equation (3.3) is equal to Equation (2.2) when $\mathbf{T} = \mathbf{X}$ and C is infinitely large. Hence the solution to Equation (3.3) is given by Equation (2.3) when $\mathbf{T} = \mathbf{X}$ and is:

$$\beta_{\text{AE}} = \mathbf{H}^\dagger \mathbf{X} \quad (3.4)$$

Theorem 3.1 shows that linear ELM-AE with zero bias ($b_i = 0$), orthogonal random input weight, and input neurons larger than hidden neurons ($d > L$) learns the between-

cluster scatter of data. In contrast, linear AE and PCA learns the variance information of data; NMF learns parts information of data and TAE learns non-linear relationships of data. The derivation of Theorem 3.1 is motivated by linear AE [10].

Theorem 3.1 *Linear ELM-AE with zero bias ($b_i = 0$), orthogonal random input weights $\mathbf{A}^T \mathbf{A} = \mathbf{I}$, and input neurons larger than hidden neurons ($d > L$) minimizes the objective function $\underset{\beta_{AE}}{\text{Minimize}} \|\mathbf{X} \mathbf{A} \beta_{AE} - \mathbf{X}\|^2$ and learns the between-cluster scatter of data multiplied by orthogonal random matrix $\beta_{AE} = \mathbf{A}^T \mathbf{V} \mathbf{V}^T$, where $\mathbf{V}$ are the eigenvectors of covariance matrix $\mathbf{X}^T \mathbf{X}$.*

Proof: As shown in Figure 3.1, linear ELM-AE performs dimension reduction by using the function: $\underset{\beta_{AE}}{\text{Minimize}} \|\mathbf{X} \mathbf{A} \beta_{AE} - \mathbf{X}\|^2$, where $\beta_{AE} = [\beta_{AE1}, \dots, \beta_{AE L}]^T$ is the output weight matrix between the hidden nodes and the output nodes, $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_L]$ is the orthogonal random input weight and $\{(\mathbf{x}_i)_{i=1}^N\}$ are the input and output data. It is reasonable to assume that the input data matrix $\mathbf{X}$ has full rank in most applications.

The normal equations of the convex function $\underset{\beta_{AE}}{\text{Minimize}} \|\mathbf{X} \mathbf{A} \beta_{AE} - \mathbf{X}\|^2$ is given by:

$$\mathbf{A}^T \mathbf{X}^T \mathbf{X} \mathbf{A} \beta_{AE} = \mathbf{A}^T \mathbf{X}^T \mathbf{X}. \quad (3.5)$$

The $d > L$ orthogonal random matrix is created by performing Singular Value Decomposition (SVD) [41] on a $d \times d$ orthogonal random matrix where $\mathbf{A}_d^T \mathbf{A}_d = \mathbf{A}_d \mathbf{A}_d^T = \mathbf{I}$ and removing $d - L$ singular values. Hence based on Eckart-Young-Mirsky Theorem [36, 89] $\mathbf{A} \mathbf{A}^T$ is the best SVD low rank approximation of $\mathbf{A}_d \mathbf{A}_d^T$ and $\mathbf{A} \mathbf{A}^T \approx \mathbf{I}$. However, note that $\mathbf{A}^T \mathbf{A} = \mathbf{I}$ for any d and L where $d > L$.

$$\mathbf{X}^T \mathbf{X} \mathbf{A} \beta_{AE} = \mathbf{X}^T \mathbf{X} \quad (3.6a)$$

$$\mathbf{A} = \beta_{AE}^T (\beta_{AE} \beta_{AE}^T)^{-1} \quad (3.6b)$$

From Equation (3.6b) and $\mathbf{A}^T \mathbf{A} = \mathbf{I}$, we have:

$$\beta_{AE} = \mathbf{A}^T \beta_{AE}^T (\beta_{AE} \beta_{AE}^T)^{-1} \beta_{AE} \quad (3.7)$$

In Equation (3.7) $P_{\beta_{AE}} = \beta_{AE}^T (\beta_{AE} \beta_{AE}^T)^{-1} \beta_{AE}$ is the orthogonal projection matrix of β_{AE} . Hence $\beta_{AE} = \mathbf{A}^T P_{\beta_{AE}}$. The covariance of input data $\mathbf{X}$ is $C_{\mathbf{X}\mathbf{X}} = \mathbf{V} \Sigma \mathbf{V}^T$, $\mathbf{V} = [\mathbf{V}_1 \cdots \mathbf{V}_d]$ and $\Sigma = [\sigma_1 \cdots \sigma_d]$, where $\mathbf{V}_i$ and σ_i are the eigenvectors and eigenvalues of covariance matrix $\mathbf{X}^T \mathbf{X}$, respectively. The orthogonal projection $P_{\beta_{AE}}$ have a relationship with the orthogonal projection $P_{\beta_{AE}, \mathbf{V}}$ as:

$$P_{\beta_{AE}, \mathbf{V}} = \mathbf{V}^T \beta_{AE}^T (\beta_{AE} \mathbf{V} \mathbf{V}^T \beta_{AE}^T)^{-1} \beta_{AE} \mathbf{V} \quad (3.8)$$

By substituting $P_{\beta_{AE}} = \beta_{AE}^T (\beta_{AE} \beta_{AE}^T)^{-1} \beta_{AE}$ and $\mathbf{V} \mathbf{V}^T = \mathbf{I}$ to Equation (3.8), we get the following relationship:

$$P_{\beta_{AE}, \mathbf{V}} = \mathbf{V}^T P_{\beta_{AE}} \mathbf{V} \quad (3.9a)$$

$$P_{\beta_{AE}} = \mathbf{V} P_{\beta_{AE}, \mathbf{V}} \mathbf{V}^T \quad (3.9b)$$

Equation (3.6a) can be rearranged as:

$$\mathbf{X}^T \mathbf{X} \mathbf{A} \beta_{AE} = \mathbf{X}^T \mathbf{X} \quad (3.10a)$$

$$C_{\mathbf{X}\mathbf{X}} \mathbf{A} \beta_{AE} = C_{\mathbf{X}\mathbf{X}} \quad (3.10b)$$

$$\beta_{AE}^T \mathbf{A}^T C_{\mathbf{X}\mathbf{X}} \mathbf{A} \beta_{AE} = \beta_{AE}^T \mathbf{A}^T C_{\mathbf{X}\mathbf{X}} \quad (3.10c)$$

β_{AE} in Equation (3.10c) can be replaced with Equation (3.7) as:

$$P_{\beta_{AE}} \mathbf{A} \mathbf{A}^T C_{\mathbf{X}\mathbf{X}} \mathbf{A} \mathbf{A}^T P_{\beta_{AE}} = P_{\beta_{AE}} \mathbf{A} \mathbf{A}^T C_{\mathbf{X}\mathbf{X}} \quad (3.11a)$$

$$(P_{\beta_{AE}} \mathbf{A} \mathbf{A}^T C_{\mathbf{X}\mathbf{X}} \mathbf{A} \mathbf{A}^T P_{\beta_{AE}})^T = C_{\mathbf{X}\mathbf{X}} \mathbf{A} \mathbf{A}^T P_{\beta_{AE}} \quad (3.11b)$$

From Equation(3.11a) and Equation (3.11b), we get the following relationship:

$$\begin{aligned} P_{\beta_{AE}} \mathbf{A} \mathbf{A}^T C_{\mathbf{X}\mathbf{X}} \mathbf{A} \mathbf{A}^T P_{\beta_{AE}} &= P_{\beta_{AE}} \mathbf{A} \mathbf{A}^T C_{\mathbf{X}\mathbf{X}} \\ &= C_{\mathbf{X}\mathbf{X}} \mathbf{A} \mathbf{A}^T P_{\beta_{AE}} \end{aligned} \quad (3.12)$$

By using Equation (3.12) and Equation (3.9b), we can get the following relationship:

$$\mathbf{V} P_{\beta_{AE}, \mathbf{V}} \mathbf{V}^T \mathbf{A} \mathbf{A}^T \mathbf{V} \Sigma \mathbf{V}^T = \mathbf{V} \Sigma \mathbf{V}^T \mathbf{A} \mathbf{A}^T \mathbf{V} P_{\beta_{AE}, \mathbf{V}} \mathbf{V}^T \quad (3.13a)$$

$$P_{\beta_{AE}, \mathbf{V}} \mathbf{V}^T \mathbf{A} \mathbf{A}^T \mathbf{V} \Sigma = \Sigma \mathbf{V}^T \mathbf{A} \mathbf{A}^T \mathbf{V} P_{\beta_{AE}, \mathbf{V}} \quad (3.13b)$$

In Equation (3.13b) $P_{\beta_{AE}, \mathbf{V}} \mathbf{V}^T \mathbf{A} \mathbf{A}^T \mathbf{V}$ is diagonal matrix with rank L with eigenvalues 1 (L times) and eigenvalue 0 ($d - L$ times). Because Σ is a diagonal with eigenvalues $\sigma_1 > \sigma_2 > \dots > \sigma_d > 0$. Hence, $P_{\beta_{AE}, \mathbf{V}} \mathbf{V}^T \mathbf{A} \mathbf{A}^T \mathbf{V} = \mathbf{I}_L$ with L number of one's (1) and $d - L$ zeros (0) in the diagonal.

$$P_{\beta_{AE}, \mathbf{V}} \mathbf{V}^T \mathbf{A} = \mathbf{I}_L \mathbf{V}^T \mathbf{A} \quad (3.14)$$

Equation (3.14) shows that $P_{\beta_{AE}, \mathbf{V}} = \mathbf{I}_L$. By substituting $P_{\beta_{AE}, \mathbf{V}} = \mathbf{I}_L$ and $\beta_{AE} = \mathbf{A}^T P_{\beta_{AE}}$ to Equation (3.9b), the output weights β_{AE} of linear ELM-AE is:

$$\beta_{AE} = \mathbf{A}^T \mathbf{V} \mathbf{V}^T \quad (3.15)$$

where $\mathbf{V}$ is a $d \times L$ matrix.

This completes the proof. ■

The between-cluster scatter is calculated as:

$$S_B = \mathbf{M} \mathbf{M}^T \quad (3.16)$$

where $\mathbf{M} = (\mathbf{m}_1, \dots, \mathbf{m}_m)$ and $\mathbf{m}_i \in \mathbf{R}^{d \times 1}$ is the mean of class i . Ding et al. [35] showed that projecting data along the between-cluster scatter $\mathbf{X} \mathbf{M} \mathbf{M}^T$ reduces the distances be-

tween data points belonging to the same cluster and that $\mathbf{M}\mathbf{M}^T = \mathbf{V}\mathbf{V}^T \in \mathbf{R}^{d \times d}$. Hence projecting input data along the output weights $\mathbf{X}\beta_{AE}^T \in \mathbf{R}^{N \times L}$ is equivalent to projecting input data along the between-cluster scatter multiplied by a random matrix $\mathbf{X}\mathbf{V}\mathbf{V}^T\mathbf{A}$. Furthermore, projecting input data along the output weights $\mathbf{X}\beta_{AE}^T$ reduces the distance between input data points belonging to the same cluster and reduces the dimension of input data. In contrast, projecting input data along between-cluster scatter $\mathbf{X}\mathbf{V}\mathbf{V}^T \in \mathbf{R}^{N \times d}$ only reduces the distance between input data points belonging to the same cluster, but does not reduce the dimension of input data. In contrast to PCA, linear ELM-AE groups input data points belonging to the same cluster.

Linear ELM-AE achieves dimension reduction by projecting the data $\mathbf{X}$ along output weights β_{AE} as:

$$\mathbf{X}_{\text{proj}} = \mathbf{X}\beta_{AE}^T \quad (3.17)$$

3.1.2 Non-linear Extreme Learning Machine Auto-Encoder

Non-linear ELM-AE calculates hidden layer parameters (encoder stage weights) for dimension reduction as:

$$\begin{aligned} \mathbf{h}(\mathbf{x}) &= g(\mathbf{x}\mathbf{A} + \mathbf{b}) = [h_1(\mathbf{x}), \dots, h_L(\mathbf{x})] \\ &= [g(\mathbf{a}_1 \cdot \mathbf{x} + b_1), \dots, g(\mathbf{a}_L \cdot \mathbf{x} + b_L)] \\ \mathbf{A}^T \mathbf{A} &= \mathbf{I} \\ \mathbf{b}^T \mathbf{b} &= 1 \end{aligned} \quad (3.18)$$

where $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_L]$ is the orthogonal random weight matrix between the input nodes and hidden nodes. $\mathbf{b} = [b_1, \dots, b_L]$ is the orthogonal bias vector of the hidden nodes and $g(\cdot)$ is a non-linear activation function such as sigmoid.

Non-linear ELM-AE calculates output weights (decoder stage parameters) β_{NAE} as:

$$\underset{\beta_{\text{NAE}}}{\text{Minimize: }} \|\mathbf{H}\beta_{\text{NAE}} - \mathbf{X}\|^2 \quad (3.19)$$

Equation (3.19) is equal to Equation (2.2) when $\mathbf{T} = \mathbf{X}$ and C is infinitely large. Hence the solution to Equation (3.19) is given by Equation (2.3) when $\mathbf{T} = \mathbf{X}$ and is:

$$\beta_{\text{NAE}} = \mathbf{H}^\dagger \mathbf{X} \quad (3.20)$$

Non-linear ELM-AE performs dimension reduction by projecting the data $\mathbf{X}$ along output weights β_{NAE} as:

$$\mathbf{X}_{\text{proj}} = \mathbf{X}\beta_{\text{NAE}}^T \quad (3.21)$$

3.1.3 Linear Sparse Extreme Learning Machine Auto-Encoder

The main objective of linear SELM-AE is to learn sparse features, and the advantage of learning sparse features is as follows [15]:

- (1) Sparse features are more efficient than dense features: As the number of features differs in each data sample, dense features might capture unwanted information.
- (2) The entropy of sparse features is lower than dense features: High entropy indicates that features are not unique or important. Hence it is important to learn low entropy features.
- (3) Sparse features are interpretable: In contrast to sparse features, dense features are not interpretable as many features are not zero or small values. Visual cortex has shown to learn interpretable features [66], hence learning interpretable features for image data is important.

Parts-based algorithms such as TAE and NMF learns sparse features, while holistic-based algorithms such as PCA and RP learns dense features. Linear SELM-AE learns sparse output weights β_{SAE} by imposing sparse random parameters in the hidden layer. Achlioptas *et al.* [2] theories show that sparse random parameters preserve the Euclidean distance between input data and projection of input data along hidden layer weights $\mathbf{XA}$ while introducing sparsity. Linear SELM-AE calculates hidden layer parameters as:

$$\begin{aligned}\mathbf{h}(\mathbf{x}) &= \mathbf{x}\mathbf{A} + \mathbf{b} = [h_1(\mathbf{x}), \dots, h_L(\mathbf{x})] \\ &= [\mathbf{a}_1 \cdot \mathbf{x} + b_1, \dots, \mathbf{a}_L \cdot \mathbf{x} + b_L] \\ a_{ij} = b_i &= 1/\sqrt{L} \begin{cases} +\sqrt{3} & p = 1/6 \\ 0 & p = 2/3 \\ -\sqrt{3} & p = 1/6 \end{cases} \end{aligned} \quad (3.22)$$

where $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_L]$ is the sparse random matrix between the input nodes and the hidden nodes and $\mathbf{b} = [b_1, \dots, b_L]$ is the bias vector of the hidden nodes. Linear SELM-AE calculates output weights β_{SAE} as:

$$\text{Minimize}_{\beta_{SAE}}: \|\mathbf{H}\beta_{SAE} - \mathbf{X}\|^2 \quad (3.23)$$

Equation (3.23) is equal to Equation (2.2) when $\mathbf{T} = \mathbf{X}$ and C is infinitely large. Hence the solution to Equation (3.23) is given by Equation (2.3) when $\mathbf{T} = \mathbf{X}$ and is:

$$\beta_{SAE} = \mathbf{H}^\dagger \mathbf{X} \quad (3.24)$$

Theorem 3.2 shows that output weights β_{SAE} of linear SELM-AE learns the between-cluster scatter of input data projected along sparse random matrix. While linear ELM-AE learns the between-cluster scatter projected along orthogonal random matrix.

Theorem 3.2 *Linear SELM-AE with zero bias ($b_i = 0$), sparse random input weights, and input neurons larger than hidden neurons ($d > L$) minimizes the function $\text{Minimize}_{\beta_{SAE}}: \|\mathbf{XA}\beta_{SAE} - \mathbf{X}\|^2$ and learns the between-cluster scatter of input data multiplied by sparse random*

matrix $\beta_{SAE} = \mathbf{A}^T \mathbf{V} \mathbf{V}^T$, where $\mathbf{V}$ are the eigenvectors of covariance matrix $\mathbf{X}^T \mathbf{X}$.

Proof: We first show that $\mathbf{A}^T \mathbf{A} = \mathbf{I}$ as sparse random matrix $\mathbf{A}$ is not orthogonal.

By [2], we have $E[\mathbf{A}] = 0$ and $\mathbf{I} = \text{var}(\mathbf{A})$. Hence, $\mathbf{I} = \text{var}(\mathbf{A}) = E[\mathbf{A}^T \mathbf{A}] - (E(\mathbf{A}))^2$ and $\mathbf{I} = \text{var}(\mathbf{A}) = E[\mathbf{A}^T \mathbf{A}]$.

$$\begin{aligned} \mathbf{A}^T \mathbf{A} &= \text{cov}(\mathbf{A}, \mathbf{A}) = E[\mathbf{A}^T \mathbf{A}] - E[\mathbf{A}^T] E[\mathbf{A}] \\ \mathbf{A}^T \mathbf{A} &= \text{cov}(\mathbf{A}, \mathbf{A}) = \mathbf{I} \end{aligned} \tag{3.25}$$

As $\mathbf{A}^T \mathbf{A} = \mathbf{I}$ and linear SELM-AE objective function is similar to linear ELM-AE objective function; we can to show that $\beta_{SAE} = \mathbf{A}^T \mathbf{V} \mathbf{V}^T$ by following the proof in Theorem 3.1. ■

Linear SELM-AE output weights β_{SAE} is sparser than linear ELM-AE output weights β_{AE} as linear SELM-AE output weights β_{SAE} is the between-cluster scatter matrix projected along sparse random weights $\beta_{SAE} = \mathbf{A}^T \mathbf{V} \mathbf{V}^T$.

Dimension reduction is achieved by projecting the data $\mathbf{X}$ along linear SELM-AE output weights β_{SAE} as:

$$\mathbf{X}_{\text{proj}} = \mathbf{X} \beta_{SAE}^T \tag{3.26}$$

3.1.4 Non-linear Sparse Extreme Learning Machine Auto-Encoder

Non-linear SELM-AE calculates hidden layer parameters (encoder stage weights) for dimension reduction as:

$$\begin{aligned}
 \mathbf{h}(\mathbf{x}) &= g(\mathbf{x}\mathbf{A} + \mathbf{b}) = [h_1(\mathbf{x}), \dots, h_L(\mathbf{x})] \\
 &= [g(\mathbf{a}_1 \cdot \mathbf{x} + b_1), \dots, g(\mathbf{a}_L \cdot \mathbf{x} + b_L)] \\
 a_{ij} = b_i &= 1/\sqrt{L} \begin{cases} +\sqrt{3} & p = 1/6 \\ 0 & p = 2/3 \\ -\sqrt{3} & p = 1/6 \end{cases} \quad (3.27)
 \end{aligned}$$

where $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_L]$ is the sparse random weight matrix between the input nodes and hidden nodes. $\mathbf{b} = [b_1, \dots, b_L]$ is the sparse bias vector of the hidden nodes and $g()$ is a non-linear activation function such as sigmoid.

Non-linear SELM-AE calculates output weights β_{NSAE} as:

$$\text{Minimize: } \|\mathbf{H}\beta_{\text{NSAE}} - \mathbf{X}\|^2_{\beta_{\text{NSAE}}} \quad (3.28)$$

Equation (3.28) is equal to Equation (2.2) when $\mathbf{T} = \mathbf{X}$ and C is infinitely large. Hence the solution to Equation (3.28) is given by Equation (2.3) when $\mathbf{T} = \mathbf{X}$ and is:

$$\beta_{\text{NSAE}} = \mathbf{H}^\dagger \mathbf{X} \quad (3.29)$$

Non-linear SELM-AE performs dimension reduction by projecting the data $\mathbf{X}$ along output weights β_{NSAE} as:

$$\mathbf{X}_{\text{proj}} = \mathbf{X}\beta_{\text{NSAE}}^T \quad (3.30)$$

3.1.5 ELM-AE (Linear and Non-Linear) and SELM-AE (Linear and Non-Linear) Function as a Denoiser

The objective of AE is to learn important features of input data by reducing the dimension of input data (compressed representation) or increasing the dimension of input data (sparse representation) or without changing the dimension of input data (equal dimension representation). However, certain AE network architecture such as equal dimension linear AE (input neurons are equal to hidden neurons) learns the trivial identity function, which is an unimportant feature of input data. Constraints are applied on AE to learn important features of input data and such constraints are:

- (1) Applying a regularization term: Contractive Auto-Encoders (CAE) [96] learns features which are invariant to small variations in input data by including Frobenius norm of the Jacobian matrix regularization term.
- (2) Randomly corrupting the input data: Denoising Auto-Encoder (DAE) [109] and Marginalized Denoising Auto-Encoder (MDAE) [24] randomly corrupts the input data to learn features which are robust to noise. To this end, DAE and MDAE objectives functions are $\|g(\tilde{\mathbf{X}}\beta^T)\beta - \mathbf{X}\|^2$ and $\|\tilde{\mathbf{X}}\beta^T - \mathbf{X}\|^2$ respectively, where $\tilde{\mathbf{X}}$ is the randomly corrupted input data.

Linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE learn features robust to noise implicitly, while DAE and MDAE learn features robust to noise explicitly. To elaborate this, hidden layer operations of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE are described in detail as follows:

- (1) Multiplying the input data with orthogonal random data (linear ELM-AE and non-linear ELM-AE) or sparse random weights (linear SELM-AE and non-linear SELM-AE) retains the Euclidean distance information and does not corrupt the input data.

- (2) Adding the orthogonal bias or sparse random bias randomly corrupts the input data.
- (3) The non-linear activation function of non-linear ELM-AE and non-linear SELM-AE corrupts the input data non-linearly. Linear ELM-AE and linear SELM-AE corrupts input data linearly, as linear ELM-AE and linear SELM-AE uses linear hidden layer activation function.

Hence linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE learns features which are invariant to noise. Linear ELM-AE and linear SELM-AE is a linear denoiser, while non-linear ELM-AE and non-linear SELM-AE is a non-linear denoiser. In contrast to DAE which randomly corrupts the input data in the input layer, linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE corrupts the input data in the hidden layer. Furthermore in contrast to MDAE which is a two layer neural network that learns features in equal dimension representation, linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE are three layer neural networks which learn features in compressed, sparse, and equal dimension representation.

3.2 Experiments

The efficacy of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE is measured via discriminative capability, sparsity, training time, and Normalized Mean Square Error (NMSE):

- (1) Discriminative Capability: It is important to measure whether the features learned by the dimension reduction algorithm improves the generalization capability. To this end, lower dimension projection is created by projecting the input data along the features learned by the dimension reduction algorithm. Equation (2.14) Equation (2.22), Equation (2.25), Equation (3.17), Equation (3.21), Equation (3.26)

and (3.30) creates lower dimension projections of PCA, NMF, TAE, linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE respectively. The lower dimension projections are classified using ELM classifier, linear SVM classifier and Gaussian Kernel ELM classifier. A high classification testing accuracy of the lower dimensional projection shows that the dimension reduction algorithm aids the classification algorithm in finding the appropriate classes. The average testing accuracy of thirty trials is reported for linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, TAE, PCA, and NMF algorithm.

- (2) Sparsity: Sparse features have shown to be more important than dense features [15]. Sparsity can be measured by counting the number of non-zero coefficients in the features, but this method cannot find the sparsity of features if all the coefficients in the features are non-zero. Hence L_2/L_1 [67] measure is used to calculate the sparsity of features with non-zero coefficient values. L_2/L_1 measure assigns a large value to small value coefficients in features and a small value to large value coefficients in features. Hence a high L_2/L_1 value indicates high sparsity and that the dimension reduction algorithm learns parts-based features. While a low L_2/L_1 value indicates low sparsity and that the dimension reduction algorithm learns holistic-based features. Moderate L_2/L_1 value indicates that the dimension reduction algorithm to an extent learns both parts-based and holistic-based features. L_2/L_1 measure is calculated as: $L_2/L_1 = \frac{\sqrt{\sum_{ij} \beta_{ij}^2}}{\sum_{ij} |\beta_{ij}|}$
- (3) Training Time: Low training time indicates the dimension reduction algorithm can learn features quickly.
- (4) NMSE: NMSE measure the ability of the dimension reduction algorithm to reconstruct the input data from the learned features. A low NMSE indicates that reconstructed data is similar to input data and is an indication that the learned features are important. NMSE is calculated as: $\text{NMSE} = \sum_{i=1}^N (\mathbf{X}_i - \hat{\mathbf{X}}_i)^2 / (N \times (\mathbf{X}_{\max} - \mathbf{X}_{\min}))$ where $\mathbf{X}_{\max}$ and $\mathbf{X}_{\min}$ is the largest value and smallest value of input data $\mathbf{X}$, respectively.

The experiments were carried out in a high-end computer with a Xeon E5-2650 2 GHz processor and 256 GB RAM running MATLAB 2015b. Built-in MATLAB functions for PCA, NMF and TAE was used in the experiments.

3.2.1 Benchmark Datasets

Experiments of TAE, PCA, NMF, linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE have been carried on the following three datasets:

- (1) USPS handwritten dataset [34] contains gray scale images. The size of these images is 16×16 and represents digits from 0 to 9. USPS dataset have 7291 training examples and 2007 testing data samples.
- (2) CIFAR-10 object recognition dataset [75] contains color images. The size of these images is $3 \times 32 \times 32$ and represents objects belonging to 10 classes. CIFAR-10 dataset have 40000 training data samples, 10000 samples for validation and 10000 testing data samples.
- (3) NORB object recognition dataset [77] contains stereo images. The size of these images is $2 \times 96 \times 96$ and represents objects belonging to 5 classes. NORB dataset have 24300 training data samples and 24300 testing data samples. For our experiments, we down sampled the NORB dataset to $2 \times 32 \times 32$.

Images from these three datasets were normalized to have zero mean and unit variance for TAE, PCA, linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE, while for NMF images was normalized between 0 and 1.

Figure 3.2: Lower dimension projection of linear ELM-AE $\mathbf{X}_{\text{IRIS}}\beta_{\text{AE}}^T$ for iris dataset, calculated by Equation (3.17). The spread of data points belonging to the same cluster is smaller than PCA and NMF projection.

3.2.2 Visual Investigation of Features Learned by ELM-AE and SELM-AE

Linear ELM-AE and linear SELM-AE performs dimension reduction by projecting the input data along the between-cluster scatter multiplied by orthogonal or sparse random weights $\mathbf{X}\mathbf{V}\mathbf{V}^T\mathbf{A}$ respectively and reduces the Euclidean distance between data points belonging to the same cluster. This is visually verified by using iris dataset [38], which contain 150 samples from three classes. Figure 3.2, Figure 3.3, Figure 3.4 and Figure 3.5 shows that spread of data points belonging to the same class in linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE lower dimension projection is smaller than the spread of data points belong to the same class in PCA and NMF lower dimension projection (as shown in Figure 3.6 and Figure 3.7).

Figure 3.3: Lower dimension projection of non-linear ELM-AE $\mathbf{X}_{\text{IRIS}}\beta_{\text{NAE}}^{\text{T}}$ for iris dataset, calculated by Equation (3.21). The spread of data points belonging to the same cluster is smaller than PCA and NMF projection.

Figure 3.4: Lower dimension projection of linear SELM-AE $\mathbf{X}_{\text{IRIS}}\beta_{\text{SAE}}^{\text{T}}$ for iris dataset, calculated by Equation (3.26). The spread of data points belonging to the same cluster is smaller than PCA and NMF projection.

Figure 3.5: Lower dimension projection of non-linear SELM-AE $\mathbf{X}_{\text{IRIS}}\beta_{\text{NSAE}}^T$ for iris dataset, calculated by Equation (3.30). The spread of data points belonging to the same cluster is smaller than PCA and NMF projection.

Figure 3.6: Lower dimension projection of PCA $\mathbf{X}_{\text{IRIS}}\mathbf{V}^T$ for iris dataset, calculated by Equation (2.14). The spread of data points belonging to the same cluster is larger than linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE projection.

Figure 3.7: Lower dimension projection of NMF $\mathbf{X}_{\text{IRIS}}\mathbf{W}^T$ for iris dataset, calculated by Equation (2.22). The spread of data points belonging to the same cluster is larger than linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE projection.

3.2.3 Performance Comparison of Different Algorithms in USPS, CIFAR-10, and NORB Datasets

Table 3.4 shows the average testing accuracy, training time, and NMSE for features learned by linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, TAE, PCA, and NMF using ELM classifier. The number of user given parameters of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, TAE, PCA, and NMF for USPS, CIFAR-10, and NORB dataset using ELM classifier are:

- (1) Number of features.
- (2) Number of hidden neurons used by ELM classifier.
- (3) L_2 norm parameter of ELM classifier.

Table 3.1 shows the range of user given parameters used to find the best average testing

accuracy of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, TAE, PCA, and NMF.

The number of user given parameters of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, TAE, PCA, and NMF for USPS, CIFAR-10, and NORB dataset using linear SVM classifier are:

- (1) Number of features.
- (2) L_2 norm parameter of linear SVM parameter.

Table 3.2 shows the range of user given parameters used to find the best average testing accuracy of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, TAE, PCA, and NMF using linear SVM classifier.

The number of user given parameters of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, TAE, PCA, and NMF for USPS, CIFAR-10, and NORB dataset using Gaussian Kernel ELM classifier are:

- (1) Number of features.
- (2) Gaussian kernel parameter
- (3) L_2 norm parameter of Gaussian Kernel ELM parameter.

Table 3.3 shows the range of user given parameters used to find the best average testing accuracy of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, TAE, PCA, and NMF using Gaussian Kernel ELM classifier.

When ELM classifier is used, Table 3.4 shows that for USPS dataset non-linear SELM-AE lower dimensional projection outperforms TAE, PCA, and NMF lower dimensional projection by 0.44%, 0.3%, and 0.2% respectively. For CIFAR-10 dataset non-linear

DataSet	The number of features	The number of hidden neurons for ELM	The L_2 norm parameter
USPS	10, 20, ..., 240, 250	2000, 4000, 6000, 7000	$1e-8, 1e-7, \dots, 1e7, 1e8$
CIFAR-10	100, 200, ..., 2900, 3000	10000, 20000, 30000, 40000	$1e-8, 1e-7, \dots, 1e7, 1e8$
NORB	100, 200, ..., 1900, 2000	10000, 20000, 24000	$1e-8, 1e-7, \dots, 1e7, 1e8$

Table 3.1: The parameter ranges used to determine the best parameters of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, TAE, PCA, and NMF using ELM classifier.

DataSet	The number of features	The L_2 norm parameter
USPS	10, 20, ..., 240, 250	$1e-8, 1e-7, \dots, 1e7, 1e8$
CIFAR-10	100, 200, ..., 2900, 3000	$1e-8, 1e-7, \dots, 1e7, 1e8$
NORB	100, 200, ..., 1900, 2000	$1e-8, 1e-7, \dots, 1e7, 1e8$

Table 3.2: The parameter ranges used to determine the best parameters of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, TAE, PCA, and NMF using linear SVM classifier.

DataSet	The number of features	Gaussian Kernel parameter	The L_2 norm parameter
USPS	10, 20, ..., 240, 250	$1e-8, 1e-7, \dots, 1e7, 1e8$	$1e-8, 1e-7, \dots, 1e7, 1e8$
CIFAR-10	100, 200, ..., 2900, 3000	$1e-8, 1e-7, \dots, 1e7, 1e8$	$1e-8, 1e-7, \dots, 1e7, 1e8$
NORB	100, 200, ..., 1900, 2000	$1e-8, 1e-7, \dots, 1e7, 1e8$	$1e-8, 1e-7, \dots, 1e7, 1e8$

Table 3.3: The parameter ranges used to determine the best parameters of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, TAE, PCA, and NMF using Gaussian Kernel ELM classifier.

SELM-AE lower dimensional projection outperforms TAE, PCA, and NMF lower dimensional projection by 7.14%, 0.6%, and 2.17% respectively. For NORB dataset non-linear ELM-AE lower dimensional projection outperforms TAE, PCA, and NMF lower dimensional projection by 6%, 3.51%, and 5.22% respectively.

When ELM classifier is used, Table 3.4 shows that for USPS, CIFAR-10, and NORB datasets the training time of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE is lower than TAE and NMF by at least 200 times and eight times respectively. For USPS and CIFAR-10 datasets the training time of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE is equal to PCA. However, for NORB dataset the training time of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE is five times higher than PCA.

When ELM classifier is used, Table 3.4 shows that for USPS, CIFAR-10, and NORB datasets NMSE values of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE is in general equal to PCA and at least four times smaller than TAE. Hence linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE models input data more accurately than TAE. However, for USPS and CIFAR-10 datasets NMSE values of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE is equal to NMF and for NORB dataset at least 500 times smaller than NMF. Hence it seems that NMF converges to a local minimum for CIFAR-10 dataset.

Figure 3.8, Figure 3.9, Figure 3.10, Figure 3.11, Figure 3.12, and Figure 3.13 shows that for USPS, CIFAR-10, and NORB datasets linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE are sparser than PCA and not sparser than TAE and NMF. Furthermore, for USPS, CIFAR-10, and NORB datasets NMF is sparser than TAE. Hence, linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE to an extent learns both holistic-based and parts-based features. Furthermore, Figure 3.9, Figure 3.11, and Figure 3.13 show that linear SELM-AE is sparser than linear ELM-AE, non-linear SELM-AE, and non-linear ELM-AE for large features.

However, non-linear SELM-AE is not sparser than linear ELM-AE, non-linear ELM-AE, and linear SELM-AE as the sigmoid hidden layer activation function of non-linear SELM-AE reduces the sparsity.

Table 3.4 shows that TAE, linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE requires more features than PCA and NMF. This is because the objective of TAE, linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE is to maximize the mutual information [109] and learns parts-based and holistic-based features. In contrast, PCA only learns holistic-based features and NMF only learns parts-based features.

When linear SVM classifier is used, Table 3.5 shows that for USPS dataset performance of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, PCA and NMF lower dimensional projection is similar. For CIFAR-10 dataset, NMF lower dimensional projection classified by linear SVM outperforms linear-ELMAE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE and PCA lower dimensional projection by 17.23%, 17.05% 17.6%, 17.09% and 1.96% respectively. Furthermore, for CIFAR-10 dataset, PCA lower dimensional projection classified by linear SVM outperforms linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE lower dimensional projection by 15.23%, 15.05%, 15.6% and 15.09% respectively. For NORB dataset, PCA lower dimensional projection classified by linear SVM outperforms linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE and NMF lower dimensional projection by 0.85%, 1.17%, 0.84%, 1.31% and 10.75% respectively. Hence we could see that classifying low dimensional projections of linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE using linear SVM is not useful.

When linear SVM classifier is used, Table 3.5 shows that for USPS, CIFAR-10, and NORB datasets NMSE values of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE is in general equal to PCA, but at least 10 times lower than NMF. Time taken to generate NMF features is at least 2 times larger than linear

ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE and PCA for USPS and CIFAR-10 datasets. However, for NORB dataset time taken to generate linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE and PCA features is at least 4 seconds larger than NMF.

When Gaussian Kernel ELM classifier is used, Table 3.6 shows that for USPS dataset performance of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, PCA and NMF lower dimensional projection is similar. For CIFAR-10 dataset, non-linear ELM-AE lower dimensional projection classified by Gaussian Kernel ELM outperforms linear SELM-AE, non-linear SELM-AE, NMF and PCA lower dimensional projection by 1.05%, 0.69%, 1.28% and 4.16% respectively. For NORB dataset, non-linear ELM-AE lower dimensional projection classified by Gaussian Kernel ELM outperforms linear SELM-AE, non-linear SELM-AE, PCA and NMF lower dimensional projection by 0.21%, 0.18%, 1.11% and 5.89% respectively. Hence we could see that classifying low dimensional projections of linear-ELMAE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE using Gaussian Kernel ELM is useful.

When Gaussian Kernel ELM classifier is used, Table 3.6 shows that for USPS, CIFAR-10, and NORB datasets NMSE values of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE is in general equal to PCA, but at least 10 times lower than NMF. Time taken to generate NMF features is 100 times larger than linear-ELMAE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE and PCA for USPS dataset. Time taken to generate NMF features is 4 times larger than linear-ELMAE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE and PCA for CIFAR-10 dataset. Time taken to generate NMF features is at least one second larger than linear-ELMAE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE and PCA for NORB dataset.

Table 3.4, Table 3.5 and Table 3.6 shows that the training time of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE is in general equal. It is not possible to compare training times between linear ELM-AE, non-linear ELM-AE, linear

SELM-AE and non-linear SELM-AE, as windows 2012 operating system and matlab used in the experiments is not a real-time software. Hence, it is not possible to capture small execution times accurately. However, the training times can be used to compare between the following algorithms: 1) linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear ELM-AE vs. NMF; 2) linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear ELM-AE vs. TAE.

3.3 Summary

The objective of feature learning algorithms is to capture important information of input data and remove trivial information. To this end, this chapter introduces feature learning algorithms based on ELM and AE referred as ELM-AE and SELM-AE with different type of hidden layer activation functions. ELM-AE and SELM-AE with linear hidden layer activation function referred to as linear ELM-AE and non-linear ELM-AE respectively. ELM-AE and SELM-AE with sigmoid hidden layer activation function referred to as non-linear ELM-AE and non-linear SELM-AE respectively.

Linear ELM-AE and non-linear ELM-AE input weight are chosen to be orthogonal random, and only the output weight is learned. Similarly, linear SELM-AE and non-linear SELM-AE weight are chosen to be sparse random, and only the output weight is learned. The main motivation of using sparse random weight in linear SELM-AE and non-linear SELM-AE is to learn sparse features, as sparse features are more interesting than dense features.

Linear AE learns variance information of input data. In contrast, the proposed linear ELM-AE and linear SELM-AE learns the between-cluster scatter multiplied by orthogonal and sparse random weight respectively. Hence linear ELM-AE and linear SELM-AE reduces the Euclidean distance between data points belonging to the same cluster.

Linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE

DataSet	Algorithm	Features	Testing Accuracy (%)	Training Time (s)	NMSE
USPS	ELM-AE (linear)	40	98.71 (± 0.1)	0.084	0.03(± 0)
	ELM-AE (sigmoid)	40	98.7 (± 0.11)	0.065	0.03(± 0)
	SELM-AE (linear)	50	98.71 (± 0.12)	0.058	0.06(± 0)
	SELM-AE (sigmoid)	40	98.77 (± 0.12)	0.078	0.06(± 0)
	TAE [17]	180	98.33 (± 0.21)	31.71	0.20(± 0.01)
	PCA [53, 92]	20	98.46 (± 0.13)	0.054	0.027(± 0)
	NMF [78]	60	98.56 (± 0.16)	40.35	0.06(± 0)
	ELM-AE (linear)	300	53.43 (± 0.26)	6.3	0.01(± 0)
CIFAR-10	ELM-AE (sigmoid)	300	53.65 (± 0.28)	6.5	0.06(± 0)
	SELM-AE (linear)	300	53.48 (± 0.32)	6.2	0.01(± 0)
	SELM-AE (sigmoid)	300	53.92 (± 0.3)	6.3	0.01(± 0)
	TAE [17]	400	46.78 (± 0.18)	3484.1	0.23(± 0.001)
	PCA [53, 92]	100	53.32 (± 0.36)	9.1	0.07(± 0)
	NMF [78]	100	51.75 (± 0.34)	3361	0.063(± 0.048)
	ELM-AE (linear)	1000	91.58(± 0.3)	12.78	0.0001(± 0.02)
	ELM-AE (sigmoid)	1000	91.76 (± 0.26)	12.29	0.0003(± 0)
NORB	SELM-AE (linear)	1000	91.54 (± 0.37)	11.65	0.0001(± 0)
	SELM-AE (sigmoid)	1000	91.69 (± 0.26)	11.65	0.0006(± 0)
	TAE[17]	700	85.76 (± 0.33)	4149.5	0.26(± 0.003)
	PCA [53, 92]	100	88.25 (± 0.28)	2.63	0.003(± 0)
	NMF [78]	100	86.54 (± 0.69)	84.69	0.33(± 0.002)

Table 3.4: The testing accuracy, training time, and NMSE of USPS, CIFAR-10, and NORB dataset of ELM-AE (linear and non-linear), SELM-AE (linear and non-linear), TAE, PCA, and NMF using ELM classifier.

DataSet	Algorithm	Features	Testing Accuracy (%)	Training Time (s)	NMSE
USPS	ELM-AE (linear)	220	95.4 ($\pm$ 0.13)	0.39	0.002($\pm$ 0)
	ELM-AE (sigmoid)	210	95.34 ($\pm$ 0.16)	0.39	0($\pm$ 0)
	SELM-AE (linear)	240	95.32 ($\pm$ 0.15)	0.48	0($\pm$ 0)
	SELM-AE (sigmoid)	210	95.49 ($\pm$ 0.18)	0.39	0.004($\pm$ 0)
	PCA [53, 92]	180	95.27 ($\pm$ 0)	0.24	0.001($\pm$ 0)
	NMF [78]	150	95.48 ($\pm$ 0.25)	40.35	0.07($\pm$ 0.006)
CIFAR-10	ELM-AE (linear)	1800	22.98 ($\pm$ 0.88)	35.36	0($\pm$ 0)
	ELM-AE (sigmoid)	1800	23.16 ($\pm$ 1.78)	36.52	0($\pm$ 0)
	SELM-AE (linear)	1800	22.61 ($\pm$ 1.53)	30.48	0($\pm$ 0)
	SELM-AE (sigmoid)	1800	23.12 ($\pm$ 1.84)	31.74	0($\pm$ 0)
	PCA [53, 92]	1100	38.21 ($\pm$ 0)	38.24	0($\pm$ 0)
	NMF [78]	200	40.21 ($\pm$ 0.25)	80.84	0.18($\pm$ 0.043)
NORB	ELM-AE (linear)	2000	79.87($\pm$ 0.16)	23.21	0($\pm$ 0)
	ELM-AE (sigmoid)	1900	79.55 ($\pm$ 0.27)	22.47	0($\pm$ 0)
	SELM-AE (linear)	2000	79.88 ($\pm$ 0.16)	19.93	0($\pm$ 0)
	SELM-AE (sigmoid)	1800	79.41 ($\pm$ 0.15)	16.16	0.003($\pm$ 0)
	PCA [53, 92]	1600	80.72 ($\pm$ 0)	19.84	0.003($\pm$ 0)
	NMF [78]	300	69.97 ($\pm$ 2.86)	15.11	0.14($\pm$ 0.02)

Table 3.5: The testing accuracy, training time, and NMSE of USPS, CIFAR-10, and NORB dataset of ELM-AE (linear and non-linear), SELM-AE (linear and non-linear), TAE, PCA, and NMF using linear SVM classifier.

DataSet	Algorithm	Features	Testing Accuracy (%)	Training Time (s)	NMSE
USPS	ELM-AE (linear)	70	98.76 (± 0.13)	0.09	0.017(± 0.003)
	ELM-AE (sigmoid)	60	98.85 (± 0.08)	0.08	0.021(± 0.003)
	SELM-AE (linear)	40	98.77 (± 0.13)	0.1	0.03(± 0.004)
	SELM-AE (sigmoid)	70	98.79 (± 0.08)	0.12	0.02(± 0.002)
	PCA [53, 92]	80	98.75 (± 0)	0.2	0.005(± 0)
	NMF [78]	50	98.77 (± 0.09)	25.68	0.06(± 0.003)
CIFAR-10	ELM-AE (linear)	400	58.55 (± 0.21)	5.87	0.004(± 0)
	ELM-AE (sigmoid)	400	58.61 (± 0.24)	6.03	0.004(± 0)
	SELM-AE (linear)	500	57.56 (± 0.31)	8.14	0.003(± 0)
	SELM-AE (sigmoid)	500	57.92 (± 0.14)	8.37	0.005(± 0)
	PCA [53, 92]	400	57.33 (± 0)	11.91	0.002(± 0)
	NMF [78]	200	54.45 (± 0.98)	80.83	0.182(± 0.043)
NORB	ELM-AE (linear)	1200	93.65 (± 0.22)	11.05	0(± 0)
	ELM-AE (sigmoid)	1000	93.75 (± 0.12)	9.15	0(± 0)
	SELM-AE (linear)	1300	93.54 (± 0.15)	10.35	0(± 0)
	SELM-AE (sigmoid)	1000	93.57 (± 0.16)	8.25	0(± 0)
	PCA [53, 92]	300	92.64 (± 0)	4.6	0(± 0)
	NMF [78]	300	87.86 (± 0.36)	15.11	0.14(± 0.02)

Table 3.6: The testing accuracy, training time, and NMSE of USPS, CIFAR-10, and NORB dataset of ELM-AE (linear and non-linear), SELM-AE (linear and non-linear), TAE, PCA, and NMF using Gaussian Kernel ELM classifier.

Figure 3.8: USPS dataset sparsity results of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, PCA, NMF, and TAE. Sparsity of linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE is larger than PCA but smaller than NMF and TAE. Hence linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE is moderately sparse.

Figure 3.9: USPS dataset sparsity results of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE. Linear SELM-AE is sparser than linear ELM-AE, non-linear ELM-AE and non-linear SELM-AE for large number of features. While linear ELM-AE is sparser than non-linear ELM-AE and non-linear SELM-AE for large number of features.

Figure 3.10: CIFAR-10 dataset sparsity results of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, PCA, NMF, and TAE. Sparsity of linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE is larger than PCA but smaller than NMF and TAE. Hence linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE is moderately sparse.

Figure 3.11: CIFAR-10 dataset sparsity results of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE. Linear SELM-AE is sparser than linear ELM-AE, non-linear ELM-AE and non-linear SELM-AE for large number of features. While linear ELM-AE is sparser than non-linear ELM-AE and non-linear SELM-AE for large number of features.

Figure 3.12: NORB dataset sparsity results of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, non-linear SELM-AE, PCA, NMF, and TAE. Sparsity of linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE is larger than PCA but smaller than NMF and TAE. Hence linear ELM-AE, non-linear ELM-AE, linear SELM-AE and non-linear SELM-AE is moderately sparse.

Figure 3.13: NORB dataset sparsity results of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE. Linear SELM-AE is sparser than linear ELM-AE, non-linear ELM-AE and non-linear SELM-AE for large number of features. While linear ELM-AE is sparser than non-linear ELM-AE and non-linear SELM-AE for large number of features.

learns features robust to noise and is a denoiser. To further elaborate, the hidden layer operations of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE is explained: 1) input data is multiplied by orthogonal random weight (linear ELM-AE and non-linear ELM-AE) or sparse random weight (linear SELM-AE and non-linear SELM-AE) which retains the Euclidean distance information of input data and does not corrupt the input data; 2) orthogonal random bias (linear ELM-AE and non-linear ELM-AE) and sparse random bias (linear SELM-AE and non-linear SELM-AE) corrupts input data; 3) non-linear activation function of non-linear ELM-AE and non-linear SELM-AE adds non-linear noise to input data. It could be seen that the hidden layer randomly corrupts the input data and output layer maps the hidden layer output (corrupted input data) to input data. Hence linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE learn features which are robust to noise.

The efficacy of linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE is measured via discriminative capability, sparsity, training time, and NMSE on USPS handwritten digit dataset, CIFAR-10 object recognition dataset, NORB object recognition dataset. Experimental results show that: 1) When ELM or Gaussian Kernel ELM classifier is used, linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE lower dimensional projection is more discriminative than PCA, NMF, and TAE lower dimensional projection. However when linear SVM classifier is used PCA lower dimensional projection is more discriminative than linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE lower dimensional projection; 2) linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE learn features sparser than PCA but not sparse as NMF and TAE; 3) linear SELM-AE features are sparser than linear ELM-AE, non-linear ELM-AE, and non-linear SELM-AE; 4) linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE NMSE values are smaller than TAE and similar to PCA and NMF; 5) linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE training time is lower than TAE and NMF, and similar to PCA.

Chapter 4

Extreme Learning Machine based Multi-Layer Neural Network

Current multi-layer neural network learning algorithms consist of two steps: 1) an unsupervised learning algorithm such as RBM or TAE learns hidden layer parameters of the multi-layer neural network; 2) BP fine tunes parameters of the multi-layer neural network. This chapter investigates an ELM based multi-layer neural network referred to as Multi-Layer Extreme Learning Machine (ML-ELM) which consist of only one learning step. Non-linear ELM-AE learns the hidden layer parameters of ML-ELM and ML-ELM do not have a fine tuning step. Hence the training time of ML-ELM is significantly lower than SAE and DBN while the generalization capability is similar.

4.1 Non-linear Extreme Learning Machine Auto-Encoder

Chapter 3 showed the compressed representation or dimension reduction capability of non-linear ELM-AE. This chapter investigates equal dimension and sparse dimension representation of non-linear ELM-AE. Non-linear ELM-AE calculates the hidden layer

output for equal dimension and sparse representation as:

$$\begin{aligned}\mathbf{h}(\mathbf{x}) &= g(\mathbf{a}\mathbf{x} + \mathbf{b}) \\ \mathbf{A}\mathbf{A}^T &= \mathbf{I}, \mathbf{b}\mathbf{b}^T = 1\end{aligned}\tag{4.1}$$

Where $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_L]$ is the orthogonal random weight, $\mathbf{b} = [b_1, \dots, b_L]$ is the orthogonal random bias between the input nodes and hidden nodes and $g()$ is the hidden layer activation function. Objective function of non-linear ELM-AE for compressed and sparse representation is:

$$\underset{\beta_{\text{NAE}}}{\text{Minimize:}} \|\beta_{\text{NAE}}\|^2 + C\|\mathbf{H}\beta_{\text{NAE}} - \mathbf{X}\|^2\tag{4.2}$$

Equation (3.3) is similar to Equation (4.2) when $\sigma_1 = \sigma_2 = p = q = 2$ and $\mathbf{T} = \mathbf{X}$. Hence the solution to Equation (4.2) is:

$$\beta_{\text{NAE}} = \left(\frac{\mathbf{I}}{C} + \mathbf{H}^T \mathbf{H} \right)^{-1} \mathbf{H}^T \mathbf{X}\tag{4.3}$$

Where $\mathbf{H} = [\mathbf{h}_1, \dots, \mathbf{h}_N]$ is the hidden layer outputs of non-linear ELM-AE and $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ is input data and output data of non-linear ELM-AE.

Objective function of non-linear ELM-AE for equal dimension representation is:

$$\begin{aligned}\|\mathbf{H}\beta_{\text{NAE}} - \mathbf{X}\|^2 \\ \beta_{\text{NAE}}^T \beta_{\text{NAE}} = \mathbf{I}\end{aligned}\tag{4.4}$$

Peter et al. [103] showed that Equation (4.4) can be solved using Singular Value Decomposition (SVD) and is given by Algorithm 2.

$$\begin{aligned}\mathbf{M} &= \mathbf{X}^T \mathbf{H}; \\ [\mathbf{U}, \Sigma, \mathbf{V}] &= \text{svd}(\mathbf{M}); \\ \beta_{\text{NAE}} &= \mathbf{U}\mathbf{V}^T\end{aligned}$$

Algorithm 2: Algorithm to solve Equation (4.4).

4.2 Multi-Layer Extreme Learning Machine

ML-ELM consist of three hidden layers, and each hidden layer weight is learned by non-linear ELM-AE as follows:

- (1) The first hidden layer reduces the dimension of input data to remove unwanted information. Non-linear ELM-AE for compressed representation or dimension reduction learns the first hidden layer weight.
- (2) The second hidden layer has the same dimension as the first hidden layer and non-linear ELM-AE for equal dimension representation learns the second hidden layer weight. The objective of the second layer is to learn uncorrelated features by $\beta_{\text{NAE}}^T \beta_{\text{NAE}} = \mathbf{I}$ constraint. Second hidden layer activation function of ML-ELM is linear.
- (3) The third hidden layer has a larger dimension than the second hidden layer and non-linear ELM-AE for sparse representation learns the third hidden layer weight. Third hidden layer activation function of ML-ELM is non-linear.
- (4) Ridge regression learns output layer weight.

Figure 4.1 illustrates the learning algorithm of ML-ELM given by Algorithm 3.

ML-ELM inference is performed by Algorithm 4.

Figure 4.1: ML-ELM network structure. (a) Non-linear ELM-AE learns the 1-st layer weights β^1 of ML-ELM using input data $\mathbf{x}$. (b) Non-linear ELM-AE learns the i -th layer weights β^{i+1} of ML-ELM using i -th hidden layer output $\mathbf{h}^i$. (c) Regularized least square learns the ML-ELM output layer weights.

4.3 C Parameter of Non-Linear ELM-AE Selects Important Features

Effect of non-linear ELM-AE regularization parameter C is understood by applying Singular Value Decomposition (SVD) [39, 42, 46] to Equation (4.3):

$$\mathbf{H}\beta = \sum_{i=1}^L \mathbf{u}_i \frac{\mathbf{d}_i^2}{\mathbf{d}_i^2 + \frac{1}{C}} \mathbf{u}_i^T \mathbf{X} \quad (4.5)$$

where $\mathbf{u}$ are the eigenvectors of $\mathbf{H}\mathbf{H}^T$ and $\mathbf{d}$ are the singular values of $\mathbf{H}$. Essence feature learning is to retain important information of input data, while discarding unwanted information. According to Equation (4.5) small C value will remove most singular val-

ues, while large C value will retain most singular values. Hence C parameter acts as a feature selector. The original idea to add regularization parameter in ELM is to improve stability of the learning mechanism. Interestingly, ELM-AE regularization parameter helps feature learning.

4.4 Visual Investigation of Features Learned by Non-Linear ELM-AE

Visualizing features of a multi-layer neural network has shown to be an effective method to identify whether the learned features capture important information of input data [117]. Hence non-linear ELM-AE features are visualized to investigate whether non-linear ELM-AE features resembles input data. To this end, following data and non-linear ELM-AE networks are used: 1) MNIST handwritten digit dataset [74] which consist of ten classes representing digits '0' to '9'; 2) ten non-linear ELM-AE with a network architecture of 784 input neurons, 20 hidden neurons and 784 output hidden neurons (784-20-784). Each non-linear ELM-AE is trained with MNIST data belonging to the same class i.e. first non-linear ELM-AE is trained with '0' digit MNIST data. Figure 4.2 shows that non-linear ELM-AE learns features resembling input data and Algorithm 5 describes the process of generating Figure 4.2.

4.5 Visual Investigation of Features Learned by a Two Layer ML-ELM

It is important to verify visually whether ML-ELM second hidden layer weights (features) improve ML-ELM first hidden layer weights (features)¹. To this end, ML-ELM

¹As ML-ELM hidden layer weights are learned by non-Linear ELM-AE, ML-ELM weights can be referred as features.

Figure 4.2: Features learned by non-linear ELM-AE resembles input data. The first 20 digits of 0 in this figure is created by taking all the digits 0 in the MNIST dataset as input data for a non-linear ELM-AE with 20 hidden neurons and reshaping the output weights of non-linear ELM-AE to $28 \times 28 \times 20$. The next 20 digits of 1 in this figure is created by taking all the digits of 1 in the MNIST dataset as input data for a non-linear ELM-AE with 20 hidden neurons and reshaping the output weights of non-linear ELM-AE to $28 \times 28 \times 20$.

with a network architecture of 784 input neurons, 100 first hidden layer neurons, and 100 second hidden layer neurons is created (784-100-100) from MNIST handwritten recognition training data. By comparing Figure 4.3 and Figure 4.4, it could be seen that combining the output of second hidden layer weights (features) and first hidden layer weights (features) generates visually discernible features.

4.6 Performance Evaluation

MNIST dataset consists of 28×28 Gray scale images representing 10 classes of handwritten digits from '0' to '9'. There are 60000 training samples and 10000 testing samples. Experimental results are carried out in a laptop with core i7-3740-QM 2.7 Ghz processor and 32 GB RAM running MATLAB 2013a. Symbol "*" in front of Gaussian kernel, ELM experiment in Table 4.1, indicates this experiment was carried out in a high performance computer with dual Xeon E5-2650 2GHz and 256 GB RAM running MATLAB 2013a as Gaussian kernel ELM requires more memory than 32 GB. ML-ELM, random feature ELM, and Gaussian kernel ELM were executed ten times and the average testing accuracy, and average training time is reported. Table 4.1 shows the re-

sults of ML-ELM, random feature ELM, Gaussian kernel ELM, DBN, Deep Boltzmann Machine (DBM) [101], SAE, and Stacked Denoising Auto-Encoder (SDAE) [109]. ML-ELM is three layer multi-layer neural network consisting 784 input neurons, 700 first hidden layer neurons, 700 second hidden layer neurons, 15000 third hidden layer neurons, and 10 output layer neurons (784-700-700-15000-10). Non-linear ELM-AE with C parameter value 10^{-1} learns weights in the first hidden layer (784 – 700), non-linear ELM-AE learns second hidden layer weights (700 – 700) without using a C parameter, non-linear ELM-AE learns third hidden layer weights (700 – 15000) with C parameter value 10^3 and ridge regression learns output layer (15000 – 10) weights with C parameter 10^8 . ML-ELM achieves average testing accuracy of 99.03% which is similar to DBM, but higher than random feature ELM, Gaussian kernel ELM, DBN, SAE, and SDAE. Furthermore, ML-ELM training time is significantly lower than DBN, DBM, SAE, and SDAE.

4.7 Summary

DBN, DBM, SAE, and SDAE algorithms train multi-layer neural networks in two steps: 1) unsupervised learning algorithm to learn initial hidden layer weights; 2) BP fine tunes the whole network. In contrast, non-linear ELM-AE learns the three hidden layer parameters of ML-ELM and ML-ELM do not use BP to fine tune the network. Ridge regression learns the output weights of ML-ELM. Number of hidden neurons in the first hidden layer is smaller than the input neurons, and non-linear ELM-AE for compressed representation or dimension reduction learns the first hidden layer weight of ML-ELM. Hence first hidden layer of ML-ELM removes unwanted information of input data. Number of hidden neurons in the second hidden layer is equal to the first hidden layer, and non-linear ELM-AE for equal dimension representation learns the second hidden layer weight of ML-ELM. The second hidden layer of ML-ELM learns uncorrelated features as the objective function of non-linear ELM-AE for compressed representation is constrained as follows: $\beta_{\text{NAE}}^T \beta_{\text{NAE}} = \mathbf{I}$. Number of hidden neurons in

the third hidden layer of ML-ELM is larger than the second hidden layer, and non-linear ELM-AE for sparse representation learns the third hidden layer weights of ML-ELM. The third hidden layer of ML-ELM projects the data to high dimension. Ridge regression learns the output layer weight of ML-ELM, which maps the output of the third layer to targets.

The regularization parameter in non-linear ELM-AE for compressed and sparse representation is seen to perform feature selection.

```

InputData = normalize trainData (trainData is zero mean and standard
deviation 1);
i = 1;
while i ≤ numberOfLayers do
    previousHiddenLayerDimension = size(InputData,2);
    CurrentHiddenLayerDimension = Number of hidden neurons of layer i;
    InputWeight =
    rand(previousHiddenLayerDimension,CurrentHiddenLayerDimension)*2-
    1;

    InputWeight = Get a orthogonal basis of InputWeight (Use orth
    command in matlab);
    Bias = rand(CurrentHiddenLayerDimension,1)*2-1;
    Bias = Get orthogonal Basis of Bias ;
    H = sigmoid(InputData*InputWeight + Bias);
    if previousHiddenLayerDimension ≠ CurrentHiddenLayerDimension
        then
            | Use Equation (4.3) to find  $\beta_{\text{NAE}}$ ;
        else
            | Use Equation (4.4) to find  $\beta_{\text{NAE}}$ ;
        end
    stacked.w(i) =  $\beta_{\text{NAE}}$ . Save the calculated output weights;
    InputData = InputData* $\beta_{\text{NAE}}^T$  ;
    if previousHiddenLayerDimension ≠ CurrentHiddenLayerDimension
        then
            | InputData = sigmoid(InputData);
        else
            end
        i = i + 1;
    end
 $\beta_{\text{last}}$  = Use Equation 2.4 to find  $\beta_{\text{last}}$  where H = InputData;
stacked.w(numberOfLayers + 1) =  $\beta_{\text{last}}$ ;

```

Algorithm 3: ML-ELM training Algorithm.


```

InputData = normalize testData (testData is zero mean and standard
deviation 1);
i = 1;
while i ≤ numberOfLayers + 1 do
    previousHiddenLayerDimension = size(InputData,2);
    CurrentHiddenLayerDimension = Number of hidden neurons of layer i;
    InputData = InputData*stacked.w(i)T ;
    if previousHiddenLayerDimension ≠ CurrentHiddenLayerDimension
        and i < numberOfLayers + 1 then
            InputData = sigmoid(InputData);
        else
            end
        i = i + 1;
    end
predictedTargets = InputData;

```

Algorithm 4: ML-ELM Inference Algorithm.

```

InputData = normalize testData (testData is zero mean and standard
deviation 1);
i = 0;
while i ≤ numberOfLayers + 1 do
    InputData = MNIST data belonging to class i;
    Learn  $\beta_{\text{NAE}i}$  using Equation (4.1) and Equation (4.3);
    i = i + 1;
end
concatenateBeta = [ $\beta_{\text{NAE}0}; \dots; \beta_{\text{NAE}9}$ ];
display(concatenateBeta);

```

Algorithm 5: Generating Figure 4.2.

Algorithms	Network Architecture	Testing Accuracy% (dev%)	Training Time
ML-ELM	784-700-700-15000-10	99.03 (± 0.04)	444.655s
ELM (random features)	784-15000-10	97.39 (± 0.1)	545.95s
* ELM (Gaussian kernel)	-	98.75	790.96s
DBN	784-500-500-2000-10	98.87	20580s
DBM	784-500-1000-10	99.05	68246s
SAE[17]	784-500-500-2000-10	98.6	-
SDAE[109]	784-500-500-2000-10	98.72	-

Table 4.1: Performance comparison of ML-ELM with state-of-the-art Deep Networks.

Figure 4.3: 1-st layer weights β_{NAE} of ELM-AE network 784-100-784. All the data of MNIST data is taken as input data for a non-linear ELM-AE with 100 hidden neurons and the output weights of non-linear ELM-AE is reshaped to $28 \times 28 \times 100$.

Figure 4.4: 2-nd layer weights β_{NAE} of ELM-AE network 100-100-100 and 1-st layer weights β_{NAE} of ELM-AE 784-100-784. All the data of MNIST data is taken as input data for a non-linear ELM-AE with 100 hidden neurons. Then the output weights of the 1st non-linear ELM-AE is multiplied with input data and sigmoid activation function is applied to create a lower dimensional projection ($\text{sigmoid}(\mathbf{X}\beta_{NAE}^T)$). This lower dimension projection $\text{sigmoid}(\mathbf{X}\beta_{NAE}^T)$ is taken as input data for the second non-linear ELM-AE with 100 hidden neurons. The first non-linear ELM-AE output weights and second non-linear ELM-AE output weights are multiplied to create this figure.

Chapter 5

Extreme Learning Machine based Local Receptive Field Neural Network

CNN is an local receptive field network which uses BP to learn hidden layer parameters. BP algorithm is not suitable for learning multi-layer local receptive field network parameters as it is maimed by the vanishing or exploding gradient problem [18]. However as non-linear ELM-AE does not use BP algorithm to learn the output weights, it is not affected by the vanishing exploding gradient problem. This chapter introduces Local Receptive Field Extreme Learning Machine Auto-Encoder (LRF-ELMAE) which uses non-linear ELM-AE to learn hidden layer parameters and is an extention of ELM-LRF [58]. Experimental results show that LRF-ELMAE achieves state-of-the-art results and have a lower training time than BP based local receptive field networks.

5.1 Local Receptive Field Extreme Learning Machine Auto-Encoder

As shown in Figure 5.1, LRF-ELMAE is constructed from four type of layers:

- (1) The convolution layer captures specific parts of the image such as lines and curves. Non-linear ELM-AE learns the convolution layer parameters.
- (2) The pooling layer makes the network translation invariant. This layer does not have any learn-able parameters.
- (3) The fully connected layer combines the output of the previous layer. Non-linear ELM-AE learns the fully connected layer parameters.
- (4) The output layer maps previous layer output to targets. Ridge regression learns the output layer parameters.

LRF-ELMAE network is created by stacking convolution, pooling, fully connected and output layer based on the following conditions:

- (1) The first layer of LRF-ELMAE is always a convolution layer.
- (2) LRF-ELMAE adds pooling after a convolution layer.
- (3) LRF-ELMAE adds convolution layer after a convolution layer or pooling layer.
- (4) LRF-ELMAE adds fully connected layer before output layer or another fully connected layer.
- (5) Adding a fully connected layer is optional.
- (6) Smallest LRF-ELMAE network consist of a convolution layer, pooling layer, and output layer.

5.1.1 Learning Convolution Layer Parameters of Local Receptive Field Extreme Learning Machine Auto-Encoder

As convolution layer is a partially connect layer, and LRF-ELMAE learns convolution layer parameters using patches. Hence patch dataset $\mathbf{X}_{np}$ must be created from input data as follows:

Figure 5.1: LRF-ELMAE Structure. Non-linear ELM-AE learns the convolution layer weights. The pooling layer does not contain any learnable parameters and norm pooling operation is applied. Non-linear ELM-AE learns the fully connected layer weights and regularized least squares learns the output weights.

Figure 5.2: Generating the patch dataset $\mathbf{X}_p$. Randomly select patches of size $r \times r$ from the training data. Normalize these patches by removing the mean and dividing by the standard deviation to create the patch dataset.

- For a given patch size r , patch dataset is created by extracting image patches $r \times r$ from input data as shown in Figure 5.2. Random patches are selected from the patch dataset to reduce the size of patch dataset.
- Some patches in the patch dataset might contain the same pixel value, such as all values in patch is 1. LRF-ELMAE removes such not useful patches. LRF-ELMAE identifies such patches by variance, as the variance of a patch with same pixel value is 0.
- Each patch is normalized by reducing the mean value and dividing by the standard deviation of the patch $\mathbf{x}_{np_{i,j}} = (\mathbf{x}_{p_{i,j}} - \text{mean}(\mathbf{x}_{p_{i,j}})) / \text{std}(\mathbf{x}_{p_{i,j}})$. These patches are concatenated row wise to create the patch dataset:

$$\mathbf{X}_{np} = [\mathbf{x}_{np_{i,j}}; \mathbf{x}_{np_{i,j+1}}; \dots; \mathbf{x}_{np_{i,N_p}}; \mathbf{x}_{np_{i+1,j}}; \dots; \mathbf{x}_{np_{i+1,iN_p}}] \in \mathbf{R}^{N \times r^2}.$$

Non-linear ELM-AE learns the convolution layer weight of LRF-ELMAE by first calculating non-linear ELM-AE hidden layer output and then calculating non-linear ELM-AE output weights. If the number of hidden neurons L are smaller than smaller than patch size r^2 , then LRF-ELMAE calculates hidden layer output of non-linear ELM-AE using Equation (3.18) otherwise, LRF-ELMAE calculates hidden layer output of non-linear ELM-AE using Equation (4.1) where $\mathbf{X} = \mathbf{X}_{np}$. LRF-ELMAE calculates the output weights of non-linear ELM-AE using Equation (4.3) where $\mathbf{X} = \mathbf{X}_{np}$. The convolution

layer weight is the output weights of non-linear ELM-AE $\beta_{\text{NAE}} \in \mathbf{R}^{L \times r^2}$. Convolution layer weights β_{NAE} is reshaped to a three dimensional matrix $\beta_{\text{conv}} \in \mathbf{R}^{L \times r \times r}$. Where $\beta_{\text{conv}} = [\beta_{\text{conv}_1}, \dots, \beta_{\text{conv}_L}]$. LRF-ELMAE convolves the first convolution layer weight $\beta_{\text{conv}_1} \in \mathbf{R}^{r \times r}$ with input data to create the 1-st feature map $\mathbf{c}_1$, and LRF-ELMAE performs the convolution operation as:

$$c_{1i,j}(\mathbf{x}) = \sum_{m=1}^r \sum_{n=1}^r x_{i+m-1,j+n-1} \cdot \beta_{\text{conv}_1 m,n} \quad (5.1)$$

$$i, j = 1, \dots, (d - r + 1)$$

5.1.2 Pooling Layer of Local Receptive Field Extreme Learning Machine Auto-Encoder

LRF-ELMAE adds pooling layer after convolution layer. Hence for L feature maps there will be L pool maps. The pooling layer does not contain any learnable parameters and is shown to introduce translation invariance which is an important image processing criteria [25]. Commonly applied pooling operations are: 1) max [90]; 2) average [76]; 3) norm [102]. LRF-ELMAE uses norm pooling as norm pooling introduce frequency selectivity and translation invariance [102], and LRF-ELMAE calculates norm pooling operation as:

$$h_{p,q,k} = \sqrt{\sum_{i=p-e/2}^{p+e/2} \sum_{j=q-e/2}^{q+e/2} c_{i,j,k}^2} \quad (5.2)$$

Equation (5.2) shows that norm pooling consist of three operations: 1) create patches with size $e \times e$; 2) square each element in the patch; 3) add the squared elements in the patch and square root the value.

5.1.3 Learning Fully Connected Layer Weights of Local Receptive Field Extreme Learning Machine Auto-Encoder

Figure 5.1 shows that, LRF-ELMAE adds fully connected layer after pooling layer. Hence for L pool maps there will be L fully connected layers and LRF-ELMAE calculate fully connected layer weight $\beta_{\text{full}L}$ as follows:

- LRF-ELMAE reshapes pool map $\mathbf{H}_L$ of pool map L to a vector and normalizes to have zero mean and unit standard deviation as: $(\mathbf{H}_L - \text{mean}(\mathbf{H}_L))/\text{std}(\mathbf{H}_L)$
- LRF-ELMAE calculates fully connected layer output weights $\beta_{\text{full}L}$ from Equation (3.18), Equation (4.1), and Equation (4.3), where $\mathbf{X} = \mathbf{H}_L$.

5.1.4 Learning Output Weights of Local Receptive Field Extreme Learning Machine Auto-Encoders

The output layer maps the output of the fully connected layer to targets by learning output weights β_{output} as follows:

- LRF-ELMAE calculates output of fully connected layer $\mathbf{H}^f \in \mathbf{R}^{N \times L \cdot L^f}$ as: $\mathbf{H}^f = [\mathbf{H}_1 \cdot \beta_{\text{full}_1}^T; \dots; \mathbf{H}_k \cdot \beta_{\text{full}_k}^T]$. Where L^f is the number of hidden neurons of non-linear ELM-AE.
- LRF-ELMAE calculates output weight β_{output} by either Equation (2.4) or Equation (2.5). Which equation to use depends on the number of features $L \cdot L^f$ and number of training samples N , where $\mathbf{H} = \mathbf{H}^f$. If $N \geq L \cdot L^f$ then calculating the inverse $\mathbf{H}^T \mathbf{H}$ is computationally less expensive than $\mathbf{H} \mathbf{H}^T$, hence for this case LRF-ELMAE calculates output weight β_{output} using Equation (2.4). Otherwise LRF-ELMAE calculates output weight β_{output} using Equation (2.5).

LRF-ELMAE consist of four type of layers referred as convolution, pool, fully connected and output layer. In contrast, ELM-LRF consist of three type of layers referred as convolution, pool, and output layer. Furthermore, non-linear ELM-AE learns LRF-ELMAE convolution and fully connected layer weights.

Hierarchical Extreme Learning Machine (H-ELM) [123] consist of three type of layers referred as convolution, pool, and output layers. Non-linear ELM-AE learns H-ELM convolution layer weight, but preprocess the input using the whitening function. The whitening function is a time consuming process, and LRF-ELMAE do not apply the whitening function on input data.

CNN and variants such as Tiled CNN [71] consist of four layers referred as convolution, pool, fully connected, and output layers. BP learns the hidden parameters of CNN and Tiled CNN, which is time consuming.

5.2 Experiments

Experiments compare LRF-ELMAE with ELM-LRF, Convolutional Kernel Network [84], H-ELM [123], Tiled CNN [71], CNN [76], and Deep Belief Network (DBN) [49] on MNIST handwritten digit dataset [74] and NORB object recognition dataset. MNIST handwritten digit dataset consists 28×28 size images from 10 classes representing digits 0 to 9. There are 60000 training samples and 10000 testing samples in MNIST dataset [76]. NORB object recognition dataset consists stereo $2 \times 96 \times 96$ size images from 5 classes representing trucks, airplanes, humans, animals, and cars. There are 24300 training samples and 24300 testing samples in NORB dataset. For the experiments, NORB images were down sampled to $2 \times 32 \times 32$. The average testing and average training accuracy of 10 runs are reported in Table 5.3. Experiments were carried out in a high end computer with Xeon E5-2650 2Ghz processor and 256GB ram running Matlab 2013a. Parameter range used to determine the best parameters for LRF-ELMAE is given in Table 5.2. The network architecture and the parameters which generated best

average testing accuracy is given in Table 5.1.

LRF-ELMAE classifies MNIST dataset with 5 layers and these layers are: 1) first layer is a convolution layer; 2) second layer is also a convolution layer; 3) third layer is a norm pooling layer; 4) fourth layer is a fully connected layer; 5) fifth layer is the output layer. Table 5.3 shows that LRF-ELMAE achieves a testing accuracy of 99.55% which is a 0.15% improvement (accurately classifies 15 more testing samples) compared with ELM-LRF testing accuracy of 99.4% for MNIST dataset. However the training time of LRF-ELMAE 2.11 times larger than LRF-ELM. LRF-ELMAE accurately classifies 10 more testing samples than H-ELM, 47 more testing samples than CNN and 75 more testing samples than DBN. However, LRF-ELMAE accurately classifies 3 testing samples less than convolutional kernel network which is statistically insignificant. Furthermore, LRF-ELMAE training time is 22.36 times lower than convolutional kernel network.

LRF-ELMAE classifies NORB dataset with 4 layers and these layers are: 1) first layer is a convolution layer; 2) second layer is a norm pooling layer; 3) third layer is a fully connected layer; 4) fourth layer is the output layer. Table 5.3 shows that LRF-ELMAE achieves state-of-the-art results for NORB dataset. LRF-ELMAE classifies 185,486,1123, and 1098 more testing images accurately than ELM-LRF, Tiled CNN, CNN, and DBN respectively.

5.3 Summary

LRF-ELMAE is made of 4 type of layers: 1) convolution layer; 2) pooling layer; 3) fully connected layer; 4) output layer. Non-linear ELM-AE learns the hidden layer parameters of LRF-ELMAE, and ridge regression learn the output weights. In contrast, ELM-LRF uses random hidden layer parameters and ridge regression learns output layer weight, while BP learns CNN hidden layer and output layer parameters. Experimental results show that LRF-ELMAE achieves state-of-the-art results for NORB dataset, and results comparable to state-of-the-art for MNIST dataset. However, the training time of

Table 5.1: LRF-ELMAE network parameters for MNIST and NORB dataset.

Dataset	# of Layers	# feature maps for 1st conv layer	L_2 para for conv layer	1st conv layer patch size	# feature maps for 2nd conv layer	L_2 para for 2nd conv layer	2nd conv layer patch size	Norm pooling patch size	# hidden neurons for fully connected layer	L_2 para for fully connected layer	Output layer para
MNIST	5	10	10^3	4	9	10^3	4	4	450	10^{-1}	10^1
NORB	4	40	10^{-2}	3	-	-	-	7	1000	10^3	10^{-3}

Table 5.2: Parameter range used to determine the best parameters for LRF-ELMAE.

ELMAE range	L_2 parameter range	# feature range for layer	Conv layer size range	pool patch size range	Norm layer size range	# hidden neuron for fully layer	Output range	layer L_2 para
$[10^{-8}, 10^{-7}, \dots, 10^7, 10^8]$		$[10, 20, \dots, 40, 50]$	$[2, 3, \dots, 9, 15]$	$[2, 3, \dots, 9, 15]$	$[2, 3, \dots, 9, 15]$	$[50, 100, \dots, 950, 1000]$	$[10^{-8}, 10^{-7}, \dots, 10^7, 10^8]$	

Table 5.3: Testing Accuracy for MNIST and NORB dataset.

MNIST			
Algorithm	Testing Accuracy	Training Time (seconds)	speedup
LRF-ELMAE	99.55% (± 0.02)	3137	12.27
ELM-LRF	99.4%	1485	25.9
H-ELM	99.45%	14949	2.57
Convolutional Kernel Network	99.58%	70152	0.55
CNN	99.07%	14296	0.31
DBN	98.8%	38448	1
NORB			
LRF-ELMAE	98.02% (± 0.17)	1133	22.7
ELM-LRF	97.26% (± 0.02)	394.16	217.55
Tiled CNN	96.1%	15104.55	5.68
CNN	93.4%	53378.16	1.6
DBN	93.5%	85717.14	1

LRF-ELMAE is higher than ELM-LRF, but lower than other CNN, convolutional kernel network, Tiled CNN, and H-ELM.

Chapter 6

Extreme Learning Machine based Facial Point Detection

SDM and CFAN facial point detection algorithms find the location eyes, nose, mouth and face contour. However, SDM and CFAN facial point detection algorithms have the following shortcomings:

- (1) SDM and CFAN use fully connected neural networks to find facial points. Lecun et al. [76] shows that local receptive field networks outperform fully connected neural networks for computer vision problems.
- (2) SDM and CFAN use HOG values to find facial points. It is difficult to find facial points from HOG values, as HOG values do not contain distance information.

ELM based Facial Point Detection (ELM-FPD) algorithm resolves the shortcoming of SDM and CFAN and finds the facial points as follows: 1) Viola Jones face detection algorithm [110] finds the location of the face; 2) LRF-ELMAE estimates the facial points on the located faces using mean face shape; 3) Extreme Learning Machine Supervised Descent Method (ELM-SDM) improves the estimated facial points using lower dimensional response maps which contain distance information of facial points. Experimental

results show that ELM-FPD outperforms SDM and CFAN.

Facial point datasets contain a limited amount of training samples due to labeling costs. Hence ELM-FPD uses LRF-ELMAE to estimate the facial points, as it has been shown ELM based algorithms outperform other algorithms for datasets with a limited amount of training data [9, 58].

6.1 Extreme Learning Machine Supervised Descent Method

ELM-SDM improves the estimated facial points as follows:

$$\Delta \mathbf{t}_{iter}^T = g(\mathbf{a}, h(\mathbf{t}_{0iter}), \mathbf{b}) \beta_{elm-sdm_{iter}} \quad (6.1)$$

where $\mathbf{a}$ is the randomly chosen input weight matrix of ELM, $\mathbf{b}$ is the input bias of ELM and $g()$ is a non-linear activation function of ELM such as sigmoid function. $h()$ function performs the following steps:

- (1) Calculate the HOG values around the estimated facial points $\mathbf{t}_0$.
- (2) Generate response maps for the HOG values calculated in step 1.
- (3) Reduce the dimension of the response maps as response maps contain lot of zeros [7].
- (4) Concatenate the lower dimensional response maps.

Coarse to fine SDM has shown to improve the generalization performance by reducing unwanted information by increasing the resolution of the image (Refer to Section 2.4 for details). Similarly, ELM-SDM also uses coarse to fine strategy to reduce unwanted information. LRF-ELMAE finds the initial facial points $\mathbf{t}_0$.

Figure 6.1: Response Map Training of ELM-FPD. Get a image patch and generate HOG features for this image patch. For this HOG image patch generate the appropriate response map. If the image patch contains facial point in the middle the appropriate response map is shown in training sample 1. If the image patch contains facial point shifted to the right the appropriate response map is shown in training sample 2. Use the HOG features as input data and response map as output data to train a ELM.

6.1.1 Creating response maps

ELM-SDM algorithm generates response maps from HOG values. Hence this subsection describes response maps and response map generation process.

For a given image patch response map represents the location of a facial point using a 2D matrix which has the same size as the image patch, instead of using x, y coordinates. Hence each facial point can be represented by a response map. Response map shares the same spatial relationship as image patch (i.e. pixel at row 0 and column 0 is represented in the response map by row 0 and column 0) and each value in the response map represents the normalized distance to the facial point. Value one indicates that the facial point is located in this location i.e. if the response map has a value 1 at row 16 column 16 the facial point is located in the image patch at row 16 and column 16, if the response map has a value 1 at row 8 and column 16 the facial point in the image patch is located at row 8 column 16.

Traditionally response maps are created as a classification problem [11, 82, 113], by extracting small image patches from the image patch and labeling them as positive if

the small image patch contains the facial point or as negative. Disadvantages of this method are: 1) response map generation is time consuming as response map is created by evaluating multiple small image patches; 2) small image patch contains a limited amount of pixel information. Hence classifying small image patches as positive and negative is difficult as positive and negative small image patches might look alike. Hence response maps are created using a single large image patch as a regression problem. Figure 6.1 shows that ELM is trained using the following input data and output data: 1) input data to this regression problem is HOG values surrounding the facial point; 2) output data is the corresponding response maps to these HOG values.

The proposed ELM-FPD is described in Algorithm 6.

Input face image detected by Viola Jones face detector;
Output facial points $iter=0$;
 reshape the face image to 100×100 ;
while $iter < 5$ **do**
 if $iter == 0$ **then**
 LRF-ELMAE estimates facial points using mean face shape;
 else
 ELM-FPD estimates the facial points as: $\mathbf{t}_{0_{iter}} = \mathbf{t}_{0_{iter-1}} + \Delta \mathbf{t}_{iter-1}$;
 end
 ELM-FPD calculates response maps for the estimated facial points;
 ELM-AE reduces the dimension of each response map;
 ELM-FPD concatenates the lower dimensional response maps to a
 vector;
 ELM-SDM improves the estimated facial points using low dimensional
 response maps;
 Increase the resolution of the face by 100 pixels;
 $iter = iter + 1$;
end

Algorithm 6: ELM-FPD algorithm.

6.2 Results of Extreme Learning Machine base Facial Point Detection Algorithm

Experiments of ELM-FPD, SDM, and CFAN is performed on helen [72], Labeled face parts in the wild [14], Annotated faces in the wild [124], ibug [100], and multiple data [45]. Helen dataset contains 2000 training images and 330 testing images with illumination, occlusion, facial expressions and different head poses collected from *Flickr*. Labeled face parts in the wild dataset originally contained 1132 training images and 300 testing images collected from the Internet. However, as some links are unavailable, only 811 training images and 224 testing images was used. Ibug dataset contains 135 images and does not have a separate training and testing images. Multiple dataset is created from images collected in a laboratory environment under various illumination, head poses between $+90$ and -90 degrees with an interval of 15 degrees and facial expressions such as smile, disgust, surprise, and scream. This thesis uses multiple image data with head poses between $+45$ and -45 degrees. Multiple dataset contains 750,000, but only a fraction of these images contain face facial point information, and 3801 images were selected. Actual label annotated faces in the wild dataset is also created from images collected from *Flickr* similar to the helen dataset. However, the images of annotated faces in the wild contain larger variation in head pose, facial expressions, and appearance (sun glasses, make up, aging). Helen, labeled face parts in the wild, ibug, and multiple dataset contains only one face per image. However, annotated faces in the wild contains multiple faces per image, and there are 205 images with 468 faces.

SDM code available at <https://github.com/tntrung/impSDM> and CFAN code available at <http://vip1.ict.ac.cn/sites/default/files/basicpage/file> was used in the experiments. Training and testing code is available for SDM, but only the testing code is available for CFAN. As automatic face detection algorithms cannot detect all the faces in the datasets due to occlusion, illuminations, facial expressions, and head poses; face bounding box is calculated from the maximum and minimum x, y coordinate ground truth facial points.

The efficacy of ELM-FPD is measured via NMSE of the facial points and the number of face images less than 0.05 NMSE. NMSE is calculated by dividing the Mean Square Error (MSE) between predicted facial points and ground truth facial points by the distance between eye centers. ELM-FPD and SDM is trained with helen training data, labeled face parts in the wild training data, and annotated faces in the wild data. CFAN model is available on-line is trained with helen training and testing data, labeled face parts in the wild training data, and annotated faces in the wild data. ELM-FPD, SDM, and CFAN is tested with labeled face parts in the wild test data, ibug data, helen test data, and multiple data.

Figure 6.2, Figure 6.3, Figure 6.4, and Figure 6.5 shows the cumulative error plot of labeled face parts in the wild, ibug, helen, and multiple data. Figure 6.2, Figure 6.3, Figure 6.4, and Figure 6.5 shows that ELM-FPD performs better than SDM and CFAN. Table 6.1 shows that for labeled face parts in the wild, ELM-FPD performs 8.9% than CFAN and 26.3% than SDM. Table 6.1 shows that for ibug, ELM-FPD performs 8.8% than CFAN and 6.6% more than SDM. Table 6.1 shows that for helen, ELM-FPD performs 17.27% more than CFAN¹ and 24.54% more images than SDM. Table 6.1 shows that for multiple, ELM-FPD performs 8.89% more than CFAN and 2.9% more than SDM. Table 6.1 shows that the mean NMSE of ELM-FPD is lower than SDM and CFAN for labeled face parts in the wild, ibug, helen, and multiple data.

It is important to investigate whether ELM-FPD improves the NMSE at each iteration. To this end, Figure 6.6, Figure 6.7, Figure 6.8, and Figure 6.9 shows the ELM-FPD cumulative error plot of each iteration and each iteration reduces the NMSE.

Figure 6.10 shows the output of the proposed ELM-FPD algorithm for facial images from labeled face parts in the wild, ibug, and helen datasets with occlusion, beard, large head pose, and sunglasses. Based on the failure cases we see that ELM-FPD does not work well when the head is rolled significantly.

¹Result taken from [119] as CFAN code available on-line was trained on HELEN training and testing data, labeled face parts in the wild training data, and annotated faces in the wild data.

Figure 6.2: ELM-FPD, CFAN, and SDM results of labeled face parts in the wild dataset. ELM-FPD outperforms CFAN and SDM for labeled face parts in the wild dataset.

Figure 6.3: ELM-FPD, CFAN, and SDM results of ibug dataset. ELM-FPD outperforms CFAN and SDM for ibug dataset.

Figure 6.4: ELM-FPD, CFAN, and SDM results of HELEN dataset. ELM-SDM outperforms SDM for HELEN dataset.

Figure 6.5: ELM-FPD, CFAN, and SDM results of Multiple dataset. ELM-FPD outperforms CFAN and SDM for multiple dataset.

Figure 6.6: ELM-FPD cumulative error plot of each iteration for labeled face parts in the wild dataset. Each iteration improves the performance of ELM-FPD.

Figure 6.7: ELM-FPD cumulative error plot of each iteration for ibug dataset. Each iteration improves the performance of ELM-FPD.

Figure 6.8: ELM-FPD cumulative error plot of each iteration for HELEN dataset. Each iteration improves the performance of ELM-FPD.

Figure 6.9: ELM-FPD cumulative error plot of each iteration for multiple dataset. Each iteration improves the performance of ELM-FPD.

Figure 6.10: Each row starting from top represents ELM-FPD output of images from labeled face parts in the wild, ibug, and helen datasets respectively. The five columns from left represents face images with occlusion, beard, pose, smile, and sunglasses respectively. The right hand side column with the red outline represents failure cases of ELM-FPD.

Table 6.1: Mean NMSE values of ELM-FPD, CFAN, and SDM for labeled face parts in the wild, ibug, and HELEN datasets.

Dataset	Algorithm	Mean NMSE	# of faces with NMSE < 0.05
Labeled face parts in the wild	ELM-FPD	0.044	163
	CFAN	0.0536	123
	SDM	0.0558	104
ibug	ELM-FPD	0.1204	12
	CFAN	0.2292	0
	SDM	0.1439	3
HELEN	ELM-FPD	0.0467	245
	CFAN	N/A	198[119]
	SDM	0.0569	164
multiple	ELM-FPD	0.0676	421
	CFAN	0.1324	82
	SDM	0.0827	311

6.3 Summary

ELM-FPD is introduced to resolve the shortcomings of SDM and CFAN. ELM-FPD finds the facial point as follows: 1) automatic face recognition algorithm such as Viola Jones find face locations in the image; 2) LRF-ELMAE estimates the facial points based on the located faces using mean face shape; 3) ELM-SDM improves the estimated facial points using lower dimensional response maps.

Response maps are created by classifying small image patches as positive and negative as a classification problem. However, this method is time consuming as multiple small images must be evaluated and positive and negative small image patches might look alike due to limited amount of pixel information. Hence ELM-SDM generates response maps using a single large image patch as a multi-output regression problem.

Experimental results show that ELM-FPD outperforms CFAN and SDM for labeled face parts in the wild, ibug, helen, and multiple data.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis extends Extreme Learning Machine (ELM) for feature learning. To this end, Extreme Learning Machine Auto-Encoder (ELM-AE) and Sparse Extreme Learning Machine Auto-Encoder (SELM-AE) is introduced. In contrast to Tied Weight Auto-Encoders (TAE), ELM-AE and SELM-AE hidden layer parameters (encoder parameters) are chosen to be orthogonal random and sparse random respectively, which are not tuned. Hence ELM-AE and SELM-AE learns the output weight (decoder weight) analytically. This thesis investigates ELM-AE with linear hidden layer activation function referred to as linear ELM-AE, ELM-AE with sigmoid hidden layer activation function referred to as non-linear ELM-AE, SELM-AE with linear hidden layer activation function referred to as linear SELM-AE and SELM-AE with sigmoid hidden layer activation function referred to as non-linear SELM-AE.

Chapter 3 investigates linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE for dimension reduction (compressed representation). In theory, it could be shown that for dimension reduction linear ELM-AE and linear SELM-AE learns the between-cluster scatter, which reduces the Euclidean distance between data

points belonging to the same cluster. Linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE performance is determined via discriminative capability, sparsity, Normalized Mean Reconstruction Error (NMSE), and training time. The results show that linear ELM-AE, non-linear ELM-AE, linear SELM-AE, and non-linear SELM-AE performs better than PCA, NMF, and TAE. Hence grouping data points belonging to the same cluster is important.

Chapter 4 and Chapter 5 investigates fully connected ELM based multi-layer network referred to as Multi-Layer Extreme Learning Machine (ML-ELM) and ELM-AE based local receptive field network referred to as Local Receptive Field Extreme Learning Machine Auto-Encoder (LRF-ELMAE) respectively. Non-linear ELM-AE learns the hidden layer parameters of ML-ELM and LRF-ELMAE. Experimental results show that ML-ELM and LRF-ELMAE generalization capability is similar to other fully connected multi-layer neural networks and local receptive networks respectively, but the training time is significantly lower. Hence fully connected and local receptive field networks can be trained without using BP.

Chapter 6 investigates ELM based Facial Point Detection (ELM-FPD) algorithm. Facial Point Detection is a non-linear problem solved by Supervised Descent Method (SDM) and Coarse to Fine Auto-Encoder (CFAN). ELM-FPD resolves the shortcomings of SDM and CFAN and finds facial points as follows: 1) automatic face detection algorithm finds face locations; 2) LRF-ELMAE estimates the facial points for the located faces using mean face shape; 3) Extreme Learning Machine Supervised Descent Method (ELM-SDM) improves the estimated facial points using low dimensional response maps. Experimental results show that ELM-FPD outperforms SDM and CFAN. Hence using local receptive field networks and response maps are important in facial point detection.

7.2 Future work

This thesis investigates ELM for feature learning and for fully connected and local receptive field multi-layer neural networks. Some interesting research areas to be investigated are:

- It has been shown that in theory linear ELM-AE and linear SELM-AE learn the between-cluster for dimension reduction. Hence it is interesting to investigate theoretically whether features learned by non-linear ELM-AE and non-linear SELM-AE also represent the between-cluster for dimension reduction.
- This thesis shows that ELM-AE and SELM-AE are holistic-based feature learning algorithms for dimension reduction. Hence it is interesting to investigate ELM parts-based feature learning algorithms. As parts-based feature learning algorithms perform well on image data.
- Non-linear ELM-AE unsupervised learning algorithm learns ML-ELM and LRF-ELMAE hidden layer parameters. Hence ML-ELM and LRF-ELMAE require a large number of hidden layer parameters than other fully connected and local receptive field networks. Hence it is worthwhile to investigate an ELM based supervised feature learning algorithm which learns the hidden layer parameters of ML-ELM and LRF-ELMAE.

Publication list

Journal Papers:

1. **Liyanaarachchi Lekamalage Chamara Kasun**, Yang Yan, Guang-Bin Huang and Zhengyou Zhang, “Dimension Reduction with Extreme Learning Machine,” *IEEE Transactions on Image Processing*, vol. 28, no. 8, pp. 3906–3918, 2016.
2. Guang-Bin Huang, Bai Zuo, **Liyanaarachchi Lekamalage Chamara Kasun** and Chi Man Vong, “Local Receptive Fields Based Extreme Learning Machine”, *IEEE Computational Intelligence Magazine*, vol. 10, no. 2, pp. 18–29, 2015.
3. **Liyanaarachchi Lekamalage Chamara Kasun**, Hongming Zhou, Guang-Bin Huang and Chi Man Vong, “Representational Learning with Extreme Learning Machine for Big Data”, *IEEE Intelligent Systems*, vol. 28, no. 5, pp. 31–34, 2013.

Conference Papers:

1. Bai Zuo, **Liyanaarachchi Lekamalage Chamara Kasun** and Guang-Bin Huang, “Generic Object Recognition with Local Receptive Fields Based Extreme Learning Machine”, *INNS Conference on Big Data*, San Francisco, USA, Aug 8–10, 2015.
2. **Liyanaarachchi Lekamalage Chamara Kasun**, Tianchi Liu, Yan Yang, Zhiping Lin and Guang-Bin Huang, “Extreme Learning Machine based Clustering”, *The International Conference on Extreme Learning Machine*, Singapore, Dec 8–10, 2014.

Bibliography

- [1] OPELM and OPKNN in Long-Term Prediction of Time Series using Projected Input Data. *Neurocomputing*, 73(1012):1976 – 1986, 2010.
- [2] D. Achlioptas. Database-Friendly Random Projections. In *Proceedings of the 20th Symposium on Principles of Database Systems*, pages 274–281, 2001.
- [3] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A Learning Algorithm for Boltzmann Machines. *Cognitive Science*, 9(1):147–169, 1985.
- [4] A. Akusok, A. Lendasse, F. Corona, R. Nian, and Y. Miche. ELMVIS: A Non-linear Visualization Technique Using Random Permutations and ELMs. *IEEE Intelligent Systems*, 28(6):41–45, 2013.
- [5] A. Akusok, Y. Miche, K.-M. Björk, R. Nian, P. Lauren, and A. Lendasse. *ELMVIS+: Improved Nonlinear Visualization Technique Using Cosine Distance and Extreme Learning Machines*, pages 357–369. Springer International Publishing, 2016.
- [6] C. Andrieu, N. de Freitas, A. Doucet, and M. I. Jordan. An Introduction to MCMC for Machine Learning. *Machine Learning*, 50(1):5–43, 2003.
- [7] A. Asthana, S. Zafeiriou, S. Cheng, and M. Pantic. Robust Discriminative Response Map Fitting with Constrained Local Models. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3444–3451, June 2013.

- [8] Z. Bai, G. B. 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.
- [9] Z. Bai, L. L. C. Kasun, and G.-B. Huang. Generic Object Recognition with Local Receptive Fields Based Extreme Learning Machine. *Procedia Computer Science*, 53:391–399, 2015.
- [10] P. Baldi and K. Hornik. Neural Networks and Principal Component Analysis: Learning from Examples Without Local Minima. *Neural Networks*, 2(1):53–58, 1989.
- [11] T. Baltrusaitis, P. Robinson, and L.-P. Morency. Constrained Local Neural Fields for Robust Facial Landmark Detection in the Wild. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 354–361, 2013.
- [12] 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, 1998.
- [13] Y. Bazi, N. Alajlan, F. Melgani, H. Hichri, S. Malek, and R. R. Yager. Differential Evolution Extreme Learning Machine for the Classification of Hyperspectral Images. *IEEE Geoscience and Remote Sensing Letters*, 11(6):1066–1070, 2014.
- [14] P. N. Belhumeur, D. W. Jacobs, D. J. Kriegman, and N. Kumar. Localizing Parts of Faces Using a Consensus of Exemplars. In *IEEE Conference on Computer Vision and Pattern Recognition*, June 2011.
- [15] Y. Bengio. Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.
- [16] Y. Bengio, A. Courville, and P. Vincent. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

- [17] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy Layer-Wise Training of Deep Networks. In *Advances in Neural Information Processing Systems*, volume 19, pages 153–160. 2007.
- [18] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, Mar 1994.
- [19] D. Bitouk, N. Kumar, S. Dhillon, P. Belhumeur, and S. K. Nayar. Face Swapping: Automatically Replacing Faces in Photographs. *ACM Transactions on Graphics*, 27(3):1–8, 2008.
- [20] J. B. Butcher, D. Verstraeten, B. Schrauwen, C. R. Day, and P. W. Haycock. Reservoir Computing and Extreme Learning Machines for Non-linear Time-series Data Analysis. *Neural Networks*, 38:76–89, 2013.
- [21] F. Cao, B. Liu, and D. S. Park. Image Classification based on Effective Extreme Learning Machine. *Neurocomputing*, 102:90–97, 2013.
- [22] M. A. Carreira-Perpinan and G. E. Hinton. On Contrastive Divergence Learning. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, pages 33–40, 2005.
- [23] N.-B. Chang, M. Han, W. Yao, L.-C. Chen, and S. Xu. Change Detection of Land Use and Land Cover in an Urban Region with SPOT-5 Images and Partial Lanczos Extreme Learning Machine. *Journal of Applied Remote Sensing*, 4(1):043551–043551–15, 2010.
- [24] M. Chen, Z. Xu, K. Weinberger, and F. Sha. Marginalized Denoising Autoencoders for Domain Adaptation. In *Proceedings of the 29th International Conference on Machine Learning*, pages 767–774. July 2012.
- [25] A. Coates, H. Lee, and A. Ng. An Analysis of Single-Layer Networks in Unsupervised Feature Learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15, pages 215–223, 2011.

- [26] T. Cootes and C. Taylor. Active Shape Models Smart Snakes. In *Proceedings of the British Machine Vision Conference*, pages 266–275. 1992.
- [27] T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active Appearance Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):681–685, 2001.
- [28] T. F. Cootes and C. J. Taylor. Active Shape Model Search Using Local Grey-Level Models: A Quantitative Evaluation. In *Proceedings of the British Machine Vision Conference*, 1993.
- [29] T. F. Cootes, C. J. Taylor, and A. Lanitis. Active Shape Models: Evaluation of a Multi-Resolution Method for Improving Image Search. In *Proceedings of the British Machine Vision Conference*, pages 1–10, 1994.
- [30] C. Cortes and V. Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.
- [31] D. Cristinacce and T. Cootes. Automatic Feature Localisation with Constrained Local Models. *Pattern Recognition*, 41(10):3054–3067, 2008.
- [32] D. Cristinacce and T. F. Cootes. Feature Detection and Tracking with Constrained Local Models. In *Proceedings of the British Machine Vision Conference*, pages 95.1–95.10, 2006.
- [33] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 886–893, June 2005.
- [34] A. Demiriz, K. Bennett, and J. Shawe-Taylor. Linear Programming Boosting via Column Generation. *Machine Learning*, 46(1-3):225–254, 2002.
- [35] C. Ding and X. He. K-means Clustering via Principal Component Analysis. In *Proceedings of the 21st International Conference on Machine Learning*, pages 29–37, 2004.

- [36] C. Eckart and G. Young. The Approximation of One Matrix by Another of Lower Rank. *Psychometrika*, 1(3):211–218, 1936.
- [37] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Annals of Statistics*, 32:407–499, 2004.
- [38] R. A. Fisher. The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*, 7(7):179–188, 1936.
- [39] W. K. G. Golub. Calculating the Singular Values and Pseudo-Inverse of a Matrix. *Journal of the Society for Industrial and Applied Mathematics: Series B, Numerical Analysis*, 2(2):205–224, 1965.
- [40] S. Geman and D. Geman. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741, Nov 1984.
- [41] G. Golub and W. Kahan. Calculating the Singular Values and Pseudo-Inverse of a Matrix. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, 2(2):205–224, 1965.
- [42] G. H. Golub and C. Reinsch. Singular Value Decomposition and Least Squares Solutions. *Numerische Mathematik*, 14(5):403–420, 1970.
- [43] I. Goodfellow, H. Lee, Q. V. Le, A. Saxe, and A. Y. Ng. Measuring Invariances in Deep Networks. In *Advances in Neural Information Processing Systems*, volume 22, pages 646–654. 2009.
- [44] I. J. Goodfellow, D. Warde-farley, M. Mirza, A. Courville, and Y. Bengio. Maxout Networks. In *International Conference in Machine Learning*, 2013.
- [45] R. Gross, I. Matthews, J. Cohn, T. Kanade, and S. Baker. Multi-PIE. *Image and Vision Computing*, 28(5):807–813, may 2010.

- [46] M. R. Hestenes. Inversion of Matrices by Biorthogonalization and Related Results. *Journal of the Society for Industrial and Applied Mathematics*, 6(1):51–90, 1958.
- [47] G. E. Hinton. To Recognize Shapes, First Learn to Generate Images. *Progress in brain research*, 165:535–547, 2007.
- [48] G. E. Hinton, S. Osindero, and Y.-W. Teh. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527–1554, July 2006.
- [49] G. E. Hinton, S. Osindero, and Y.-W. Teh. A Fast Learning Algorithm for Deep Belief Nets. *Neural Comput.*, 18(7):1527–1554, 2006.
- [50] G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, 2006.
- [51] G. E. Hinton and R. S. Zemel. Autoencoders, Minimum Description Length and Helmholtz Free Energy. In *Neural Information Processing Systems*, pages 3–10, 1993.
- [52] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In Kremer and Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001.
- [53] H. Hotelling. Analysis of a Complex of Statistical Variables Into Principal Components. *Journal of Educational Psychology*, 24:417–441, 1933.
- [54] G. Huang, T. Liu, Y. Yang, Z. Lin, S. Song, and C. Wu. Discriminative clustering via extreme learning machine. *Neural Networks*, 70:1 – 8, 2015.
- [55] G. Huang, T. Liu, Y. Yang, Z. Lin, S. Song, and C. Wu. Discriminative Clustering via Extreme Learning Machine. *Neural Networks*, 70:1 – 8, 2015.

- [56] G. Huang, S. Song, J. Gupta, and C. Wu. Semi-Supervised and Unsupervised Extreme Learning Machines. *IEEE Transactions on Cybernetics*, 44(12):2405–2417, 2014.
- [57] G.-B. Huang. An Insight into Extreme Learning Machines: Random Neurons, Random Features and Kernels. *Cognitive Computation*, 6(3):1–15, 2014.
- [58] G. B. Huang, Z. Bai, L. L. C. Kasun, and C. M. Vong. Local Receptive Fields Based Extreme Learning Machine. *IEEE Computational Intelligence Magazine*, 10(2):18–29, 2015.
- [59] G.-B. Huang and L. Chen. Convex Incremental Extreme Learning Machine. *Neurocomputing*, 70:3056–3062, 2007.
- [60] G.-B. Huang and L. Chen. Enhanced Random Search Based Incremental Extreme Learning Machine. *Neurocomputing*, 71(16-18):3460–3468, 2008.
- [61] G.-B. Huang, L. Chen, and C.-K. Siew. Universal Approximation Using Incremental Constructive Feedforward Networks with Random Hidden Nodes. *IEEE Transactions on Neural Networks*, 17(4):879–892, 2006.
- [62] G.-B. Huang, L. Chen, and C.-K. Siew. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Transactions on Neural Networks*, 17(4):879–892, 2006.
- [63] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang. Extreme Learning Machine for Regression and Multiclass Classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 42(2):513–529, 2012.
- [64] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme Learning Machine: Theory and Applications. *Neurocomputing*, 70:489–501, 2006.
- [65] D. H. Hubel and T. N. Wiesel. Receptive Fields and Functional Architecture of Monkey Striate Cortex. *Journal of Physiology (London)*, 195:215–243, 1968.

- [66] J. J. Hunt, P. Dayan, and G. J. Goodhill. Sparse Coding Can Predict Primary Visual Cortex Receptive Field Changes Induced by Abnormal Visual Input. *Public Library of Science Computational Biology*, 9(5):1–17, 2013.
- [67] N. Hurley and S. Rickard. Comparing Measures of Sparsity. *IEEE Transactions on Information Theory*, 55(10):4723–4741, 2009.
- [68] P. Indyk and R. Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 604–613, 1998.
- [69] W. Johnson and J. Lindenstrauss. Extensions of Lipschitz Mappings into a Hilbert space. In *Conference in modern analysis and probability*, volume 26, pages 189–206. 1984.
- [70] H. Larochelle and Y. Bengio. Classification Using Discriminative Restricted Boltzmann Machines. In *Proceedings of the 25th International Conference on Machine Learning*, International Conference on Machine Learning, pages 536–543, 2008.
- [71] Q. V. Le, J. Ngiam, Z. Chen, D. J. hao Chia, P. W. Koh, and A. Y. Ng. Tiled convolutional neural networks. In *Neural Information Processing Systems*, pages 1279–1287, 2010.
- [72] V. Le, J. Brandt, Z. Lin, L. Bourdev, and T. S. Huang. Interactive facial feature localization. In *Proceedings of the European Conference on Computer Vision*, pages 679–692, 2012.
- [73] Y. Le Cun. *Modeles connexionnistes de l'apprentissage*. PhD thesis, 1987. Thse de doctorat dirige par Milgram, Maurice Sciences appliques Paris 6 1987.
- [74] Y. LeCun, L. Bottou, Y. Bengio, , and Haffner. Gradient-Based Learning Applied to Document Recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

- [75] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [76] Y. LeCun, F. J. Huang, and L. Bottou. Learning Methods for Generic Object Recognition with Invariance to Pose and Lighting. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 97–104, June 2004.
- [77] Y. LeCun, F. J. Huang, and L. Bottou. Learning Methods for Generic Object Recognition with Invariance to Pose and Lighting. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 97–104, June 2004.
- [78] D. D. Lee and H. S. Seung. Learning the Parts of Objects by Nonnegative Matrix Factorization. *Nature*, 401:788–791, 1999.
- [79] D. D. Lee and H. S. Seung. Algorithms for Non-negative Matrix Factorization. In *Neural Information Processing Systems*, pages 556–562, 2001.
- [80] H. Lee, C. Ekanadham, and A. Y. Ng. Sparse Deep Belief Net Model for Visual Area V2. In *Advances in Neural Information Processing Systems*, volume 20, pages 873–880. 2008.
- [81] M. Lin, Q. Chen, and S. Yan. Network In Network. *Computer Research Repository*, 2013.
- [82] X. Liu. Generic Face Alignment using Boosted Appearance Model. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2007.
- [83] D. G. Lowe. Object Recognition from Local Scale-Invariant Features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157, 1999.

- [84] J. Mairal, P. Koniusz, Z. Harchaoui, and C. Schmid. Convolutional Kernel Networks. *Computing Research Repository*, abs/1406.3332, 2014.
- [85] D. Martínez-Rego, O. Fontenla-Romero, B. Pérez-Sánchez, and A. Alonso-Betanzos. *Fault Prognosis of Mechanical Components Using On-Line Learning Neural Networks*, pages 60–66. Springer Berlin Heidelberg, 2010.
- [86] Y. Miche, A. Akusok, D. Veganzones, K.-M. Bjrk, E. Sverin, P. du Jardin, M. Termenon, and A. Lendasse. SOM-ELM Self-Organized Clustering using ELM. *Neurocomputing*, 165:238 – 254, 2015.
- [87] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, and A. Lendasse. OP-ELM: Optimally Pruned Extreme Learning Machine. *IEEE Transactions on Neural Networks*, 21(1):158–162, Jan 2010.
- [88] Y. Miche, M. van Heeswijk, P. Bas, O. Simula, and A. Lendasse. TROP-ELM: A double-regularized ELM using LARS and Tikhonov regularization. *Neurocomputing*, 74(16):2413 – 2421, 2011.
- [89] L. Mirsky. Symmetric Gauge Functions and Unitarily Invariant Norms. *Quarterly Journal of Mathematics Oxford Second Series*, 11:50–59, 1960.
- [90] J. Nagi, F. Ducatelle, G. A. D. Caro, D. Cirean, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella. Max-Pooling Convolutional Neural Networks for Vision-based Hand Gesture Recognition. In *IEEE International Conference on Signal and Image Processing Applications*, pages 342–347, Nov 2011.
- [91] M. Pal, A. E. Maxwell, and T. A. Warner. Kernel-based Extreme learning Machine for Remote-Sensing Image Classification. *Remote Sensing Letters*, 4(9):853–862, 2013.
- [92] K. Pearson. On Lines and Planes of Closest Fit to Systems of Points in Space. *Philosophical Magazine*, 2(6):559–572, 1901.

- [93] Y. Peng, W.-L. Zheng, and B.-L. Lu. An Unsupervised Discriminative Extreme Learning Machine and Its Applications to Data Clustering. *Neurocomputing*, 174(PA):250–264, Jan. 2016.
- [94] M. Ranzato, C. S. Poultney, S. Chopra, and Y. LeCun. Efficient Learning of Sparse Representations with an Energy-Based Model. In *Neural Information Processing Systems*, pages 1137–1144, 2006.
- [95] M. Ranzato, C. S. Poultney, S. Chopra, and Y. LeCun. Efficient Learning of Sparse Representations with an Energy-Based Model. In *Neural Information Processing Systems*, pages 1137–1144, 2006.
- [96] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive Auto-Encoders: Explicit Invariance During Feature Extraction. In *International Conference on Machine Learning*, pages 833–840, 2011.
- [97] O. Rudovic and M. Pantic. Shape-constrained Gaussian process regression for facial-point-based head-pose normalization. In *International Conference on Computer Vision*, pages 1495–1502, 2011.
- [98] D. Rumelhart, G. Hintont, and R. Williams. Learning Representations by Back-Propagating Errors. *Nature*, 323(6088):533–536, 1986.
- [99] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Internal Representations by Error Propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol: Foundations*, pages 318–362. MIT Press, Cambridge, MA, 1986.
- [100] C. Sagonas, G. Tzimiropoulos, S. Zafeiriou, and M. Pantic. A Semi-automatic Methodology for Facial Landmark Annotation. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 896–903, June 2013.
- [101] R. Salakhutdinov and G. Hinton. Deep Boltzmann machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 5, pages 448–455, 2009.

- [102] A. M. Saxe, P. W. Koh, Z. Chen, M. Bhand, B. Suresh, and A. Y. Ng. On Random Weights and Unsupervised Feature Learning. In *International Conference on Machine Learning*, pages 1089–1096. Omnipress, 2011.
- [103] P. Schnemann. A Generalized Solution of the Prthogonal Procrustes Problem. *Psychometrika*, 31(1):1–10, 1966.
- [104] Y. Sun, X. Wang, and X. Tang. Deep Convolutional Network Cascade for Facial Point Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3476–3483, 2013.
- [105] J. Sung and D. Kim. Pose-Robust Facial Expression Recognition Using View-Based 2D + 3D AAM. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 38(4):852–866, July 2008.
- [106] J. A. K. Suykens and J. Vandewalle. Least Squares Support Vector Machine Classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.
- [107] M. D. Tissera and M. D. McDonnell. Deep extreme learning machines: supervised autoencoding architecture for classification. *Neurocomputing*, 174, Part A:42 – 49, 2016.
- [108] M. D. Tissera and M. D. McDonnell. Enhancing deep extreme learning machines by error backpropagation. In *2016 International Joint Conference on Neural Networks*, pages 735–739, July 2016.
- [109] 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. *Journal of Machine Learning Research*, 11:3371–3408, 2010.
- [110] P. Viola and M. Jones. Rapid Object Detection Using a Boosted Cascade of Simple Features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 511–518, 2001.

- [111] N. Wang, D. Tao, X. Gao, X. Li, and J. Li. A Comprehensive Survey to Face Hallucination. *International Journal of Computer Vision*, 106(1):9–30, 2014.
- [112] X. Wang and M. Han. Multivariate Time Series Prediction based on Multiple Kernel Extreme Learning Machine. In *International Joint Conference on Neural Networks*, pages 198–201, July 2014.
- [113] Y. Wang, S. Lucey, and J. Cohn. Enforcing Convexity for Improved Alignment with Constrained Local Models. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2008.
- [114] T. Weise, S. Bouaziz, H. Li, and M. Pauly. Realtime Performance-based Facial Animation. *ACM Transactions on Graphics*, 30(4):1–10, 2011.
- [115] X. Xiong and F. de la Torre. Supervised Descent Method and Its Applications to Face Alignment. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 532–539, June 2013.
- [116] H. Yang, W. Xu, J. Zhao, D. Wang, and Z. Dong. Predicting the Probability of Ice Storm Damages to Electricity Transmission Facilities based on ELM and Copula Function. *Neurocomputing*, 74(16):2573–2581, 2011.
- [117] M. D. Zeiler and R. Fergus. Visualizing and Understanding Convolutional Networks. In *13th European Conference on Computer Vision*, pages 818–833. Springer International Publishing, 2014.
- [118] C. Zhang, S. Xia, B. Liu, and L. Zhang. Extreme Maximum Margin Clustering. *IEICE Transactions on Information and Systems*, E96.D(8):1745–1753, 2013.
- [119] J. Zhang, S. Shan, M. Kan, and X. Chen. Coarse-to-Fine Auto-Encoder Networks (CFAN) for Real-Time Face Alignment. In *European Conference on Computer Vision*, volume 8690, pages 1–16. 2014.
- [120] R. Zhang, Y. Lan, G.-B. Huang, and Z.-B. 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.

- [121] W. H.-L. Zhang Xian. Incremental Regularized Extreme Learning Machine based on Cholesky Factorization and its Application to Time Series Prediction. *Acta Physica Sinica*, 60(11):110201, 2011.
- [122] H. Zhou, G. B. Huang, Z. Lin, H. Wang, and Y. C. Soh. Stacked Extreme Learning Machines. *IEEE Transactions on Cybernetics*, 45(9):2013–2025, Sept 2015.
- [123] W. Zhu, J. Miao, L. Qing, and G. B. Huang. Hierarchical Extreme Learning Machine for Unsupervised Representation Learning. In *International Joint Conference on Neural Networks*, pages 1–8, July 2015.
- [124] X. Zhu and D. Ramanan. Face Detection, Pose Estimation, and Landmark Localization in the Wild. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2879–2886, 2012.