

NEW ALGORITHMS FOR DEEP LEARNING MACHINES

BY

ISSAM HADJ LARADJI

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

MAY 2014

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA


DEANSHIP OF GRADUATE STUDIES

This thesis, written by **ISSAM HADJ LARADJI** under the direction of his thesis adviser and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN DEPARTMENT OF INFORMATION AND COMPUTER SCIENCE**.

Thesis Committee



Dr. Lahouari Ghouti (Adviser)




Dr. Sabri Mahmoud (Member)



Dr. Shokri Selim (Member)



Dr. Adel F. Ahmed
Department Chairman



Dr. Safam A. Zummo
Dean of Graduate Studies

Date

3/7/14



©Issam Hadj Laradji
2014

Dedicated to my parents, and my brothers

ACKNOWLEDGMENTS

I would like to thank Dr. Lahouari Ghouti, my supervisor, whose help and motivation encouraged me to accomplish the most rewarding goal that is this thesis. I would also like to thank my thesis committee members, Prof. Sabri Mahmoud and Prof. Shokri Selim, whose feedback and comments were vital to the completion of this thesis. Furthermore, I express my gratitude for my professors with whom I had the honour to be their student - among them are Dr. Adam Salahadin, Dr. Wasfi Al-khatib, Dr. Musab Alturki, Dr. Mohammad Alshayeb, and Dr. Sultan Almuhammadi. Finally, I would like to thank my family who supported and inspired me to strive hard for my ambitious goals, allowing me to enjoy the memorable journey of working through developing this thesis.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	iii
LIST OF TABLES	vii
LIST OF FIGURES	ix
ABSTRACT (ENGLISH)	xi
ABSTRACT (ARABIC)	xiii
CHAPTER 1	
INTRODUCTION	1
1.1 Problem Statement	3
1.2 Thesis Contributions	3
1.3 Thesis Breakdown	3
CHAPTER 2	
LITERATURE REVIEW	5
2.1 Notations	5
2.2 Overview	5
2.3 Overview of ANN Algorithms	12
2.3.1 Logistic Regression	13
2.3.2 Multi-Layer Perceptron	15
2.3.3 Recurrent ANNs	19
2.3.4 Sparse autoencoders	20

2.3.5	Deep Belief Networks	23
2.3.6	Current Performance of Deep Learning Networks	26
2.4	Overview of Extreme Learning Machine Algorithms	32
2.4.1	Mathematical Notation	32
2.5	Extreme Learning Machines	33
2.5.1	Weighted ELMs	35
2.5.2	Sequential ELMs	37
2.6	Activation Functions	39
2.7	Kernel Functions	41
2.7.1	Model Selection and Criteria	42
CHAPTER 3		
RECURRENT EXTREME LEARNING MACHINES		46
3.1	Introduction	46
3.1.1	Recurrent-Hidden Extreme Learning Machines	47
3.1.2	Recurrent-Output Extreme Learning Machines	49
3.2	Experimental Results	53
3.2.1	Experimental Setup	53
3.2.2	Experimental Design	53
3.2.3	Experimental Results	55
CHAPTER 4		
EXTREME LEARNING MACHINE BASED AUTOENCODERS		62
4.1	Introduction	62
4.2	Experimental Results	69
4.2.1	Experimental Setup	69
4.2.2	Experimental Design	69
4.2.3	Experimental Results	70
4.2.4	ELM-AE Features	71
4.2.5	ELM-AE feature weights	73
4.2.6	ELM-Based DBNs vs. DBNs	75

4.2.7	Performance results in the literature	77
CHAPTER 5		
CONVOLUTIONAL EXTREME LEARNING MACHINES		78
5.1	Introduction	78
5.2	Experimental Results	85
5.2.1	Experimental Setup	85
5.2.2	Experimental Design	85
5.2.3	Experimental Results	86
CHAPTER 6		
CONCLUSION		90
REFERENCES		93
VITAE		104

LIST OF TABLES

2.1	Notations.	6
3.1	Dataset statistics. S and F are the number of samples and features in the dataset, respectively.	54
3.2	Comparison between algorithms using the Mean-Absolute Error performance metric on the yahoo dataset.	55
3.3	Comparison between algorithms using the Mean-Absolute Error performance metric against the 1D Wind dataset.	56
3.4	Comparison between algorithms using the Mean-Square Error performance metric on the Short-movement Stock Prices dataset.	56
4.1	Dataset statistics.	69
4.2	Algorithms' settings.	70
4.3	Performance on the MNIST dataset for 500, 1K, and 2K samples.	71
4.4	Performance on the MNIST dataset for 4k, 8k samples.	71
4.5	Performance on the AHDBase dataset for 500, 1K, and 2K samples.	72
4.6	Performance on the AHDBase dataset for 4k, 8k samples.	72
4.7	Performance on the MAHDBase dataset for 500, 1K, and 2K samples.	72
4.8	Performance on the MAHDBase dataset for 4k, 8k samples.	73
4.9	Comparison between algorithms with respect to training time.	75
4.10	Comparison between DBN and ELM-DBN.	76
4.11	Overview of algorithms in the literature.	77
5.1	Dataset statistics.	85

5.2	Comparison between algorithms using the accuracy performance metric.	86
5.3	Performance on the MNIST dataset for 500, 1K, and 2K samples.	87
5.4	Performance on the MNIST dataset for 4K and 8K.	87
5.5	Performance on the MAHD dataset for 500, 1K, and 2K samples.	87
5.6	Performance on the MAHD dataset for 4K and 8K.	87
5.7	Performance on the AHDBase dataset for 500, 1K, and 2K samples.	88
5.8	Performance on the AHDBase dataset for 4K and 8K.	88

LIST OF FIGURES

1.1	ANN with one hidden layer.	1
2.1	XOR problem formulation.	7
2.2	The green line represents the decision boundary.	8
2.3	The trained decision function separating the two classes.	8
2.4	4-Point XOR example.	9
2.5	A multi-curve solution of the XOR problem.	10
2.6	Weight initialization effect.	10
2.7	DBN pre-trained by RBM.	11
2.8	Levels of abstraction.	12
2.9	MLP with one hidden layer.	15
2.10	Elman Neural network.	19
2.11	Sparse autoencoder network.	21
2.12	Relationship between KLD and \hat{p} when $p = 0.2$	22
2.13	Network with two hidden layers.	24
2.14	First hidden layer training.	24
2.15	Second hidden layer training.	25
2.16	Output layer training.	25
2.17	Plot of the logistic function	40
2.18	Plot of hyperbolic tan.	41
3.1	Recurrent-hidden extreme learning machines.	48
3.2	Training Recurrent Hidden ELM.	49
3.3	Recurrent-output extreme learning machines.	50

3.4	Training Recurrent Output Neural Network	51
3.5	Visualizing Spatio-Temporal learning.	53
3.6	Illustrates the values of the chaos datasets.	55
3.7	RH-ELM vs. RO-ELM vs. ELM on the 4-D japanese Wind dataset.	57
3.8	RH-ELM vs. RO-ELM vs. ELM on the Chaos dataset A.	58
3.9	RH-ELM vs. RO-ELM vs. ELM on the Chaos dataset B.	58
3.10	RH-ELM vs. RO-ELM vs. ELM on the Chaos dataset C.	59
3.11	RH-ELM vs. RO-ELM vs. ELM on the Chaos dataset D.	59
3.12	RH-ELM vs. RO-ELM vs. ELM on the Chaos dataset E.	60
3.13	RH-ELM vs. RO-ELM vs. ELM on the Chaos dataset F.	60
4.1	Sparse autoencoder network.	64
4.2	Training DBN using ELM-AE.	67
4.3	Samples of the three digit datasets.	70
4.4	Displays feature weights β trained by ELM-AE.	74
4.5	Displays feature weights W_1 trained by SAE.	74
5.1	Convolution of a kernel with a 2D image subset.	79
5.2	Image reconstruction using ELM-CA.	80
5.3	Dot product representation.	82
5.4	Returns the hidden filters from training ELM-CA as features.	83
5.5	A CNN of one convolutional layer followed by a fully connected layer.	84
5.6	Trains an ELM-CNN consisting of one convolutional layer followed by a fully connected layer.	84

THESIS ABSTRACT

NAME: Issam Hadj Laradji
TITLE OF STUDY: New Algorithms for Deep Learning Machines
MAJOR FIELD: Department of Information and Computer Science
DATE OF DEGREE: May 2014

The objective of the thesis is two-fold. First, it provides an outline of the fundamental concepts revolving around artificial neural networks (ANNs) and extreme learning machines (ELMs). Second, built on these concepts, it presents three new deep learning algorithms. The first algorithm uses the least squares technique to learn spatio-temporal patterns. It addresses problems including the prediction of stock market values, wind speed and chaotic patterns. The algorithm uses information provided by previous time-steps to improve its decision function. Two variants are proposed for of this algorithm: 1) recurrent-hidden ELM and; 2) recurrent-output ELM. Computer simulations showed that, on average, the algorithm outperforms traditional ELMs on six different datasets. Called ELM-based autoencoder (ELM-AE), the second algorithm is a feature extractor based on the

least squares technique too. To extract structural hidden features from the input data, the least squares algorithm is preferred over the backpropagation algorithm for its learning speed. Further, the second algorithm can be used to train larger neural networks consisting of two or more hidden layers. Using the optical character recognition (OCR) problem, simulation results revealed that the extracted digit strokes using the ELM-AE and sparse autoencoders are similar. However, it is worth noting that the ELM-AE algorithm has a faster learning curve. Moreover, large networks, trained by the ELM-AE algorithm, perform as well as deep belief networks trained with the backpropagation algorithm using the same number of hidden layers. Like its predecessors, the third algorithm, called the ELM-based convolutional autoencoders (ELM-CA), uses the least squares algorithm to train a convolutional layer to reconstruct the input images. Such design allows constructing large convolutional neural networks (CNNs) that train very quickly as opposed to those based on backpropagation. In fact, the latter networks can take weeks to train. Reported results showed that CNNs trained using the ELM-based CA algorithm achieve competitive results that greatly outperform support vector machines (SVM) and standard ELM algorithms. The contributions, made in this thesis, aim to start a new trend of deep learning algorithms that are quick to train and efficient as well.

ملخص الرسالة

الاسم: محام الحاج لعراجي

عنوان الرسالة: خوارزميات جديدة لأدوات التعلم العميق

التخصص: علوم الحاسب الآلي

تاريخ الدرجة العلمية: مايو 2014

تمهذه هذه الرسالة إلى تحقيق هدفين وهما: أولاً تبين مفصل للمفاهيم الأساسية للشبكات العصبية الصناعية وأدوات التعلم السريعة. ثانياً، بناءاً على هذه المفاهيم المقدمة يتم تصميم ثلاثة خوارزميات جديدة للتعلم العميق. تستخدم الخوارزمية الأولى بتقنية المربعات الصغرى لمعرفة الأنماط المكانية والزمانية. وتطبيق لهذه الخوارزمية، تتم معالجة مشاكل مثل تنبؤ قيم أسهم الأوراق المالية، وسرعة الرياح، وتعلم أنماط الفوضى. وتعتمد هذه الخوارزمية على المعلومات المتمثلة في الخطوات السابقة لتحسين وظيفة قرارها. وتم اقتراح نسختين من هذه الخوارزميات وهما آلات التعلم السريعة ذات طبقة خفية متكررة (recurrent-hidden ELM)، وآلات التعلم السريعة ذات مخرجات متكررة (recurrent-output ELM). وقد أظهرت محاكاة المقارنة أنه، في المتوسط، تتفوق الخوارزمية المقترحة الأولى بنسختيها على آلات التعلم السريعة التقليدية في حالة ستة مجموعات بيانات مستخدمة. أما الخوارزمية المقترحة الثانية والمعروفة باسم آلات التعلم السريعة للتشفير الذاتي (ELM-AE) وهي طريقة استخراج الميزات على أساس المربعات الأقل. وبما أنها لا تستخدم تقنية النشر العكسي (backpropagation) لتدريب الأوزان هي سريعة للغاية بخلاف النشر العكسي التي تعاني من بطء التعلم. وبذلك تتمكن هذه الخوارزمية من استخراج السمات الهيكلية الخفية عبر بيانات الإدخال. علاوة على ذلك، يمكن استخدامها لتدريب شبكات عصبية كبيرة الحجم والمتكونة من طبقتين خفيتين أو أكثر. ومن خلال محاكاة التعرف على الحروف، تبين تشابه الميزات المستخرجة عبر خوارزميتي آلات التعلم السريعة للتشفير الذاتي وآلات التشفير الذاتي القليلة التواصل (sparse autoencoders) على الرغم من أن سرعة تدريب آلات التعلم السريعة للتشفير الذاتي هو أقل بكثير. كما أظهرت الدراسة أيضاً أن الشبكات العصبية كبيرة الحجم المدربة باستخدام آلات التعلم السريعة للتشفير الذاتي تتسم بدقة مشابهة للشبكات المدربة عبر شبكات الإيمان العميق (deep belief networks) وتقنية النشر العكسي حتى عندما يستخدم نفس العدد من الطبقات الخفية. وفيما يخص الخوارزمية المقترحة الثالثة والمعروفة باسم آلات التعلم السريعة للتشفير الذاتي باعتماد التلافيف (ELM-based convolutional autoencoders) والتي تستخدم المربعات الصغرى لتدريب طبقة التلافيف لإعادة بناء الصور المدخلة. وبهذا يمكن بناء شبكات عصبية كبيرة الحجم باعتماد

التلافيف (convolutional neural networks) والتي يمكن تدريبها بسرعة أقل من تقنية النشر العكسي والتي قد تتطلب أسابيع لتدريب الشبكات. وقد أظهرت نتائج المحاكاة أن الشبكات العصبية الكبيرة الحجم بالاعتماد التلافيف والمدربة باستخدام آلات التعلم السريعة للتشفير الذاتي بالاعتماد التلافيف تحقق نتائج تنافسية ومتجايزة بشكل كبير لأداء الآلة شعاع الدعم (support vector machines) وآلات التعلم السريعة التقليدية. وأخيرا، تهدف المساهمات المنجزة في هذه الرسالة إلى بدء اتجاه جديد لخوارزميات التعلم العميقة والتي تتميز بسرعة التعلم والكفاءة في آن واحد.

CHAPTER 1

INTRODUCTION

Inspired by the brain, artificial neural networks (ANNs) mimic the behavior of the neurons in mammalian brains [1]. It could learn highly non-linear and complex functions by interconnecting as many neurons as needed [2]. A typical structure of ANNs, shown in Figure 1.1, has an input layer, one or more hidden layers, and an output layer. Each hidden layer extracts a more complex representation of the previous layer such that the last hidden layer would have a representation meant to discriminate between samples of different classes.

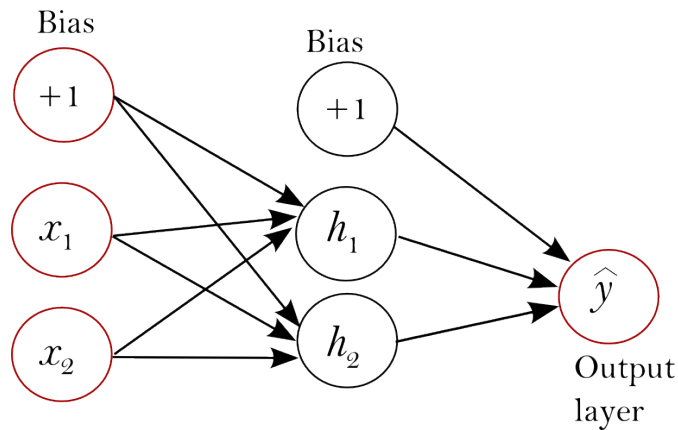


Figure 1.1: ANN with one hidden layer.

As a result, ANNs have long been useful in diverse applications including data mining [3] and reinforcement learning [4]. Compared to other tools in the literature, ANNs have been most successful in applications in which humans excel: image, video and audio recognition. In fact, ANNs try to mimic the human brain in how it learns new patterns [1]. Recently, ANNs have outperformed humans in accurately recognizing different traffic signs [5].

Over the last few years, ANNs research experienced a boom and a new trend of networks emerged. These networks, known as deep belief networks (DBN), use the idea of pre-training the network to develop better functions representing the data pattern. Sparse autoencoders, restricted boltzmann machines, and multi-layer perceptron are some of the popular algorithms for pre-training DBNs [6].

However, DBNs usually take long to train. For very large datasets, DBNs could take weeks of training time because backpropagation uses the slow stochastic gradient descent algorithm to optimize their solutions [7].

On the other hand, extreme learning machines (ELMs) emerged as a fast learning algorithm for training single-hidden layer perceptrons (SLFNs). ELMs are faster than ANNs because they use the least-square algorithm instead of back-propagation. Further, least-squares provide better generalization performance [8].

ELMs, being fast and efficient, were successful for many applications. But little attention was given to their ability to train large deep belief networks, learn spatio-temporal patterns, and extract latent features. These three limitations are studied in this thesis, as opportunities to improve ELMs.

1.1 Problem Statement

The thesis aims at developing deep learning algorithms that are efficiently trained while achieving acceptable performance. This thesis addresses the following two issues,

- First is the lack of fast algorithms to learn spatio-temporal patterns, which is addressed by proposing recurrent ELMs.
- Second is the lack of fast feature extractor models, which is addressed by proposing ELM-based autoencoders and convolutional autoencoders.

1.2 Thesis Contributions

The main contributions of the thesis are listed below.

- Proposed a least squares approach to recurrent neural networks.
- Designed a least squares based sparse autoencoders and convolutional autoencoders for feature extraction.
- A novel structure for stacked ELMs to characterize convolutional neural and deep learning networks.

1.3 Thesis Breakdown

The thesis contains the following chapters. Chapter 2 explains background information and related work in artificial neural networks (ANNs) and extreme learning

machines (ELMs). Chapter 3 describes the proposed least squares approach to recurrent ELMS. Chapters 4 and 5 give a detailed description of the least square approach to sparse auto-encoders and convolutional autoencoders, respectively. Finally, chapter 6 concludes the thesis and discusses future work.

CHAPTER 2

LITERATURE REVIEW

2.1 Notations

Table 2.1 shows the set of notations used throughout the thesis. Capital letter symbols represent matrices, whereas small letter symbols represent an entry or a row of their corresponding matrices. Artificial neural networks (ANNs) and extreme learning machines (ELMs) might have different notations as indicated in Table 2.1.

2.2 Overview

In 1958, Rosenblatt [9] developed “Preceptron” as the first adaptive learning algorithm. It can learn linear data patterns through trial and error. Consider a set of 4 samples as shown in Figure 2.1, where crosses represent class 1 data samples belong and circles represent class 2 data samples. Suppose the algorithm is

Table 2.1: Notations.

Symbol	Description
X	The matrix representing the input training samples, $X \in R^{n \times m}$ where n is the number of samples and m is the number of features
T	The vector representing the target sample labels, $T \in R^{n \times o}$ where o is the number of output neurons.
Y	The vector representing the predicted sample labels, $Y \in R^n$
ANNs: W_1 ; ELMs: W	The input weight matrix between the input layer and the hidden layer, $W_1, W \in R^{m \times L}$ where L is the number of hidden neurons
ANNs: W_2 ; ELMs: β	The weight matrix between the hidden layer and the output layer, $W_2, \beta \in R^{L \times o}$ where o is the number of output neurons
ANNs: b_1 ; ELMs: b	the hidden layer bias vector $b_1, b \in R^L$
b_2	the output layer bias vector $b_2 \in R^o$
β_{output}	the outgoing weights of the last hidden layer.
W_c	the outgoing weights of the context layer in recurrent ELMs.
C	the values representing the context layer neurons in recurrent ELMs.
$g(\cdot)$	the activation function $g(\cdot) : R \rightarrow R$

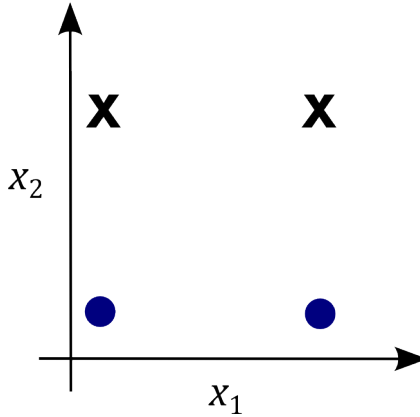


Figure 2.1: XOR problem formulation.

learning a decision function of the form,

$$f(X) = w_0x_0 + w_1x_1 \quad (2.1)$$

where w_0 and w_1 are weights and x_0 and x_1 define the sample coordinates .

First, the algorithm assigns random values to w_0 and w_1 , constructing an arbitrary decision function as shown in Figure 2.2. Since the samples are not separated by the decision function, the algorithm updates its weights as follows,

$$w_j = w_j + \alpha(t_i - y_i)x_{ji} \text{ for each sample } 0 \leq i \leq n \quad (2.2)$$

where α is the learning rate.

The weights continue to update until the decision boundary positions itself such that it separates the samples by their classes. The end result is similar to the boundary in Figure 2.3.

A problem with this algorithm is that it cannot solve non-linear patterns. For example, it is clear that a single line cannot separate the samples in Figure 2.4,

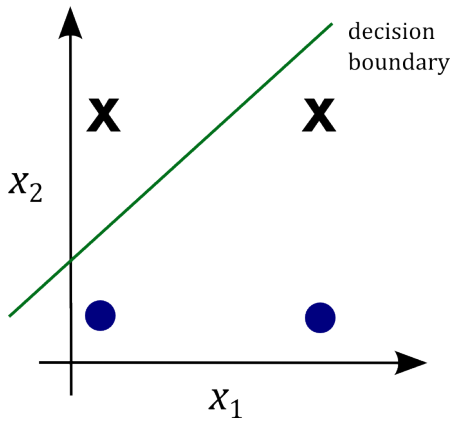


Figure 2.2: The green line represents the decision boundary.

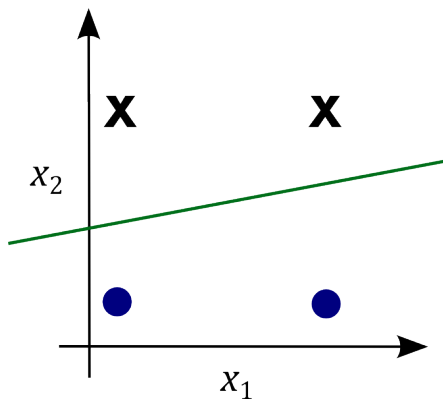


Figure 2.3: The trained decision function separating the two classes.

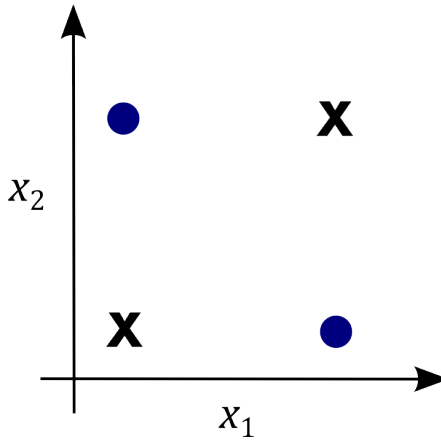


Figure 2.4: 4-Point XOR example.

which is known as the XOR problem.

This issue made the algorithm unusable for complex problems, and as a result, Artificial Neural Networks (ANNs) fell out of favor and experienced a stagnation for a decade after [10]. At that time, other algorithms like Support Vector Machines [11] were more suitable for complex tasks [12].

But in the early 1980s, backpropagation introduced by Rumelhart et al. [13] brought a new light to ANNs. It allowed ANNs to learn non-linear functions and therefore became useful for many real-life problems [14]. They use hidden layers to construct curve-shaped or multi-line decision boundaries, allowing it to easily solve the XOR problem like in Figure 2.5.

Despite being useful for many complex problems, ANNs research subsided once again because other algorithms like Decision Trees and Support Vector Machines (SVMs) hold similar capability, yet train much quicker [10]. SVMs [11] used the kernel trick [15] and utilized dual optimization [16] to learn highly complex patterns in a convenient amount of time. But backpropagation could take weeks

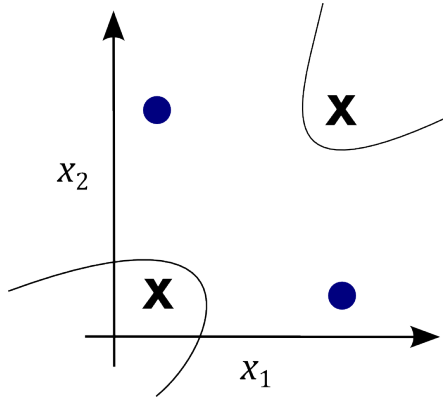


Figure 2.5: A multi-curve solution of the XOR problem.

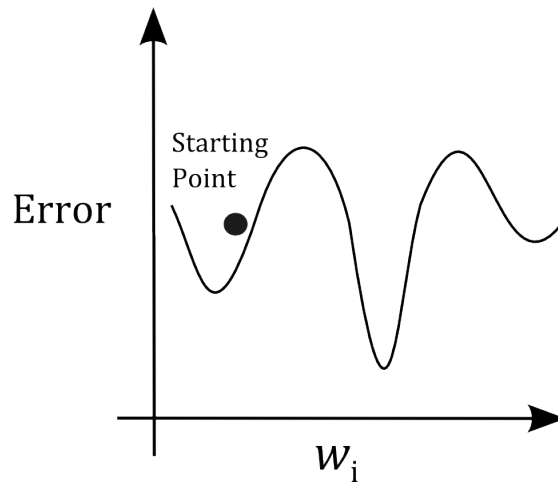


Figure 2.6: Weight initialization effect.

to train large networks and they bear other problems as well [17], [18], [19]. When backpropagation constructs the decision boundary, its effectiveness depends on the initial values of the assigned weights. Since the function is non-linear and complex, there could be many local minima, many of which resulting in higher error rate than the global minima. Therefore, optimizing the decision function could land the weights in a poor local minima [20]. Figure 2.6 shows how initializing random weights could have its values land near a local optima when the starting value is far from the global minima.

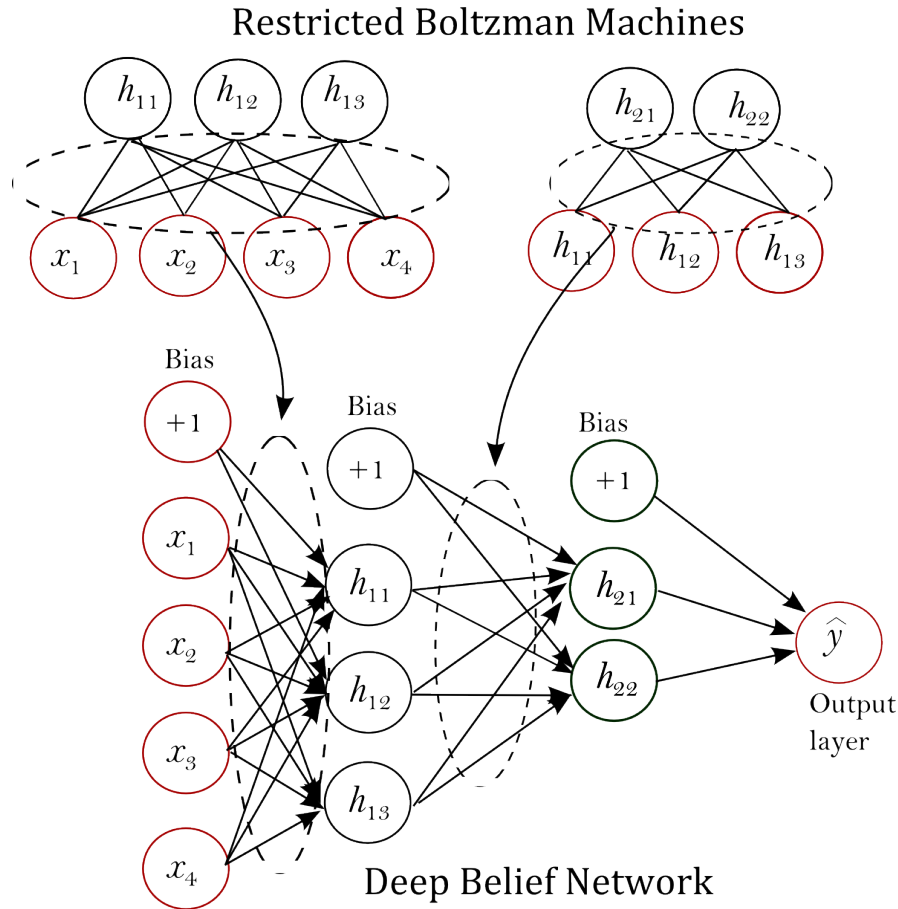


Figure 2.7: DBN pre-trained by RBM.

Soon after, in 2006, Hinton et al. [6] addressed the weight initialization issue with a new concept known as Deep Learning. Instead of randomizing weights, Hinton et al. [6] trains restricted boltzmann machines (RBMs) that provide good initial weights for the multi-layer perceptron network. Figure 2.7 displays this process.

Basically, for each layer, a restricted boltzmann machine trains on reconstructing the features given in the previous layer. The weights developed by the RBM become the initial outgoing weights of that layer [21].

RBMs are unsupervised learning algorithms that extract latent, invariable

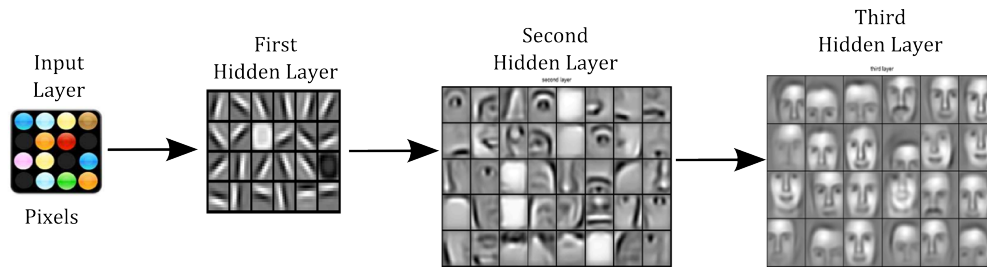


Figure 2.8: Levels of abstraction.

features representing the input data. As a result, the features constructed for each layer would constitute levels of abstractions of the input images, as shown in Figure 2.8. These features help classifiers better discriminate between sample classes.

2.3 Overview of ANN Algorithms

This section provides background information on five popular architectures of ANNs, including:

- Logistic regression
- Multi-layer perceptron
- Recurrent ANNs
- Sparse autoencoders
- Deep belief network

2.3.1 Logistic Regression

Logistic Regression [22] is a popular algorithm in machine learning. Given a matrix $X \in R^{n \times m}$, and an output matrix T representing the sample targets. The predictive function is the weighted summation of the input features followed by an activation function $g(\cdot) : R \rightarrow R$, which can either be logistic function [23], hyperbolic tangent function [24], or the like. The logistic regression function can then be written as,

$$Y = g(W^T X + b) \tag{2.3}$$

where W is the weight matrix representing the coefficients of the weighted summation.

Prediction comes mainly in three forms: binary Classification, multi-class classification, and regression [25].

Binary classification is to classify samples to either positive or negative class. An example being to label a face either as authorized (positive case) or non-authorized (negative case). Logistic regression, typically, uses the logistic function to output a value between zero and one. A threshold, usually set to 0.5, would assign input samples to the positive class if their predicted values is larger than 0.5; otherwise, they are assigned to the negative class.

Logistic regression optimizes W by first randomizing its values, and then computing the function to get an arbitrary output. Next, a loss function measures the difference between the real target value and the arbitrary output.

There are many loss functions, each having its own advantages [26], [27], [28]. Some of the popular loss functions are listed below (please note that the notations are explained in Table 2.1),

Square error,

$$J(W, b) = \frac{1}{2}(Y - T)^2 \quad (2.4)$$

Log loss,

$$J(X) = -\frac{1}{n} \sum_{i=1}^n [T \log(Y) + (1 - T) \log(1 - Y)] \quad (2.5)$$

Hinge loss,

$$J(X) = \max(0, 1 - Y \cdot T) \quad (2.6)$$

Further, some popular activation functions are given below,

The logistic function is written as,

$$f(x) = \frac{1}{(1 + e^x)} \quad (2.7)$$

and the tan hyperbolic function is represented by,

$$f(x) = \tanh(x) \quad (2.8)$$

Finally, the weight update is defined as follows,

$$W = W + \nabla_J W \quad (2.9)$$

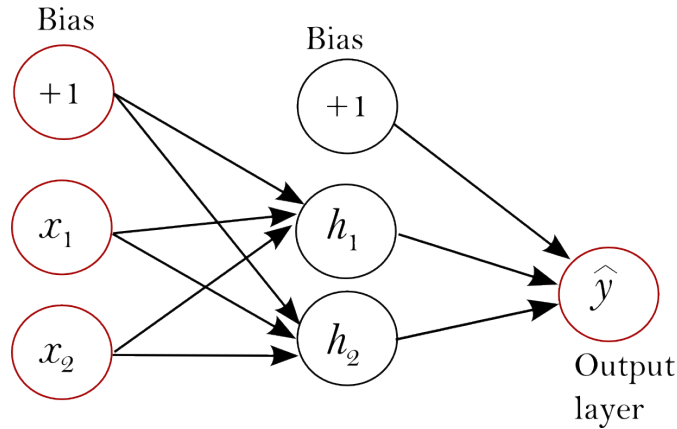


Figure 2.9: MLP with one hidden layer.

where ∇W is the derivative of the loss function $J(W, b)$ with respect to W .

The process repeats itself until the loss function is smaller than a special threshold ϵ .

2.3.2 Multi-Layer Perceptron

Multi-layer perceptron (MLP) can have one or more hidden layers, as shown in Figure 2.9. It can learn non-linear, complex functions as each hidden layer transforms the data in the previous layer to a more complex representation.

It contains three types of layers: input, hidden, and output layer.

The input layer consists of a set of neurons $\{x_i | i = 1, 2, \dots, n\}$ representing the input data as features. The number of neurons in that layer matches the number of features in the input. The hidden layers transform data by a linear transformation followed by the application of a non-linear activation function. The output layer receives the data from the last hidden layer and transforms them into output values.

As a result, multi-layer perceptron with one hidden layer has the following function:

$$Y = W_2 g(X \cdot W_1 + b_1) + b_2 \quad (2.10)$$

where W_1 represents the input weights matrix and W_2 represents the hidden weights. b_1 and b_2 are bias vectors and $g(\cdot) : R \leftarrow R$ is the activation function.

If the cost function is the square error, then the network minimizes the following function,

$$J(W, b) = \frac{1}{2}(Y - T)^2 \quad (2.11)$$

The cost function, $J(W, b)$, backpropagates across the layers in the network in order to update W_1 and W_2 for a better decision boundary. This is done by computing the gradients of W_1 and W_2 based on the cost function. Using the chain rule, the gradient is expressed as,

$$\nabla W_j = \frac{\partial J}{\partial \hat{Y}} \frac{\partial \hat{Y}}{\partial Y} \frac{\partial Y}{\partial W_j} \quad (2.12)$$

where,

$$\hat{Y} = W_2 g(X \cdot W_1 + b_1) + b_2 \quad (2.13)$$

and

$$Y = g(\hat{Y}) \quad (2.14)$$

To find ∇W_2 , the first term in eq. (2.12) is computed as,

$$\frac{\partial J}{\partial \hat{Y}} = \frac{\partial(\frac{1}{2}(Y - T)^2)}{\partial \hat{Y}} = -(Y - T) \quad (2.15)$$

The second term is written as,

$$\frac{\partial \hat{Y}}{\partial Y} = \hat{Y}(1 - \hat{Y}) \quad (2.16)$$

where the result is the well-known derivative of the logistic function. Finally, the third term is given by,

$$\frac{\partial Y}{\partial W_2} = \frac{\partial(W_2 g(X \cdot W_1 + b_1) + b_2)}{\partial W_2} = g(X \cdot W_1 + b_1) \quad (2.17)$$

As a result, the weight gradient for W_2 is,

$$\nabla W_2 = -(Y - T) \cdot \hat{T}(1 - T) \cdot g(X \cdot W_1 + b_1) \quad (2.18)$$

For W_1 , the gradient is computed as follows,

$$\nabla W_1 = -\left(\frac{\partial J}{\partial \hat{Y}} \frac{\partial \hat{Y}}{\partial Y} \frac{\partial Y}{\partial H}\right) \frac{\partial H}{\partial \hat{H}} \frac{\partial \hat{H}}{\partial W_1} \quad (2.19)$$

where $\hat{H} = X \cdot W_1 + b_1$ and $H = g(\hat{H})$. While $\frac{\partial J}{\partial \hat{Y}}$ and $\frac{\partial \hat{Y}}{\partial Y}$ are already computed,

$\frac{\partial Y}{\partial H}$ is written as,

$$\frac{\partial Y}{\partial H} = \frac{\partial(W_2 \cdot H + b_2)}{\partial H} = W_2 \quad (2.20)$$

Next, the derivative of the activation function is,

$$\frac{\partial H}{\partial \hat{H}} = H(1 - H) \quad (2.21)$$

Finally, the differentiation of the last term in ∇W_1 is,

$$\frac{\partial \hat{H}}{\partial W_1} = \frac{\partial X \cdot W_1}{\partial W_1} = X \quad (2.22)$$

Therefore, the gradient for the input weights W_1 is,

$$\nabla W_1 = -(Y - T) \cdot Y(1 - T) \cdot W_2 - \left(\frac{p}{\hat{p}}\right) \cdot H(1 - H) \cdot X \quad (2.23)$$

The weights W_1 and W_2 are then updated as follows,

$$W_1 = \alpha \nabla W_1 + W_1 \quad (2.24)$$

$$W_2 = \alpha \nabla W_2 + W_2 \quad (2.25)$$

where α is the learning rate ranging between 0 and 1.

The update for W_1 and W_2 is repeated many times until the cost function, $J(W, x)$, is smaller than some set value ϵ .

This algorithm is known as backpropagation. It has been successful in many applications including heart disease diagnosis [29], face recognition [30], damage detection of bridge structures [31], and image segmentation [32].

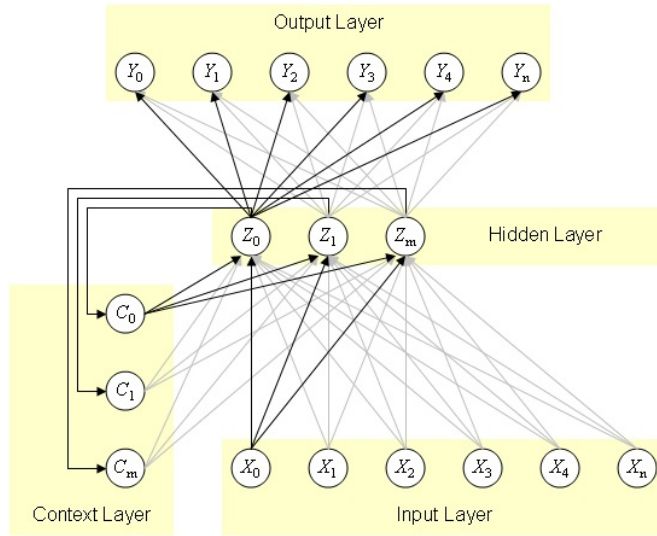


Figure 2.10: Elman Neural network.

Many new ANN algorithms are designed based on MLP models using back-propagation. The algorithms are discussed in the upcoming sections.

2.3.3 Recurrent ANNs

Recurrent ANNs use an extra layer, usually known as the context layer, to learn spatio-temporal patterns. This layer retains time dimensional features - acquired from the previous time steps - to help in predicting the output of the next time step. Elman networks [33] is one popular method that can learn time-series datasets. Its context layer maintains a certain information about the last time step - like retaining the values of hidden or output neurons. To put this formally, suppose the context layer C contains the entries c_1, c_2, \dots, c_k where k is the number of neurons in the context layer. The value of c_i equals the value of h_i of the previous time step and therefore $\{c_i | i = 1, 2, \dots, k\}$ are considered time-dependant features. Figure 2.10 shows the structure of such network

The network trains using backpropagation on per sample basis. The values of the hidden layer h_i are copied to their respective nodes c_i neurons in the context layer. Next, the network updates the input and hidden weights and the outgoing weights of the context layer as well. This would create a relationship between time steps, providing more features that would otherwise improve the prediction function for time-series datasets.

2.3.4 Sparse autoencoders

Sparse autoencoders (SAE) [34] extract new features to get interesting structural information from the input data. As such, they may outperform hand-crafted features extracted using the Fourier transform [35], discrete cosine transform [36], discrete wavelet transform [37], etc.

SAE uses backpropagation to learn reconstructing the input data. The hidden layer, shown in Figure 2.11, develops these new features as it reconstructs the input data.

In order to get meaningful, sparse features, SAE trains with a sparsity constraint using Kullback-Leibler divergence (KLD). Therefore, SAE minimizes the square error function, subject to the constraint that the hidden activations (the values in the hidden layer) are close to a small set value p . Below is the resulting cost function:

$$J(W, b) = \frac{1}{2}(Y - \bar{X})^2 + KL(p||\hat{p}) \quad (2.26)$$

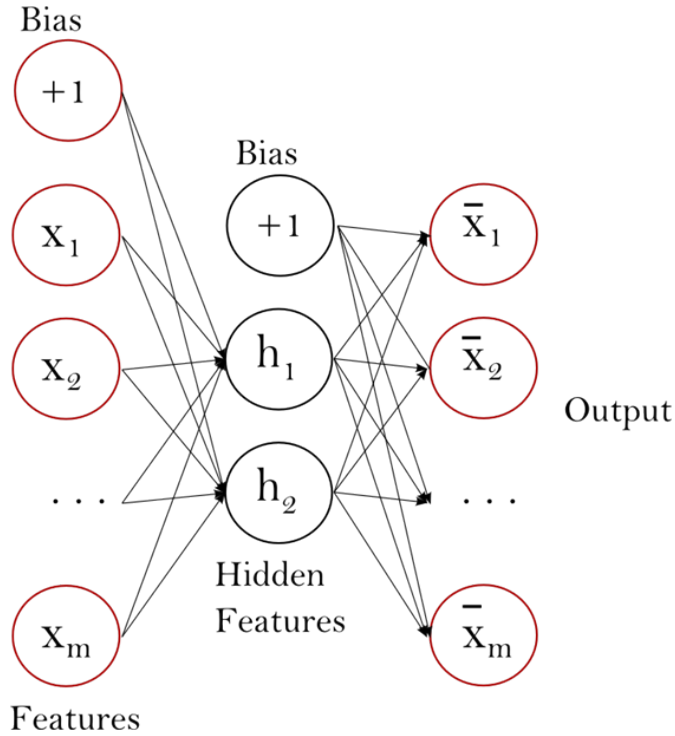


Figure 2.11: Sparse autoencoder network.

where \bar{X} is the target which is usually set as the input data, Y is the predicted value produced by a forward pass on the network, and $KL(p||\hat{p})$ is the sparsity term given as,

$$KL(p||\hat{p}) = p \log \frac{p}{\hat{p}_j} + (1 - p) \log \frac{1 - p}{1 - \hat{p}_j} \quad (2.27)$$

where p controls the values of \hat{p} , the average hidden activations of each hidden neuron over the data. $\hat{p} = H/\text{number of samples}$

If p is set to 0.2, then the relationship between KLD and \hat{p} is shown in Figure 2.12.

Given an SAE network with L hidden neurons and n training samples. Consider matrix $X \in R^{n \times m}$ defining the input vectors as $\{x_i | x_i \in R^m\}$ for $i = 0, 1, \dots, n$ where m is the number of input features representing a vector x_i . Also consider

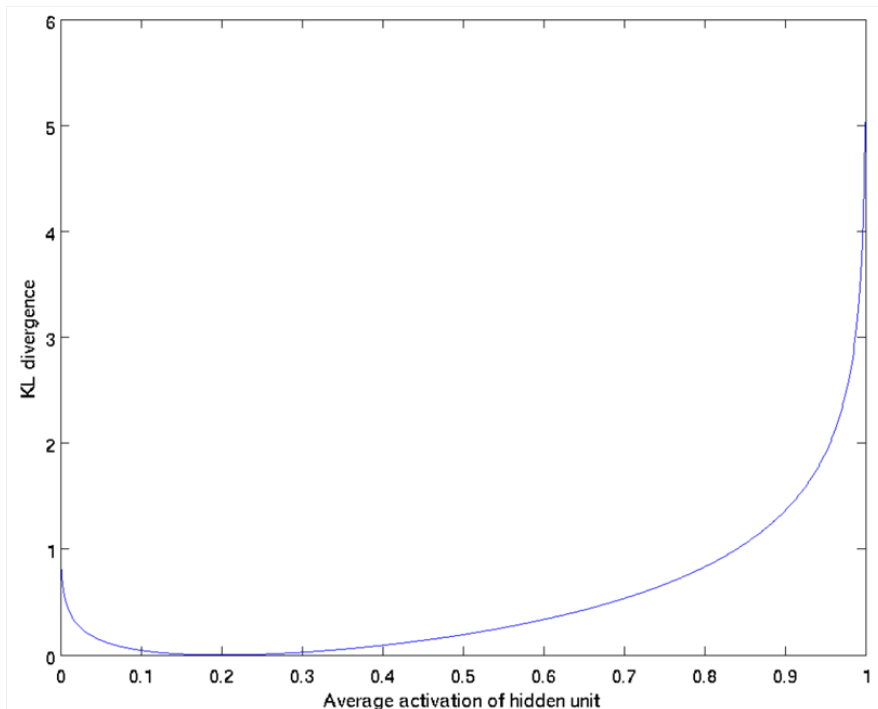


Figure 2.12: Relationship between KLD and \hat{p} when $p = 0.2$.

the bias vectors $b_1 \in R^L$ and $b_2 \in R$, and the target reconstruction matrix as X . Consider the matrices $W_1 \in R^m \times R^L$, and $W_2 \in R^L$ representing the outgoing weights of the input layer and the hidden layer, respectively.

To update the weights, ∇W_1 and ∇W_2 are computed. ∇W_2 is computed the same way as for the MLP case. But ∇W_1 is little different due to the Kullback-leibler Divergence term in the objective function, which is written as,

$$\nabla W_1 = -(Y - \bar{X}) \cdot \bar{X}(1 - \bar{X}) \cdot W_2 - \left(\frac{p}{\hat{p}} + \frac{1-p}{1-\hat{p}}\right) \cdot H(1-H) \cdot \bar{X} \quad (2.28)$$

Sparse autoencoders have been successful in many applications, especially image recognition [38]. It can also be used to train large DBNs [6], [39], to help them

learn better solutions.

2.3.5 Deep Belief Networks

Deep belief networks (DBNs) had been under the focus of research for several years. But only recently did new methods emerge that made training DBNs feasible. DBNs had the issue of poor weight initialization where weights were assigned with random values, leading to solutions that lie in poor local optima and taking long time to converge. Further, large networks suffer from the error diffusion problem, where the farther the layer is from the output layer, the less momentum the error can have in updating the weights. As such, the weights adapt slowly and thus take long to converge.

But in 2006, Hinton et al. [6] introduced a new concept involving pre-training large networks in order to initialize weight values close to the global minima.

Greedy layer-wise training using sparse autoencoders is one way to train deep belief networks (DBNs), shown in Figure 2.13. Suppose X is the input matrix, W_1 , W_2 , and W_3 are weight matrices, H_1 and H_2 are hidden layer activations and T is the target matrix.

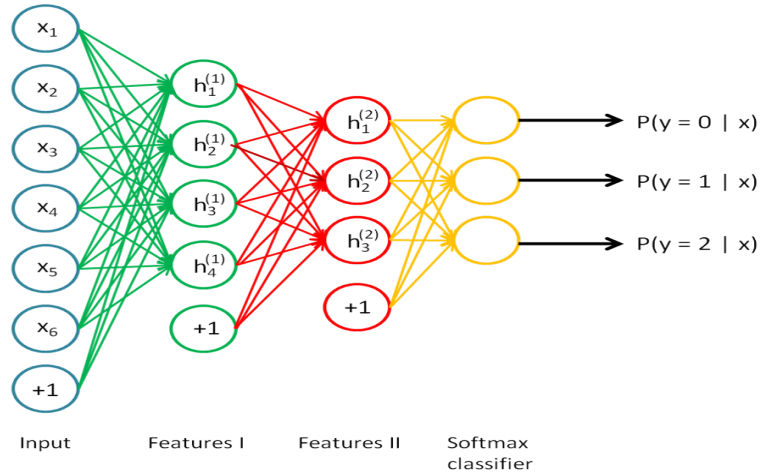


Figure 2.13: Network with two hidden layers.

First, X is fed into a sparse auto-encoder (SAE), shown in Figure 2.13. Once trained, the weights leading to the activations H_1 in the hidden layer of the SAE denote the initial values for W_1 .

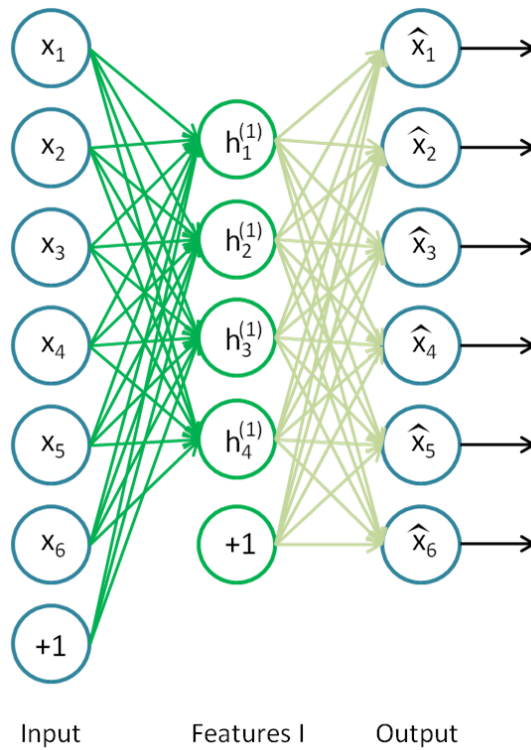


Figure 2.14: First hidden layer training.

Next, H_1 is fed to another SAE, yielding a second set of hidden activations H_2 , whose input weights denote the initial values of W_2 .

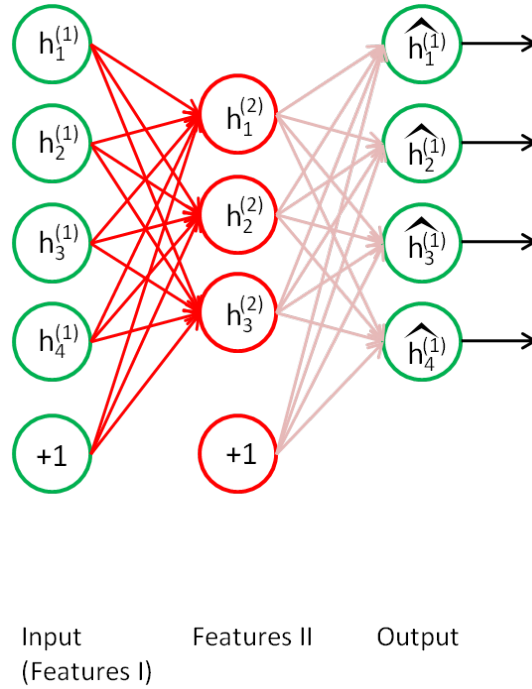


Figure 2.15: Second hidden layer training.

Finally, a softmax classifier uses H_2 as input features to learn to classify the images based on the target vector y . The trained weights become the initial values of W_3 .

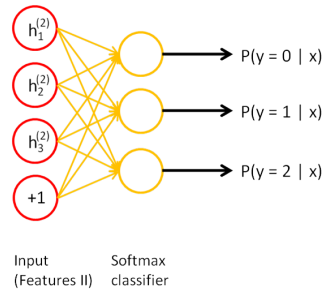


Figure 2.16: Output layer training.

This ends the pre-training phase and W_1 , W_2 , and W_3 of the two-hidden layer network are set.

Next is the fine-tuning phase where the algorithm runs backpropagation to improve on the initialized weights, developing the final decision boundary. Such decision boundary is likely to be more robust than if it was developed with weights randomly initialized.

2.3.6 Current Performance of Deep Learning Networks

Many variants of Deep learning networks are available in the literature. This section describes some of these algorithms in detail.

Many of the recent algorithms use restricted boltzmann machines (RBMs) to train artificial neural networks (ANNs). Ruslan and Hinton [40] proposed a new scheme to train DBN machines. It uses variational approximations and persistent Markov chains to estimate data-dependent expectations for RBMs, which in turn pre-train DBNs. This allows training millions of parameters of a large DBN in a feasible amount of time. Using a 2-layer RBM for pre-training, DBN achieved a 0.95% error rate on the MNIST dataset.

In a similar work, Sun et al. [41] proposed a hybrid deep learning algorithm for face verification. It uses convolutional layers trained using RBMs to verify faces under different conditions. The network extracts robust, global features, which allows it to achieve 89.6% accuracy in the CelebFaces dataset ¹.

Part of the literature involves feature extraction as well. Vincent et al. [42]

proposed a denoising autoencoder. It is different from regular autoencoders, in that it corrupts the input matrix X using zero-masking or Gaussian noise before reconstruction. However, the objective function remains the same. This algorithm was able to learn Gabor-like edge detectors from natural image patches. It helped achieve 98.7% accuracy on the MNIST dataset.

Labusch et al. [43] proposed using SVM to train on extracted unsupervised features. These features are extracted using a two-stage process. First, a Sparsenet [44] algorithm extracts basis components from the input data. Second, a local maximum operation is used to extract local coefficients from these basis components to preserve local shift invariance. Next, the model trains using these local coefficients for digit recognition. It achieved an accuracy of 98.73% on the MNIST dataset.

Different DBN algorithms were successfully applied on various problems. Shen and Song [45] proposed deep networks that can detect eye fixations. The algorithm is able to extract salient features including textures, junctions and parallelism using a 4-stage model. In the first stage, the model whitens the input data. In the second step, it uses sparse coding to extract features. In steps 3 and 4, it applies max-pooling to get more robust features on which a linear SVM trains. In terms of area under the ROC (Receiver Operating Characteristic) curve (AUC) [46], this model achieved a performance of 0.96 on the FIFA dataset [47].

Ngiam et al. [48] proposed multimodal deep learning networks which learn features over multiple modalities. In other words, it uses both audio and video

features, instead of just audio features, for recognition. This algorithm achieved competitive performance in speech classification, obtaining 91.7% accuracy on the CUAVE dataset ².

Ciresan et al. [49] proposed a multi-column deep neural network for traffic sign classification where the authors trained large convolutional networks of 9 layers. The algorithm uses an ensemble of such networks whose votes' average are taken to determine the predicted class of a traffic sign. It achieved an accuracy of 98.47%, surpassing even humans' performance in traffic sign recognition.

Coates and Ng [50] proposed a feature representation algorithm using K-means. This algorithm is fast and can scale up for a large number of samples. It combines K-means clustering, whitening, and local receptive fields to train large deep neural networks, achieving highest scores in CIFAR-10, and STL-10 datasets. Rothacker et al. [51] presents feature representation for unconstrained handwriting recognition. With the help of Hidden Markov Models, it extracts Bag-of-Features as a model for representing handwritten character images. Empirically, it achieved competitive results compared to other state-of-art algorithm for Arabic datasets. Wang et al. [52] proposed an algorithm that does not rely on hand-engineered features for text recognition. Instead, this algorithm uses multi-layer perceptron and unsupervised learning to train on a set of images containing characters, and extract new representative features. The algorithm achieved highly competitive performance on Street View Text and ICDAR 2003.

Kavukeuoglu et al. [53] proposed an algorithm that learns locally-invariant

feature descriptors. First, it uses a set of filters - like edge detectors - followed by a non-linear transform on the output. Finally, a pooling layer operates on the output. It is an unsupervised learning algorithm that generates topographic filter maps, extracting features of different orientations, scales and positions. Compared to SIFT features, features extracted by this algorithm achieve competitive results and even better results on some tasks. Coates et al. [54] proposed an unsupervised learning algorithm that can learn high-level, invariant features sensitive to commonly-occurring objects - like human faces. This algorithm is meant to deal with data that is completely unlabeled, which must be of large size (millions of samples) in order to learn invariant features. The features extracted by this system were found to be invariant to distortions like scale and translation.

Le et al. [55] used a large scale unsupervised learning algorithm that learns high-level features robust to out-of-plane rotation, scaling, and translation. A nine layer sparse autoencoder was trained on a 10 million 200x200 pixel images downloaded from the internet. A cluster of 1000 machines trained the algorithm using parallel, asynchronous stochastic gradient descent in three days. It achieved a 70% relative improvement in accuracy on ImageNet, compared to state-of-the-art algorithms. Pang et al. [56] proposed a new hand-engineered feature extraction algorithm that is affine invariant while carrying other advantages. It combines affine scale invariant feature transform (ASIFT) and the merits of speeded up robust features (SURF). It selects the latitudes and longitudes to speed up feature extraction while making sure the images are kept close to any other possible view.

Experimental results show that this algorithm achieved better than ASIFT and SURF for several test images.

Yang et al. [57] presented an SVM algorithm with a modified kernel that reduces the standard training time complexity of $O(n^2 \sim n^3)$ to $O(n)$ where n is the number of samples. It uses a modified spatial pyramid matching (SPM) kernel which uses sparse coding and multi-scale spatial max pooling instead of vector quantization. The results show that this algorithm considerably outperforms the state-of-the-art linear SOM kernel and better than nonlinear SPM kernels on several benchmarks.

Bastien et al. [58] described a deep self-taught learning algorithm that takes advantage of what is known as out-of-distribution learning. They applied a noise generator that transforms character images by adding slant, local elastic deformations, changes in thickness, background images, grey level changes, and other types of noise. Empirically, learning algorithms that train on such images achieved higher accuracy in handwritten character recognition than previous state-of-the-art algorithms. Aly [59] compared the performance of different features for face recognition using Nearest Neighbour and Nearest Cluster Center. The results show that SIFT features make for higher accuracy than Eigen Faces and Fisher Faces on AT&T and Yale datasets.

Stolyarenko and Dershowitz [60] proposed a character recognition algorithm that segments words into letters and classifies them using a grid of SIFT descriptors. Applying different window sizes, the segmentation achieving maximal confi-

dence is set. On the PATS-A01 dataset, the algorithm achieved 87-96% accuracy for classifying letters and 74-88% for classifying words. Bay et al. [61] proposed speeded up robust features (SURF) that have empirically outperformed other feature descriptors such as SIFT features. SURF features are scale- and rotation-invariant and relatively fast to extract. They are extracted using image integrals for image convolutions. This algorithm achieved state-of-the-art performance on object recognition datasets.

Ke and Sukthankar [62] proposed the combination of principal component analysis (PCA) and SIFT features to achieve features robust to various image deformations. The features encode salient aspects of an image gradient on which PCA features are extracted. The results show that these type of features increase performance and achieve faster matching. Grana et al. [63] proposed an algorithm that integrates Oriented Fast and Rotated Brief (ORB) features in bag of words model. It uses k-means like approach to deal with the binary string nature of ORB features. While this algorithm did not achieve state-of-the-art performance for the ImageCLEF 2011 dataset, it required significantly lower computational requirements, making it feasible for small devices like mobiles.

Known as word spatial arrangement (WSA), Penatti et al. [65] presents an approach that uses the bag-of-visual-words model to represent the spatial arrangement of visual words. It splits the image space into quadrants to encode the relative position of visual words. The extracted feature vectors using this method is useful for both image retrieval and classification. Such feature vectors - though

compact and encodes only spatial information - achieved substantial performance for image classification.

Yu et al. [66] described an algorithm that combines different mid-level features that complement each other and therefore achieve substantially higher performance than if the individual features were selected. The algorithm compares the integration of SIFT and Local Binary Patterns (LBP) descriptors and Histogram of Oriented Gradients (HoG) and LBP descriptors. Experimentally, using K-means clustering on SIFT-LBP features achieved state-of-the-art performance compared to existing algorithms.

2.4 Overview of Extreme Learning Machine Algorithms

This section includes an overview of three popular algorithms related to Extreme Learning Machines:

- the traditional extreme learning machine (ELM) algorithm;
- weighted ELMS; and
- sequential ELMs

2.4.1 Mathematical Notation

Assume a single hidden layer feedforward network SLFN with L hidden neurons and n training samples. Consider a matrix $X \in R^{n \times m}$ defining the input vectors as

$x_i | x_i \in R^m$ for $i = 0, 1, \dots, n$ where m is the number of input features representing a vector x_i , the bias vector $b \in R^m$ and the target vector $T \in R^n$ defined as t_i for $i = 1, 2, \dots, n$ where t_i is the output value of x_i . Let us also consider the matrices $W \in R^{m \times L}$, and $\beta \in R^L$. Then, the output of SLFN is defined as [8], [67],

$$f(x) = \beta g(X \cdot W + b) \tag{2.29}$$

where $g(x) : R \rightarrow R$ is the activation function (e.g. logistic and hyperbolic tanh).

2.5 Extreme Learning Machines

Proposed by Haung et al. [8], [67] ELM represent a supervised learning algorithm that trains SLFNs. One main issue with the backpropagation algorithm is the slow learning speed. This limitation is attributed to the slow gradient descent algorithm used to update the weights iteratively.

ELMs, on the other hand, train very quickly and efficiently. They randomly assign input weights of an SLFN and use least-square solutions to find the hidden weights. The algorithm is also simple to implement as it involves a small set of matrix operations.

To put this in a mathematical formulation, the function given by ELM is,

$$\beta g(X \cdot W + b) = T \tag{2.30}$$

In a more compact form, it is written as,

$$H\beta = Y \tag{2.31}$$

where $H = g(X \cdot W + b)$.

It is easy to find β by taking the inverse of H and multiplying it with T .

If H is a non-singular square matrix, then taking the inverse is straightforward.

Otherwise, the pseudo-inverse of H need to be computed.

A popular pseudo-inverse function is the generalized Moore-Penrose [68].

Given a matrix H , its Moore-Penrose pseudo-inverse matrix is defined as,

$$H^+ = (H^T H)^{-1} H^T \tag{2.32}$$

After finding the pseudo-inverse H^+ . The hidden weight matrix is simply given by,

$$\beta = TH^+ \tag{2.33}$$

Clearly, the algorithm is simple to implement. Yet, it is a fast, efficient learning algorithm for many applications. ELMs have been used in mental task classification [69], analysis of power utilities [70], and transmission lines protection [71].

2.5.1 Weighted ELMs

A dataset is considered imbalanced when a class has much larger number of samples than that of other classes. Algorithms that assume a balanced distribution of the classes favor the majority class by pushing the decision function towards the minority class [72]. This problem occurs since such algorithms try to maximize the number of correctly classified samples and therefore labeling all classes with the majority class label could achieve the highest accuracy.

Over- and under- sampling are two methods for balancing the distributions [73]. Over-sampling adds new samples to the minority class so it becomes as large as majority class. The new samples are generated by duplicating the existing samples and adding Gaussian noise on its features. More formally, for a sample $x \in R$, a new sample is generated as follows,

$$x_{new} = x + \mathcal{N}(\mu, \sigma^2) \tag{2.34}$$

The under-sampling method removes a fraction of the samples in the majority class to have the same size as the minority class.

The issue with these methods is that they have parameters to adjust. It is difficult to know which samples to remove without compromising information quality of the dataset. Similarly, it is difficult to know how samples need to be replicated so that they don't misrepresent the dataset.

But Zong et al. [74] proposed weighted extreme learning machines to address imbalanced datasets. During training, It assigns higher misclassification cost to

samples belonging to the minority class. The cost is a function of the class distribution. In other words, if the ratio of the positive class to the negative class is 0.5, then samples in the positive class would receive half the misclassification cost than the negative class.

Recalling the mathematical notation in section , the optimization is given as,

$$\text{Minimize: } L = \frac{1}{2} \|\beta\|^2 + C \frac{1}{2} \sum_{i=1}^N \|\psi_i\|^2 \text{ Subject to:} \quad (2.35)$$

The Karush-Kuhn-Tucker theorem informs us that the constraint can be incorporated to the objective function using lagrange multipliers α_i , and therefore, we have,

$$\text{Minimize: } L = \frac{1}{2} \|\beta\|^2 + C \frac{1}{2} \sum_{i=1}^N \|\psi_i\|^2 - \sum_{i=1}^N \alpha_i (h(x_i)\beta - t_i + \psi_i) \quad (2.36)$$

The least-square solution to equation 2.36 is the zeros of the equation's derivative with respect to β, ψ, α . That is,

$$\text{Minimize: } L = \frac{1}{2} \|\beta\|^2 + C \frac{1}{2} \sum_{i=1}^N \|\psi_i\|^2 - \sum_{i=1}^N \alpha_i (h(x_i)\beta - t_i + \psi_i) \quad (2.37)$$

Substituting for the variables α , and ψ in equation 2.37, gives us,

$$\text{Minimize: } \beta = H^T \left(\frac{I}{C} + W H H^T \right)^{-1} H^T W T \quad (2.38)$$

C is the trade-off cost defining the amount of weight given to the minority class. An automatic way of computing C for a class is to take the reciprocal of the number of samples in that class. More formally,

$$W_i = 1/\text{Number of samples} \quad (2.39)$$

where W_i is the weight for class i .

Weighted Extreme Learning Machines are useful for many applications. Few popular problems involving imbalanced data are Software Defect Prediction, Cancer Classification, and water leakage prediction.

2.5.2 Sequential ELMs

Memory can be an issue for Extreme Learning Machines (ELMs) in the face of large datasets. The reason is that the matrix representing the dataset must be put in whole in memory before training the network. These matrices can have over million rows for which some computers cannot handle.

Sequential ELMs, however, breaks the dataset into batches, small enough to fit in memory. The algorithm can recursively update its hidden weights β by learning each batch at a time. The recursive algorithm does not overlook information. In fact, it rarely performs worse than standard ELMs. This is because the recursive algorithm is simply another representation of the standard ELM function.

Liang et al. [75] developed sequential Extreme Learning machine (S-ELM), which rests on these two principles. First, the algorithm discards the batches that

have been learned. Therefore, the memory requirement should only satisfy the size of the largest batch without it being an issue to learn the whole dataset. Second, S-ELM incorporates the computations of all batches to get the same performance as though it has trained on the dataset as a whole.

Let X_0 be a chunk of size N_0 from the original data; H_0 be a the hidden activations generated by a kernel; and Y_0 be the target vector respective to X_0 . Then, S-ELM minimizes the following function,

$$\|H_0\beta_0 - Y_0\| \tag{2.40}$$

for which the least-square solution is,

$$\beta_0 = K_0^{-1}H_0^T Y_0 \text{ for } K_0 = H_0^T H_0 \tag{2.41}$$

With K_0 and X_0 in hand, it is possible to update β for the next batch X_1 without re-processing the X_0 , saving time and memory. It is easy to derive a sequential algorithm by showing that β_1 and K_1 are nothing but,

$$\beta_1 = K_1^{-1}[H_0H_1]^T[Y_0Y_1] = K_1(K_1\beta_0 - H_1^T H_1\beta_0 + H_1^T Y_1) = \beta_0 + K_1^{-1}H_1^T(Y_1 - H_1\beta_0) \tag{2.42}$$

$$K_1 = K_1^{-1}[H_0H_1]^T[Y_0Y_1] \tag{2.43}$$

It follows that the algorithm can be generalized as.

$$K_{k+1} = [H_0 H_1]^T [H_0 H_1] = K_0 K_0^{-1} H_0^T Y_0 + H_1^T Y_1 = K_0 \beta_0 + H_1^T Y_1 = K_1 \beta_0 - H_1^T H_1 \beta_0 + H_1^T Y_1 \quad (2.44)$$

$$\beta_{k+1} = \beta_k + K_{k+1}^{-1} H_{k+1}^T (Y_{k+1} - H_{k+1} \beta_k) \quad (2.45)$$

where k denotes the batch number.

2.6 Activation Functions

Many activation functions exist for ANNs. Logistic and hyperbolic tan are the two of the most popular activation functions.

Activation functions are beneficial, in that they add a non-linearity element to the algorithm's function, while scaling the output values between a fixed range.

ANNs with a hidden layer and a non-linear activation function can plot decision boundaries containing multiple curves to separate classes in non-linear datasets.

When the ANN uses the logistic function for the output, its derivative is used for the backward pass in backpropagation. It is given as,

$$\frac{d}{dx} f(x) = f(x) \cdot (1 - f(x)) \quad (2.46)$$

Figure 2.17 shows the standard logistic function

The hyperbolic tangent is a scaled version of the logistic function, which is

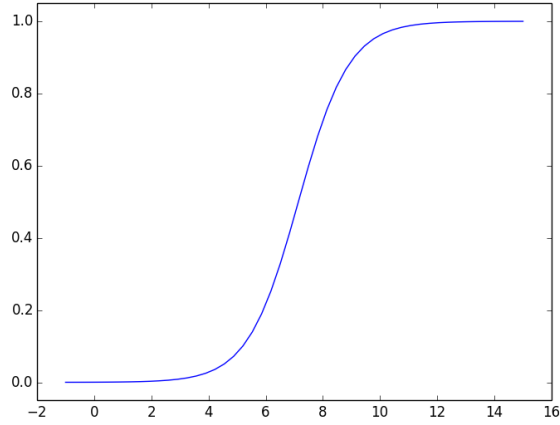


Figure 2.17: Plot of the logistic function

defined by,

$$2f(x) = 1 + \tanh\left(\frac{x}{2}\right) \quad (2.47)$$

The hyperbolic tan (\tanh) is written as,

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.48)$$

The derivative of hyperbolic tan is,

$$\frac{d}{dx} \tanh(x) = 1 - x^2 \quad (2.49)$$

Figure 2.18 shows the plot of hyperbolic tan.

Another class of activation functions are known as radial basis functions (RBFs). It includes these three functions, Gaussian RBF,

$$f(X, X_0) = \exp\left(-\frac{\|X - X_0\|^2}{2\sigma^2}\right) \quad (2.50)$$

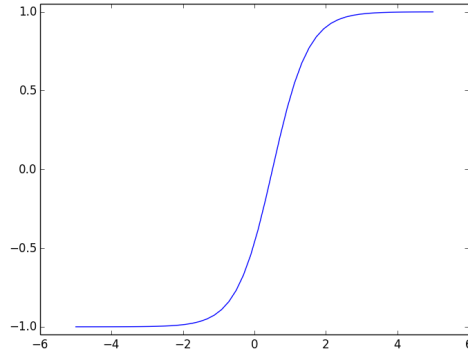


Figure 2.18: Plot of hyperbolic tan.

where σ defines the spread of the radius. Multiquadratics RBF,

$$f(X, X_0) = \sqrt{(\|X - X_0\|^2 + a^2)} \quad (2.51)$$

where a defines the spread of the radius. Inverse multiquadratics RBF,

$$f(X, X_0) = (\|X - X_0\|^2 + a^2)^{-1/2} \quad (2.52)$$

2.7 Kernel Functions

This section defines three types of popular kernels.

The most popular type is the random kernel. It multiplies, using the dot product, an input matrix $X \in R^{n \times m}$ with a randomized matrix $W \in R^{m \times o}$ to get a new representation of the input data. This is written as,

$$f(X) = X \cdot W \quad (2.53)$$

The second type is the polynomial kernel, which multiplies the input matrix X with X^T followed by the addition of a constant c . The result is taken to the power of an integer d .

$$f(X) = (X * X^T + c)^d \quad (2.54)$$

The third type is a class of RBF functions, whose properties are explained in section 2.4.

The objective of kernel functions is to get new set of features by transforming them from their original space domain S (dimensions) to another space domain V (or a new set of dimensions). The new domain space is meant to have a better representation of the original features, allowing for an easier separation of the sample classes during prediction. In fact, a linear learning algorithm can efficiently train on non-linearly separable data, if the data pass through a kernel function that transforms them to a linearly separable data.

2.7.1 Model Selection and Criteria

A fair benchmark is necessary to comparing performance between various learning algorithms. Different metrics are required for different datasets. For example, accuracy is not a suitable metric for imbalanced data; a naive learning algorithm that constantly output class 1 would achieve 99% accuracy for a dataset having 99% of its samples belonging to class 1. Other metrics - like G-mean and AUC - are more suitable in this case. For they evaluate the learning algorithm based on the ratio of correctly labelled samples to the number of samples in their respective

classes. Below is a list of some of the popular metrics.

Assume TP , FP , FN , and TN as True Positives, False Positives, False Negatives, and True Negatives, respectively.

The recall measure defines the ratio of the number of correctly classified documents in the category to the total number of documents in that category:

$$Recall = \frac{TP}{TP + FN} \quad (2.55)$$

Precision is the ratio of correctly classified documents in the category to the total number of documents classified in that category:

$$Precision = \frac{TP}{TP + FP} \quad (2.56)$$

Accuracy is another measure for classification performance calculated as,

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \quad (2.57)$$

F-measure is also prevalent for classification performance. It combines the Precision and Recall to compute its score. It is calculated as,

$$F1 - Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (2.58)$$

AUC

Area under the ROC (Receiver Operating Characteristic) curve (AUC) measure is especially important for imbalanced datasets when evaluating classification algorithms [76]. The AUC metric determines the extent of the area below a ROC curve and is computed as follows:

$$AUC = \frac{1}{mn} \sum_{i=1}^m \sum_{j=1}^n 1_{p_i > p_j} \quad (2.59)$$

where the index i loops over the correctly predicted positive class samples, and j loops over the correctly predicted negative class samples, p_i , p_j are the predicted probabilities to the data sample i and j , respectively. Finally, $1_{p_i > p_j}$ returns 1 if and only if $p_i > p_j$, and 0 otherwise.

G-means

For imbalanced data, another metric is G-means. It takes the geometric mean of the accuracy computed for each class. This is given by,

$$\text{G-mean} = \sqrt{\frac{TP}{TP + FN} \times \frac{TN}{TN + FP}} \quad (2.60)$$

It is also important to have the number of training set sufficient for contenders to reflect their true capabilities. Likewise, it is important to have a fair and

sufficient testing set. In this case, 10-fold cross-validation is appropriate. The validation scheme divides the data into ten unique chunks of 90% training and 10% testing respectively. The intersection of any two testing sets is the empty set as they are disjoint.

Because parameters play a major role in performance, having a validation set is necessary. Algorithms are evaluated with different parameters to get the parameters that achieve highest performance in the validation set. This yields more reliable test results in the benchmark.

CHAPTER 3

RECURRENT EXTREME LEARNING MACHINES

3.1 Introduction

This chapter describes a new extreme learning machines (ELMs) algorithm that can learn spatio-temporal patterns. They extend single-hidden layer feedforward networks (SLFNs) by an additional layer, usually known as the context layer, that stores information about the previous time step. It receives values of certain features extracted from the previous time step - for example, values of the hidden or the output layer, which are combined with the input features of the next time step. This allows ELMs to develop an informative correlation between time steps, which is useful for predicting future behaviour that depends on time.

This chapter presents two types of context layers,

- hidden context layer; and

- output context layer

The hidden context layer receives values of the hidden layer of the previous time step, whereas the output context layer receives values of the output layer of the previous time step. These two layers are explained in detail in the following sections.

3.1.1 Recurrent-Hidden Extreme Learning Machines

Recurrent-hidden extreme learning machines (RH-ELM) are structured as ELMs with an additional hidden context layer.

This context layer receives values of the hidden neurons of the past time step. The outgoing weights of the context layer fully connects with the hidden layer, as shown in Figure 3.1.

That way the values in the hidden layer become the weighted summation of the input features and the context layer. This is written as,

$$H = (X \cdot W) + (W_c \cdot C) \quad (3.1)$$

where W_c represents the outgoing matrix of the context layer and C is the vector representing the values of the context layer.

Suppose $elm_{RH} = (W, b, W_c, H, \beta, C)$ represents an RH-ELM where the notations are those explained in Table 2.1. Then, the algorithm for training an RH-ELM is given as,

In Line 1, the input weights W , bias vector b , and the outgoing weights W_c of

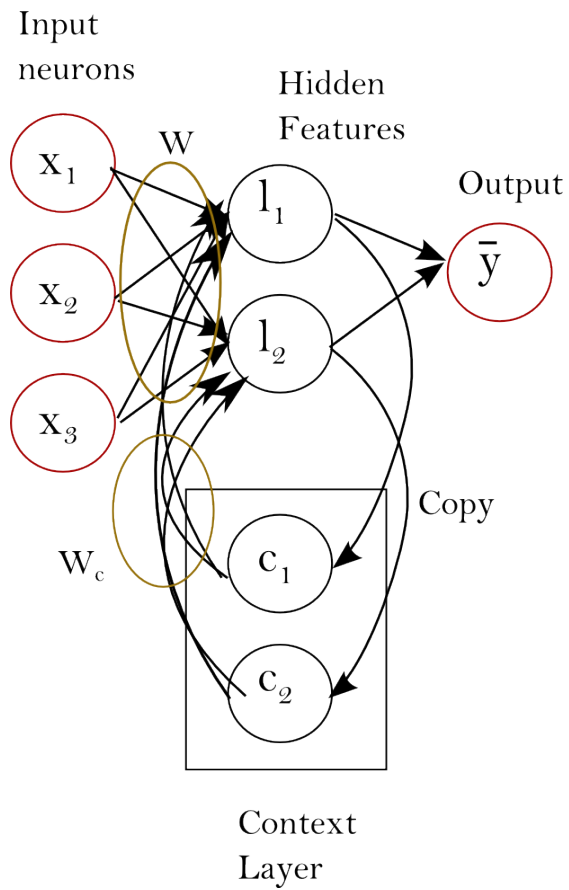


Figure 3.1: Recurrent-hidden extreme learning machines.

<p>Input: a tuple $elm_{RH} = (W, b, W_c, H, \beta, C)$, input matrix X and target vector T</p> <p>Output: a trained Recurrent Hidden neural Network</p> <ol style="list-style-type: none"> 1 $W, b, W_c \leftarrow$ Uniform Random Values; 2 $H \leftarrow g(\text{dot_product}(X, W) + b)$; 3 $C \leftarrow H[: -1]$; 4 $H \leftarrow g(\text{dot_product}(X, W) + \text{dot_product}(C, W_c) + b)$; 5 $\beta \leftarrow \text{regularized_least_square}(H, y)$; 6 return (W, b, W_c, H, β, C)
--

Figure 3.2: Training Recurrent Hidden ELM.

the context layer are assigned random initial values based on a uniform distribution in the range $[-1, 1]$. Lines 2 and 3 compute the hidden activations H and assign a subset of their values to C . Next in line 4, H is computed as the weighted summation of X and C . Finally, lines 5 and 6 solve for the hidden weights β using regularized least-square method and returns the new tuple defining a trained elm_{RH} .

3.1.2 Recurrent-Output Extreme Learning Machines

The second proposed algorithm, recurrent-output extreme learning machines (RO-ELM), has an output context layer that receives a copy of the output values of the previous time step. The neurons in the context layer have outgoing weights that fully connect with the output layer. Therefore, the output layer has incoming weights from the hidden layer and the context layer, as shown in Figure 3.3.

This network is more difficult to train than RH-ELM, since output values of the previous time step can only be determined by running the standard ELM algorithm first. Therefore, this algorithm must use sequential ELMs. It first

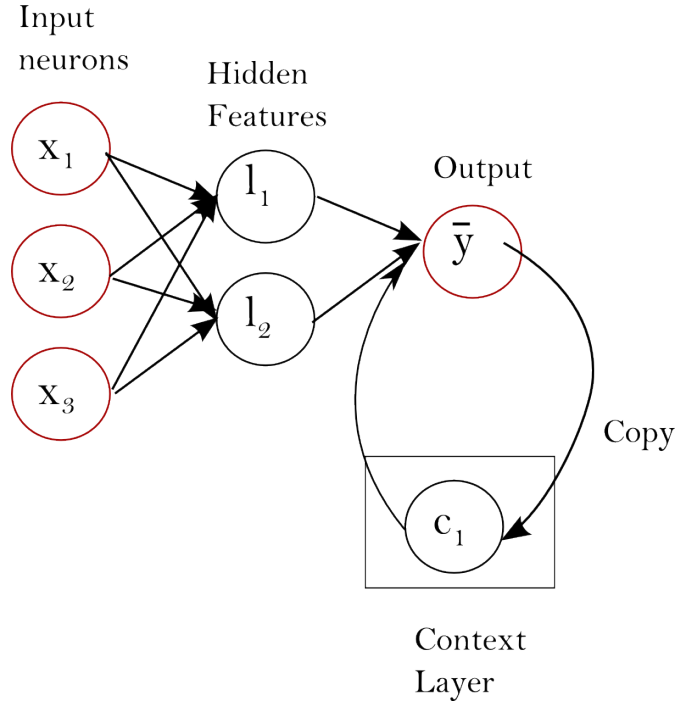


Figure 3.3: Recurrent-output extreme learning machines.

breaks X into batches. That way, the output values of the previous batches are copied to the output context layer C .

Suppose $elm_{RO} = (W, b, W_c, H, \beta, C)$ represents an RO-ELM where the notations are those explained in Table 2.1. Then, the algorithm for training an RO-ELM is given as,

The algorithm starts by splitting X and T into batches. At the first iteration (when $k = 0$), a standard *ELM* trains on X , and predicts the output of the first batch. For the remaining iterations, line 6 computes the hidden activations H and line 8 combines them with the output of the previous batch given as *last_output*. Later, using recursive least-squares as defined in section 2.5.2, β updates its values based on the features given by the hidden layer and the context layer. Once all the batches are traversed, the algorithm returns the trained

```

Input: a tuple  $elm_{RO} = (W, b, W_c, H, \beta, C)$ , a trained ELM network  $elm$ ,
          an input matrix  $X$  and a target vector  $T$ 
Output: a trained Recurrent Output ELM.
1  $X\_batches, T\_batches \leftarrow split(X), split(T)$ ;
2 for  $k$  in  $X\_batches.length$  do
3   if  $k = 0$  then
4      $last\_output = elm.predict(X\_batches[k])$ ;
5   else
6      $H \leftarrow g(dot\_product(X\_batches[i], W) + b)$ ;
7      $H \leftarrow combine\_columnwise(H, last\_output)$ ;
8      $\beta \leftarrow recursive\_least\_square(H, y\_batches[k], last\_output)$ ;
9      $last\_output = dot\_product(H, \beta)$ ;
10  end
11 end
12 return  $elm_{RO} = (W, b, W_c, H, \beta, C)$ 

```

Figure 3.4: Training Recurrent Output Neural Network

$elm_{RO} = (W, b, W_c, H, \beta, C)$.

Mathematically, the function of RO-ELM is written as,

$$H\beta = y \tag{3.2}$$

but β incorporates the outgoing weights of the hidden layer H_o as well as the context layer C_o , resulting in this matrix,

$$\beta = \begin{bmatrix} \beta_{old} \\ W_c \end{bmatrix}$$

The recursive least-square update can be explained as follows.

- First, divide the training dataset into batches;
- Train a traditional ELM on the whole data and predict the output for the

first batch. This will allow RO-ELM to use that output to train on the rest of the batches with spatio-temporal properties;

- Compute for the hidden layer H_1 , corresponding to the hidden activations of the second batch and the output of the last batch copied in the context layer C_0

$$H_1 = \begin{bmatrix} H_{subset} \\ C_0 \end{bmatrix}$$

- Compute the initial β_1 by solving for the following least-square problem; $\|H_1 \cdot \beta_1 - y_1\|$ where β_1 incorporates the outgoing weights from the hidden layer as well as the output context layer in Eq. 3.8. It follows that

$$\beta_1 = (H_1^T H_1)^{(-1)} \text{ and}$$

$$K_1 = H_1^T H_1.$$

- Finally, set k to 1.

Next is the recursive update of the variables, K , and β .

1. Compute H_{k+1} using the batch $k + 1$ in the same way as computing H_0 ;
2. Compute $K_{k+1} = K_k + H_{k+1}^T H_{k+1}$ and $\beta_{k+1} = \beta_k + K_{k+1}^{-1} H_{k+1}^T (y_{k+1} - H_{k+1} \beta_k)$, where y_{k+1} is the target output for batch $k + 1$;
3. increment k by one and return to step 1 of the recursion.

Figure 3.5 visualizes a spatio-temporal algorithm from another perspective..

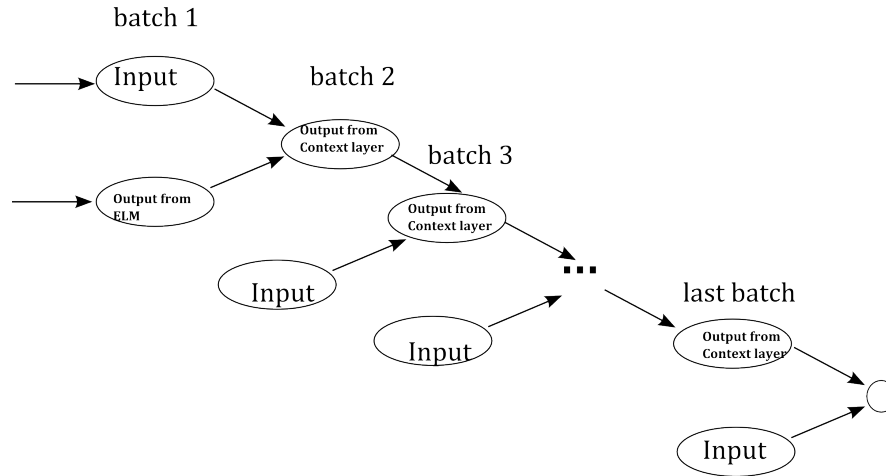


Figure 3.5: Visualizing Spatio-Temporal learning.

3.2 Experimental Results

3.2.1 Experimental Setup

The experiments were ran in a machine with 3.6 GHz quad-core CPU and 32 GB RAM operating a 64-bit Windows 7. The results are the average of 10 runs of 70-30 split cross-validation, with scores based on the mean square error.

For the benchmark, the following three algorithms were evaluated, 1. Extreme learning machines (ELMs) 2. Recurrent hidden ELMs (RH-ELMs) 3. Recurrent output ELMs (RO-ELMs)

3.2.2 Experimental Design

The experiment involves ten datasets and they are explained in Table 3.1

Figure 3.6 displays the sample values of the six chaos datasets with respect to time.

It is clear that although the values are “chaotic”, the time-steps are correlated and therefore there is a pattern that recurrent ELMs can take advantage of.

Table 3.1: Dataset statistics. S and F are the number of samples and features in the dataset, respectively.

Dataset	$S::F$	Description
Yahoo Stock Market	5296 ::2	stock values of the Yahoo! Finance stock market dating from 1991 to 2012.
Short-movement Stock Prices	11200 ::198	data representing features of various financial securities recorded at 5-minute intervals throughout a trading day.
1D Japanese Wind Data	20,000::1	recorded wind speeds at 5-minute intervals
4D Japanese Wind Data	20,000::4	recorded wind speeds from 4 directions: North, South, East, West at 5-minute intervals.
Chaos Data A	999::1	measurements on an 81.5-micron 14NH_3 cw (FIR) laser, pumped optically by the P(13) line of an N_2O laser via the vibrational $\text{aQ}(8,7)$ NH_3 transition.
Chaos Data B	16999::3	a multivariate data set recorded from a patient in the sleep laboratory of the Beth Israel Hospital in Boston, Massachusetts.
Chaos Data C	14999::3	tick-wise time record of a financial series.
Chaos Data D	49999::1	generated points by numerically integrating the equations of motion for a damped, driven particle.
Chaos Data E	27203::1	measurements of the light curve (time variation of the intensity) of the variable white dwarf star PG1159-035 during March 1989.
Chaos Data F	3823::4	measurements of four interacting degrees of freedom of the system.

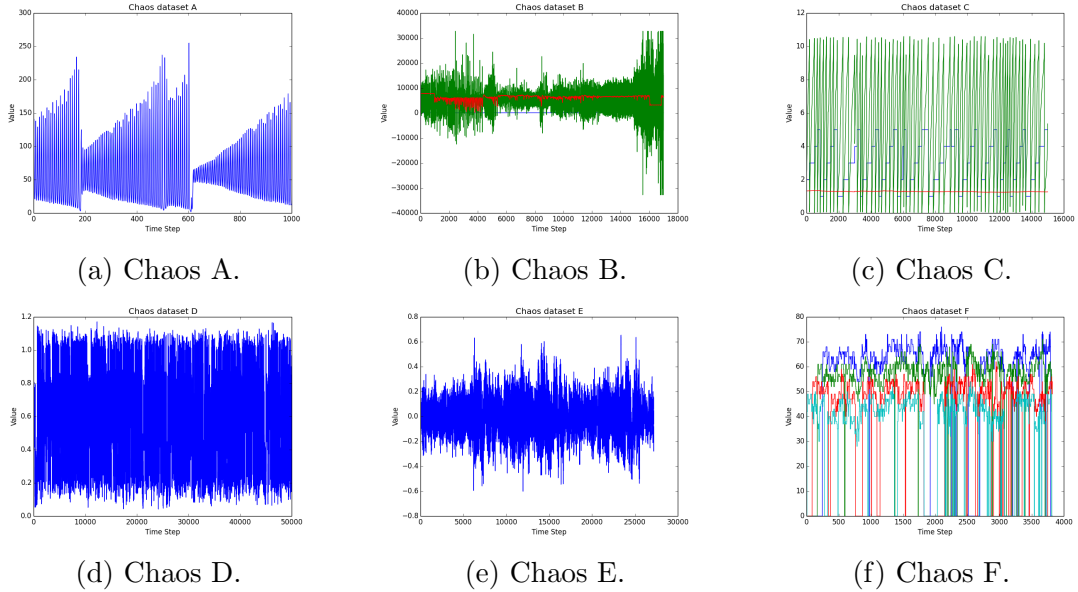


Figure 3.6: Illustrates the values of the chaos datasets.

3.2.3 Experimental Results

This section reports the results from applying ELM, RO-ELM, RH-ELM on the datasets described in Table 3.1.

Table 3.2 illustrates the tests scores of the algorithms on the yahoo dataset.

Algorithm	Mean Absolute Error
ELM	0.085
RO-ELM	0.082
RH-ELM	0.081

Table 3.2: Comparison between algorithms using the Mean-Absolute Error performance metric on the yahoo dataset.

For the yahoo dataset, it is clear that Recurrent ELM has performed better than ELMs. The reason for such, the context layer helped in extracting features representing the correlation between time steps, which helps in determining stock values.

Table 3.3 illustrates the results of the algorithms on the 1D Japanese Wind dataset.

Algorithm	Mean Absolute Error
ELM	7.29
RH-ELM	7.26
RO-ELM	6.37

Table 3.3: Comparison between algorithms using the Mean-Absolute Error performance metric against the 1D Wind dataset.

Recurrent ELMs outperformed ELM in the 1D Japanese Wind dataset. This is due to the fact that wind speeds are highly correlated with respect to time. Since ELMs does not consider such correlations, it had inferior performance to recurrent ELMs, especially RO-ELM.

Table 3.4 illustrates the results of the algorithms on the Short-movement Stock Prices dataset.

The Short-movement Stock Prices dataset contains ten features and therefore recurrent ELMs performed better than the regular ELMs by better predicting the output of the stock market. After all, having more features helps in extracting more features from the past time step.

Algorithm	Mean Absolute Error
ELM	1.6895
RH-ELM	1.478
RO-ELM	1.272

Table 3.4: Comparison between algorithms using the Mean-Square Error performance metric on the Short-movement Stock Prices dataset.

Next, Figure 3.7 reports the results of the algorithms on the 4-D japanese

Wind dataset against the number of hidden layers.

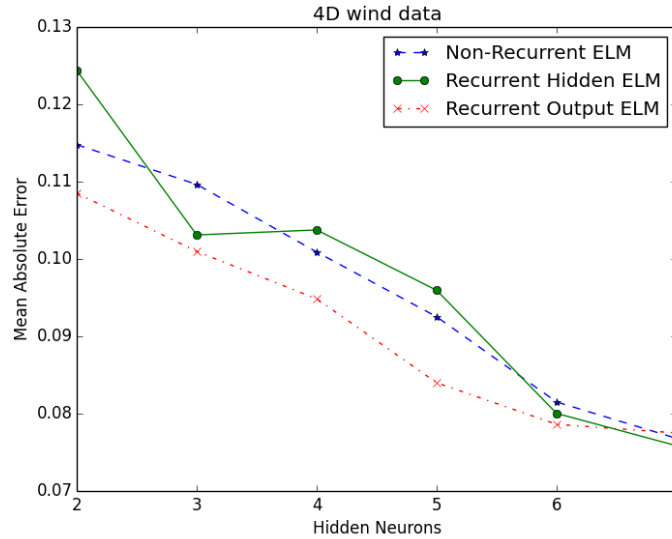


Figure 3.7: RH-ELM vs. RO-ELM vs. ELM on the 4-D japanese Wind dataset.

The reason why recurrent ELMs achieved a higher performance on the 4-D japanese Wind dataset than that on the 1-D japanese Wind dataset is because the number of features are larger 4-D japanese Wind dataset - it has 4 features rather than one. Therefore, with more hidden neurons, recurrent ELMs can capture more meaningful relationships between features of the past time step and those of the current time step.

The following six figures: Figure 3.8 to 3.13, report the results of applying the three algorithms with respect to the number of hidden neurons.

In most results (Dataset A, B, C, D), output-recurrent ELM achieved the best performance. This is expected as it uses least-square solutions to find the weights that combine previous outputs and current hidden activations H .

On the other hand, hidden-recurrent ELM did not always achieve good per-

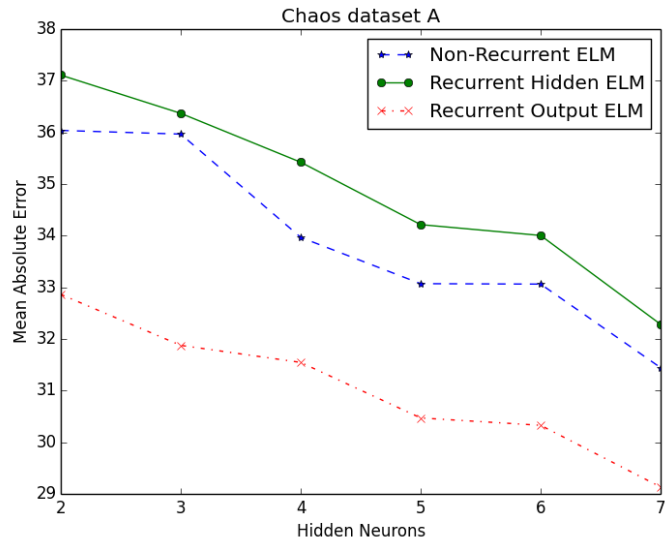


Figure 3.8: RH-ELM vs. RO-ELM vs. ELM on the Chaos dataset A.

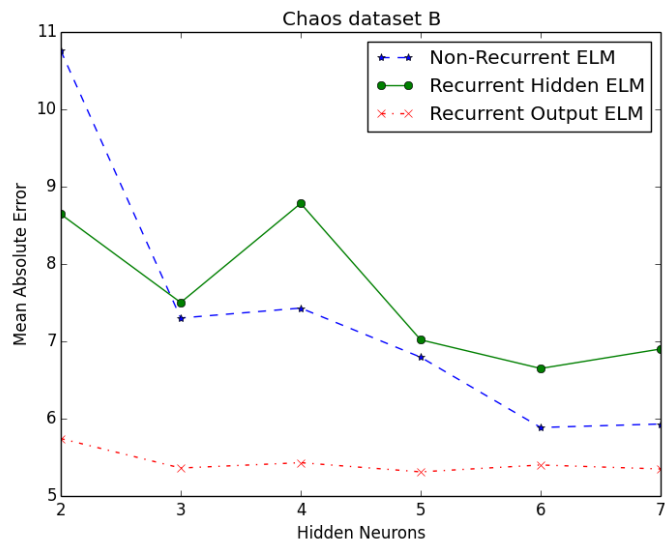


Figure 3.9: RH-ELM vs. RO-ELM vs. ELM on the Chaos dataset B.

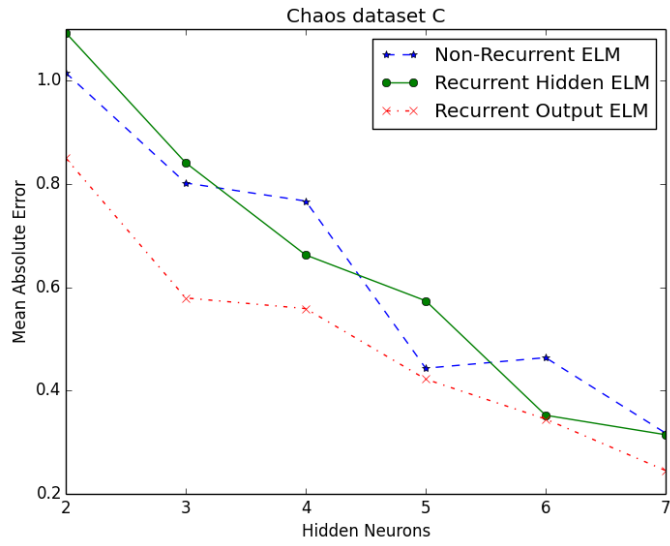


Figure 3.10: RH-ELM vs. RO-ELM vs. ELM on the Chaos dataset C.

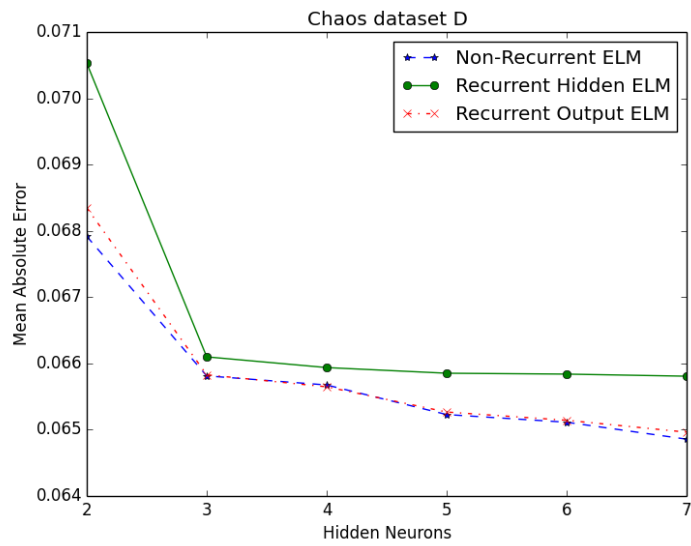


Figure 3.11: RH-ELM vs. RO-ELM vs. ELM on the Chaos dataset D.

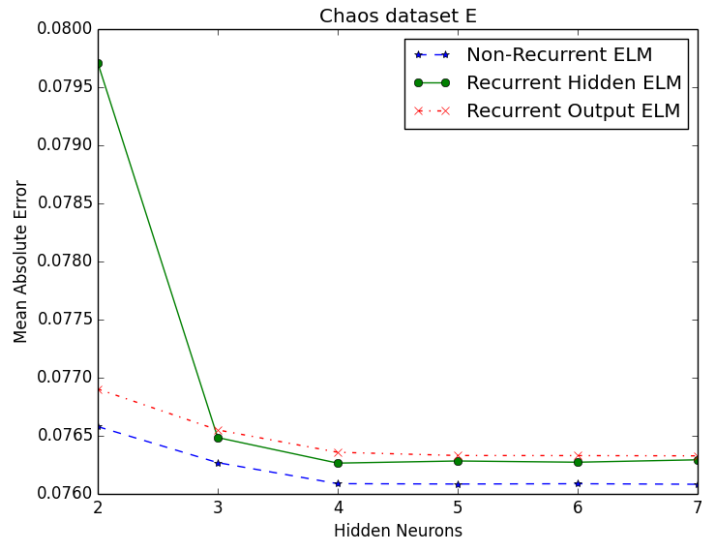


Figure 3.12: RH-ELM vs. RO-ELM vs. ELM on the Chaos dataset E.

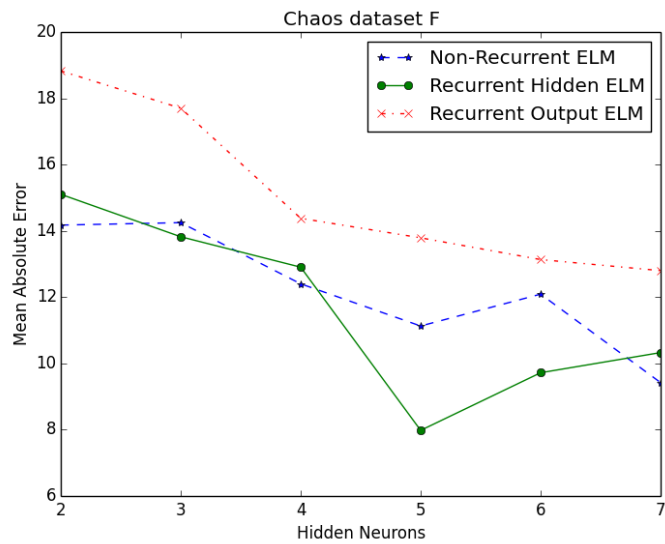


Figure 3.13: RH-ELM vs. RO-ELM vs. ELM on the Chaos dataset F.

formance, but it has outperformed non-recurrent ELMs in datasets B, C, and F. The reason is that hidden-recurrent ELM uses random weights to represent the relations between time-steps, which is not as efficient as output-recurrent ELM.

CHAPTER 4

EXTREME LEARNING MACHINE BASED AUTOENCODERS

4.1 Introduction

This chapter describes a novel implementation of sparse autoencoders (SAE) known as extreme learning machine based autoencoders (ELM-AE). Like SAE, the algorithm extracts new, hidden features from a given set of images. It can also train its weights in an unsupervised manner to help pre-train large artificial neural networks (ANNs). However, ELM-AE uses least-square solutions for training, which involves few matrix operations, and therefore makes the algorithm excel in both speed and efficiency.

Feature extraction is especially important for image, audio, and video process-

ing. Researchers have been working for decades to improve on feature extraction algorithms, such as principal component analysis, SIFT, and Gabor wavelet transform.

However, each of these algorithms require great expertise in the field from the user side in order to tune their parameters for the dataset at hand. In other words, without proper values of the parameters, the feature extractors could result in poor performance. SIFT is one complex algorithm that is difficult to implement and debug.

SAE gained great momentum as a reliable feature extractor, whose network structure is shown in Figure 4.1. Instead of manually tuning the algorithm to suit the dataset, SAE learns the structure of the data and the relationships between its features to extract more robust features that are competitive to hand-engineered features.

Backpropagation is one standard algorithm to train SAE. Starting from random weights, the algorithm optimizes them by reducing the error of input reconstruction, subject to a sparsity constraint such as the Kullback-Leibler divergence (KLD) term. The values developed in the hidden layer become the new features representing the input data.

Such algorithm is especially slow for large datasets because of the slow gradient descent involved in backpropagation. On the other hand, since ELM-AE uses least-squares to find these hidden features, it is a much faster feature extraction algorithm.

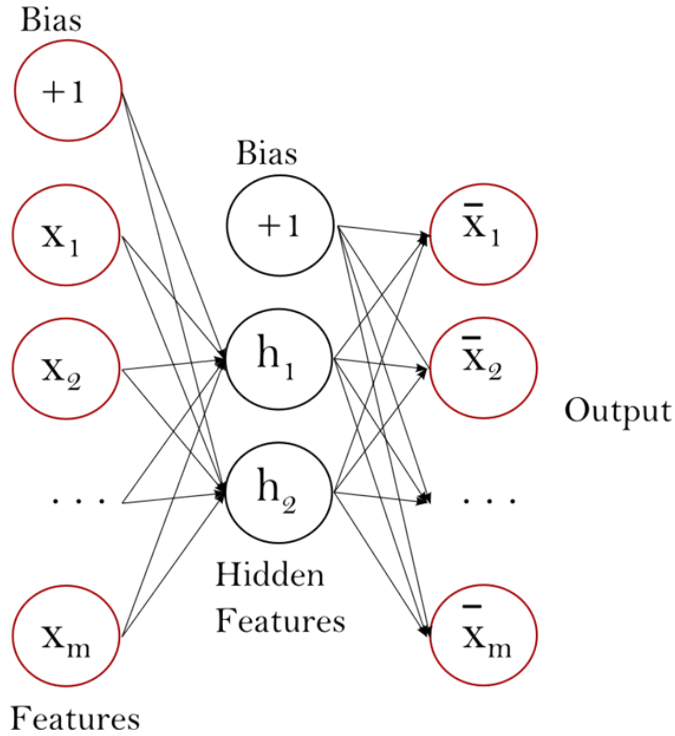


Figure 4.1: Sparse autoencoder network.

ELM-AE is described as follows. Given an input matrix $X \in \mathbb{R}^{n \times m}$, a bias vector $b \in \mathbb{R}^L$, and weight matrices $W \in \mathbb{R}^{m \times L}$ and $\beta \in \mathbb{R}^{L \times m}$. Consider a network containing m input neurons, L hidden neurons, and m output neurons. The reconstruction function is written as,

$$\bar{X} = g(X \cdot W + b) \cdot \beta \quad (4.1)$$

where $g() : \mathbb{R} \rightarrow \mathbb{R}$ is the logistic activation function, and \bar{X} is set to X .

The goal is to find β that solves equation 4.1. The hidden activations H of the hidden layer are computed as,

$$H = g(X \cdot W + b) \quad (4.2)$$

The following equation solves for β using regularization,

$$\beta = \left(\frac{I}{C} + H^T H\right)^{-1} H^T \bar{X} \quad (4.3)$$

where I is the identity matrix, and C is a constant that controls regularization.

Finally, the new features F are computed using this equations,

$$F = g(X \cdot \beta^T) \quad (4.4)$$

First, W is given small, random values. Next, zero-masking is applied so that around half of W contains zero entries. This would encourage the algorithm to learn sparse features, similar to the scheme of SAE. More precisely, the initial values of W range between -0.005 and 0.005 based on the uniform distribution. Each value in W is zero-masked with probability 50% based on the Bernoulli distribution. Next is to compute the initial hidden activations H given in eq. 4.2. After computing β as in eq. 4.3, evaluate eq. 4.4 to get the new features F .

However, in many cases, β is solved using regularized least-square solutions, as regularization counteracts the noise often present in datasets and therefore the solutions are more robust than otherwise. It computes β as follows,

Algorithm 1 represents the scheme of ELM-AE as it extracts new features from the input using least-square solutions.

Input: a tuple $elm = (W, b, H, \beta)$, a matrix X and target vector y

Output: a trained Sparse auto-encoder

- 1 $W, b \leftarrow$ Uniform Random Values;
- 2 $H \leftarrow g(\text{dot_product}(X, W) + b)$;
- 3 $\beta \leftarrow \text{regularized_least_square}(H, y)$;
- 4 *return* $\text{dot_product}(X, \beta^T)$

Algorithm 1: Returns hidden features

In line 1, W and b are given small random values based on the uniform distribution. Line 2 computes the hidden activations by taking the dot product of X and W followed by the addition of the bias vector and a non-linear activation function. β is obtained using the regularized least-square solutions given in eq. 4.3. Using β , new hidden activations can be extracted by taking the dot product between of input features \hat{X} and β^T .

While ELM-AE can extract new, robust features as explained in Algorithm 1, it can also be used to train large deep belief networks (DBNs). After training an ELM-AE, the extracted β is used as the weight matrix corresponding to a layer in a DBN. The weight matrix of each hidden layer in a DBN is assigned with the β values obtained from the corresponding trained ELM-AE. The weight matrix corresponding to the last layer of a DBN, however, is obtained using the least-squares approach. Figure 4.2 demonstrates this process.

Algorithm 2 illustrates an example of training a DBN of two hidden layers using two ELM-AEs.

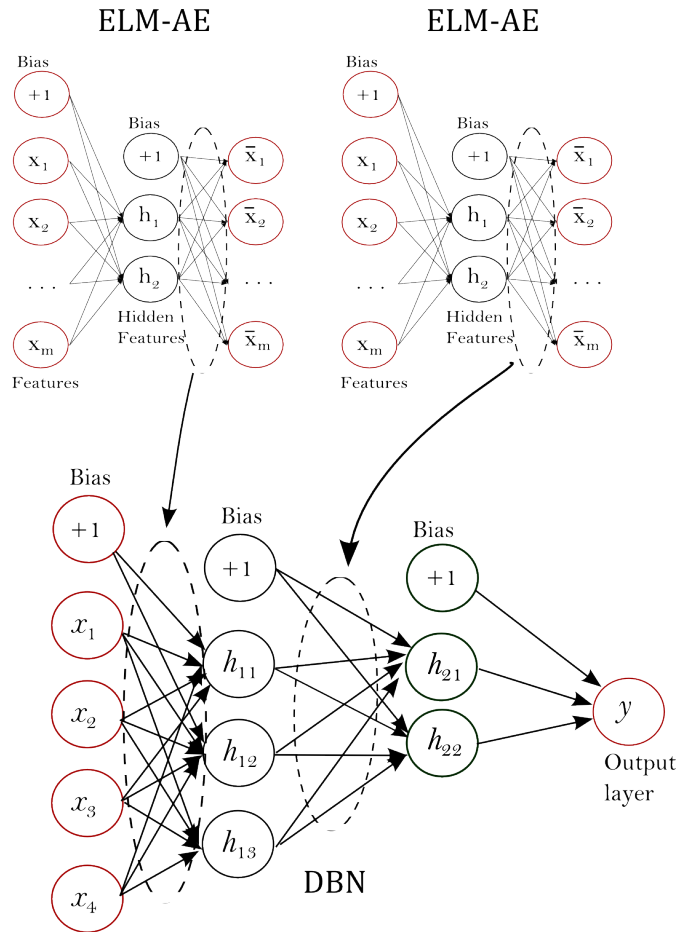


Figure 4.2: Training DBN using ELM-AE.

<p>Input: a tuple $DBN_{elm} = (W, b, H, \beta, \beta_{output})$, a matrix X and target vector T</p> <p>Output: a trained deep belief network of two hidden layers</p> <ol style="list-style-type: none"> 1 $\beta_1 \leftarrow$ get β by training ELM-AE on X; 2 $H \leftarrow g(\text{dot_product}(X, \beta_1) + b)$; 3 $\beta_2 \leftarrow$ get β by training ELM-AE on H; 4 $H_2 \leftarrow g(\text{dot_product}(H, \beta_2))$; 5 $\beta_{output} \leftarrow \text{regularized_least_square}(H_2, y)$; 6 return $DBN_{elm} = (W, b, \beta, \beta_{output})$
--

Algorithm 2: Train a deep belief network of two hidden layers.

Given a DBN represented as $DBN_{elm} = (W, b, H, \beta, \beta_{output})$ whose notations are described in Table 2.1. The algorithm, in line 1, trains an ELM-AE on X to get β and assigns β_1 to it. Line 2 obtains the hidden activations H of the first hidden layer. Line 3 trains another ELM-AE on H to get β to which β_2 is assigned. Next, in line 4, the algorithm computes the second hidden activations H_2 representing the second hidden layer. Using regularized least-squares, the algorithm solves for the outgoing weights β_{output} of the second hidden layer. The last line returns a trained deep belief network of two hidden layers containing the parameters $(W, b, \beta, \beta_{output})$.

Table 4.1: Dataset statistics.

Dataset	# Samples	Description
MNIST	500-8000	Arabic Digits from 0 to 9
AHDBase	500-8000	Hindi Digits from 0 to 9
MAHDBase	500-8000	Hindi Digits from 0 to 9

4.2 Experimental Results

4.2.1 Experimental Setup

The experiments were ran in a machine of 3.6 GHz quad-core CPU and 32 GB RAM operating a 64-bit Windows. For a fair, reliable assessment, the average of stratified 5-fold cross-validation is taken, with scores based on Accuracy.

For the benchmark, the following two algorithms are evaluated with the described settings (unless specified otherwise).

1. Support vector machines (SVM) with radial-basis kernel (RBF).
2. Extreme learning machines (ELMs)

4.2.2 Experimental Design

The algorithms were evaluated on 3 datasets: MNIST, AHDBase, and MAHDBase. Table 4.1 reports the statistics of these datasets. A sample of each dataset is shown in Figure 4.3

Using grid search on 2000 images set as the validation set from each dataset, the settings given in Table 4.2 were selected for the algorithms since they result in the best performance against that set.

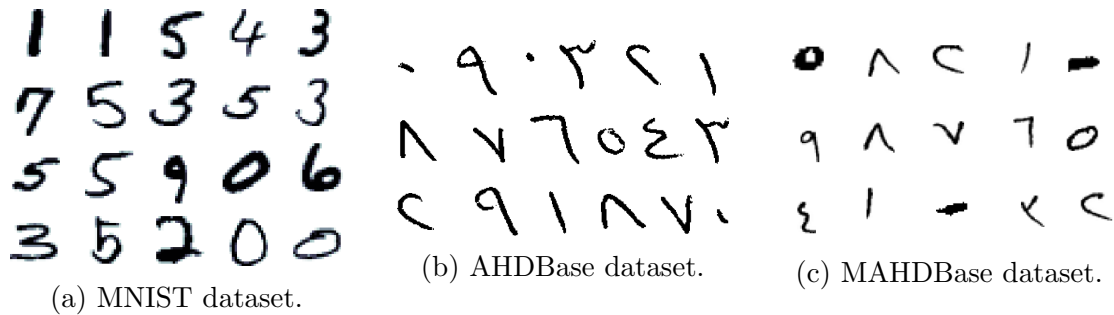


Figure 4.3: Samples of the three digit datasets.

Table 4.2: Algorithms' settings.

Algorithm	Settings
ELM	700 neurons
SVM	$C = 1.2$, RBF's spread = $\frac{1}{\#of\ features}$
ELM-AE	550 hidden neurons
SAE	150 hidden neurons, 200 iterations

4.2.3 Experimental Results

The objective of the experiment is three-fold:

- it shows how accuracy improves with ELM-AE features over raw pixels;
- it compares extracted features generated by ELM-AE with that of sparse autoencoders and their training time; and
- it compares accuracy and training time between deep belief network of two layers based on backpropagation and one that is based on extreme learning machines.

4.2.4 ELM-AE Features

This section shows that ELM-AE features improve performance over raw pixel features. Table 4.3 and 4.4 report these results on the MNIST dataset.

Table 4.3: Performance on the MNIST dataset for 500, 1K, and 2K samples.

Algorithm	500	1k	2k
SVM	0.68 ± 0.03	0.84 ± 0.013	0.87 ± 0.013
SVM trained on ELM-AE features	0.86 ± 0.016	0.91 ± 0.02	0.92 ± 0.009
ELM	0.6 ± 0.019	0.7 ± 0.042	0.84 ± 0.012
ELM trained on ELM-AE features	0.69 ± 0.052	0.76 ± 0.023	0.91 ± 0.018
ELM trained on SAE features	0.7 ± 0.05	0.58 ± 0.028	0.91 ± 0.008

Table 4.4: Performance on the MNIST dataset for 4k, 8k samples.

Algorithm	4k	8k
SVM	0.9 ± 0.009	0.92 ± 0.004
SVM trained on ELM-AE features	0.94 ± 0.007	0.95 ± 0.02
ELM	0.88 ± 0.012	0.9 ± 0.007
ELM trained on ELM-AE features	0.93 ± 0.008	0.94 ± 0.006
ELM trained on SAE features	0.93 ± 0.05	0.95 ± 0.004

In Tables 4.3 and 4.4, it is clear that ELM-AE had extracted features that greatly improved accuracy for SVM and ELM over using raw pixels as features.

The tables also show that that SAE features and ELM-AE features achieve similar results. However, SAE training time ranges between 29 to 267 seconds, whereas ELM-AE took 3 seconds as a maximum training time.

Tables 4.5 and 4.6 report the accuracies achieved by the algorithms on the AHDBase dataset. The results show that the smaller the dataset the larger the

Table 4.5: Performance on the AHDBase dataset for 500, 1K, and 2K samples.

Algorithm	500	1k	2k
SVM	0.93 ± 0.01	0.94 ± 0.021	0.95 ± 0.01
SVM trained on ELM-AE features	0.94 ± 0.01	0.95 ± 0.018	0.96 ± 0.007
ELM	0.87 ± 0.014	0.68 ± 0.033	0.95 ± 0.006
ELM trained on ELM-AE features	0.91 ± 0.024	0.77 ± 0.039	0.91 ± 0.018
ELM trained on SAE features	0.7 ± 0.05	0.58 ± 0.028	0.95 ± 0.004

Table 4.6: Performance on the AHDBase dataset for 4k, 8k samples.

Algorithm	4k	8k
SVM	0.96 ± 0.003	0.96 ± 0.005
SVM trained on ELM-AE features	0.97 ± 0.004	0.96 ± 0.006
ELM	0.96 ± 0.006	0.95 ± 0.006
ELM trained on ELM-AE features	0.96 ± 0.008	0.95 ± 0.007
ELM trained on SAE features	0.93 ± 0.05	0.95 ± 0.004

improvement made by the extracted ELM-AE features. Fortunately, when there is no improvement on larger datasets, the performance of using ELM-AE features equals at least the resulting performance of using raw pixels as input features.

Table 4.7: Performance on the MAHDBase dataset for 500, 1K, and 2K samples.

Algorithm	500	1k	2k
SVM	0.91 ± 0.029	0.92 ± 0.019	0.94 ± 0.009
SVM trained on ELM-AE features	0.94 ± 0.022	0.94 ± 0.015	0.95 ± 0.006
ELM	0.83 ± 0.018	0.71 ± 0.029	0.93 ± 0.02
ELM trained on ELM-AE features	0.87 ± 0.047	0.73 ± 0.035	0.94 ± 0.018
ELM trained on SAE features	0.7 ± 0.05	0.58 ± 0.028	0.95 ± 0.004

Tables 4.7 and 4.8 report the accuracies achieved by the algorithms on the MAHDBase dataset. Like the results achieved on AHDBase dataset, ELM-AE features result in less improvement as the dataset grows larger. But the perfor-

Table 4.8: Performance on the MAHDBase dataset for 4k, 8k samples.

Algorithm	4k	8k
SVM	0.95 ± 0.004	0.95 ± 0.005
SVM trained on ELM-AE features	0.95 ± 0.007	0.96 ± 0.002
ELM	0.95 ± 0.009	0.95 ± 0.004
ELM trained on ELM-AE features	0.95 ± 0.007	0.95 ± 0.007
ELM trained on SAE features	0.93 ± 0.05	0.96 ± 0.004

mance of ELM-AE features are at least as efficient as that of raw pixels.

The reason that ELM-AE extracts robust features is that it uses sparse weight randomization followed by regularized least-square. Pixels as individual features are not informative, since different images can be represented by the same set of pixels, if their locations are disregarded. However, algorithms like ELM-AE can extract structural relationships between these pixels which would consider their locations, and therefore provide a lot of information about the image. These features allow classifiers to better differentiate between different images.

4.2.5 ELM-AE feature weights

This section shows that the feature weights printed by ELM-AE and SAE are quite similar, while SAE take a much longer time to train.

The two algorithms were trained against 20,000 images of the MNIST, and AHDBase dataset. Both ELM-AE and SAE used 100 hidden neurons. Additionally, SAE was ran for 200 iterations using the stochastic gradient descent algorithm.

Figure 4.4 and 4.5 display the feature weights printed by ELM-AE and SAE,

respectively. Figure 4.4 illustrates the feature weights trained by ELM-AE. These feature weights represent the transpose of the matrix β , where each square in the image is a reshaped row vector in β .

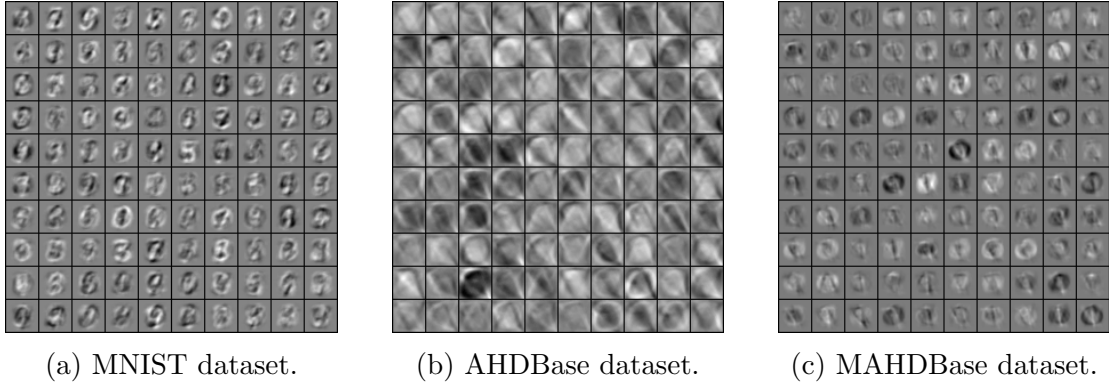


Figure 4.4: Displays feature weights β trained by ELM-AE.

Figure 4.5 displays the feature weights after training SAE on each dataset. The printed weights in the image represent the transpose of the matrix W_1 , the outgoing weights of the input layer.

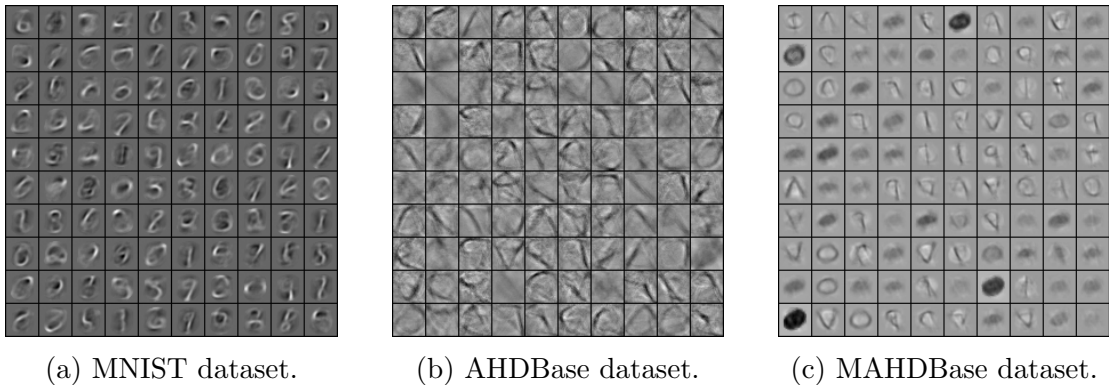


Figure 4.5: Displays feature weights W_1 trained by SAE.

Both ELM-AE and SAE produced clear features representing each of the datasets. For example, the digit strokes for the MNIST dataset reflect the subtle structure of digits of that dataset; meaning the trained weights transform the in-

put digit images into their corresponding expected structure while removing the background which could be noisy. However, ELM-AE took a much lesser time in training than SAE. Table 4.9 shows the average time taken of training ELM-AE and SAE on the datasets.

Table 4.9: Comparison between algorithms with respect to training time.

Algorithm	Training Time (in minutes)
Sparse Autoencoders	16.7 minutes
ELM-based Autoencoders	0.956

The reason why SAE take a much longer time to train is, it uses backpropagation which usually needs over hundred iterations to optimize its weights. ELM-AE, on the other hand, solves the linear system of equation only once, involving few matrix operations.

One caveat with ELM-AE, however, is the memory requirement. The whole matrix need to be in memory for training. But one can address this issue by implementing a sequential version of ELM-AE using the scheme described in section 2.5.2. It reduces the memory requirement by breaking the dataset into batches so that only one batch at a time must be in the memory for processing.

4.2.6 ELM-Based DBNs vs. DBNs

This section compares deep belief networks (DBN) with ELM-based DBNs (ELM-DBN) against 6000 images of the MNIST dataset. Using a validation set of 2000 images, the algorithm were assigned the settings that achieved the best performance for the two algorithms. The settings are,

- DBN: 200 iterations, 150 neurons in the first hidden layer, and 50 neurons in the second hidden layer
- ELM-DBN: 550 neurons in the first hidden layer, and 500 neurons in the second hidden layer

Table 4.8 shows the comparison between these two algorithms.

Table 4.10: Comparison between DBN and ELM-DBN.

Algorithm	Accuracy	Training Time (in seconds)
DBN	0.942	320
ELM-DBN	0.945	3

ELM-DBN has achieved better performance with much less training time. Regularized least-square solutions provide ELMs with a strong generalization power allowing it to excel in prediction. DBNs, on the other hand, have the tendency to overfit, meaning the parameters obtained from the validation dataset might not generalize well on the training dataset.

With a 3 second training time for such a large dataset, ELM-DBN allows researchers to test many different configurations and ideas in a short amount of time.

4.2.7 Performance results in the literature

This section provides an overview of algorithms given in the literature and their results on the MNIST dataset .

Table 4.11: Overview of algorithms in the literature.

Authors	Title	Accuracy
Hinton et al. [6]	A fast learning algorithm for deep belief nets	98.75%
Labusch et al. [43]	Simple method for high-performance digit recognition based on sparse coding	98.73%
Ruslan and Hinton [40]	Deep boltzmann machines	99.05%
Vincent et al. [42]	Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion	98.7%

CHAPTER 5

CONVOLUTIONAL EXTREME LEARNING MACHINES

5.1 Introduction

This chapter presents a novel implementation of the convolutional neural network (CNN), known as extreme learning machine based convolutional neural network (ELM-CNN). It uses least-square solutions to train a CNN regardless of how many layers it consists. The idea is to map the formulation of CNN into that of sparse autoencoders (SAE).

A convolutional layer consists of kernels that convolve with the input images to extract new, hidden features known as filters. A network with convolution avails of the images' 2D structure to extract meaningful, localized features. It preserves the spatial locality by sliding a fixed mask over an image since neighbouring pixels are highly correlated. Convolutional layers are more efficient than fully connected

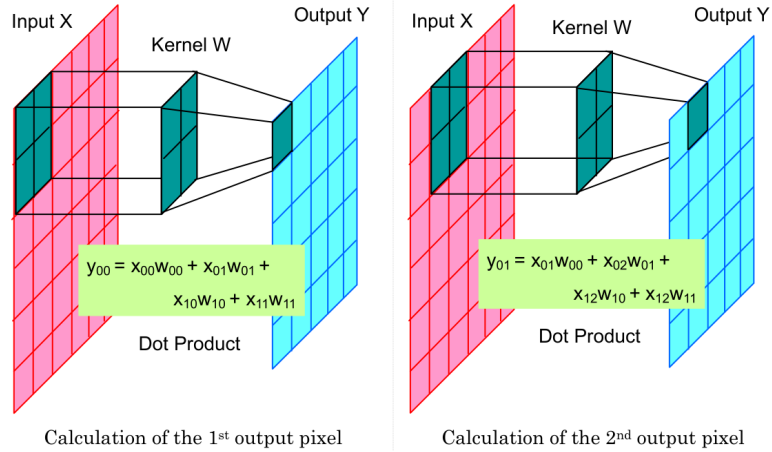


Figure 5.1: Convolution of a kernel with a 2D image subset.

layers as the latter might extract meaningless relationships between pixels residing far from each other and having no correlation. When the kernel slides over an image, the kernel's values multiply with the corresponding pixels in the image for each position, whose sum is given as the output. Figure 5.1 shows the operation of convolution between a kernel and a 2D image.

To train the layers in CNN, another network, known as ELM-based convolutional autoencoder (ELM-CA), is proposed. ELM-CA uses a network similar to that of SAE, in that the convolutional layer is nothing but a subtle representation of the SAE hidden layer. The objective function is to find the optimal kernel weights so that the input images can be reconstructed with minimum difference between the original and the reconstructed images. After training, the kernel weights can initialize a convolutional layer in a CNN. Such kernels are meant to carry efficient weights as they are able to reconstruct the input images. Figure 5.2 shows image reconstruction using ELM-CA.

Suppose an ELM-CA has a kernels for the input layer and the hidden layer,

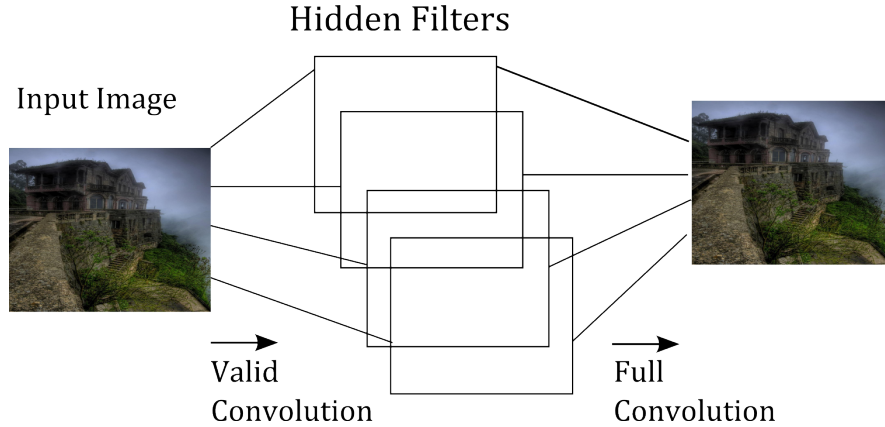


Figure 5.2: Image reconstruction using ELM-CA.

and b filters in the convolutional layer. The input and output layers represent the image X . The kernels in the input layer use 'valid' convolution so that the size of the filters is smaller than the input image. If the input image is of size $m \times m$ and the kernel is of $n \times n$, then 'valid' convolution outputs a filter size of $(m - n + 1) \times (m - n + 1)$. A smaller representation is beneficial in that it usually holds more subtle and concise features than the original representation.

Each kernel k_{input_i} in the input layer convolves with X on which a bias vector b is added, producing a 2D filter f_j . This can be written as,

$$f_j = g(X * k_i + b) \quad (5.1)$$

where $g(\cdot) : R \rightarrow R$ is the logistic activation function.

To reconstruct the input images, the set of i kernels in the hidden layer convolve with the filters using 'full' convolution. Given an $m \times m$ kernel and an $n \times n$ filter, 'full' convolution yields an $(m + n - 1) \times (m + n - 1)$ matrix, which equals the size of the input image.

Next, the algorithm takes the sum of convolutions between a kernel k_{hidden_i} in the hidden layer and each filter f_j as follows,

$$Y = \sum_{j=1}^b f_j * k_{hidden_i} \quad (5.2)$$

Using backpropagation, the kernel weights are updated until eq. 5.2 produces a correct reconstruction of the input image. However, backpropagation is both time consuming and complicated to implement. This is because, in addition to the slow stochastic gradient descent, computing equations 5.1 and 5.2 involves many loops, while finding the objective function derivatives is involved and difficult to debug.

ELM-CA, on the other hand, has a simple implementation scheme and can find these kernel weights very quickly. The idea is to represent the convolutional layer in SAE as a fully connected layer. In other words, instead of using equations 5.1 and 5.2 as the functions for convolutional reconstruction, ELM-CA uses the simple equations of 4.1 - 4.3. ELM-CA achieves this formulation by reshaping the input and kernel matrices to have a dot product operation equivalent to the original convolutional operation. The new representation of eq. 5.1 can therefore be given as,

$$H = A \cdot B \quad (5.3)$$

where H , A , and B are the new representations of $f_j|j = 0, 1, \dots, b$, X , and $k_i|i = 0, 1, \dots, a$, respectively. For each sub-image in X that overlaps with the kernel in convolution, there is a vector in A representing that sub-image. More

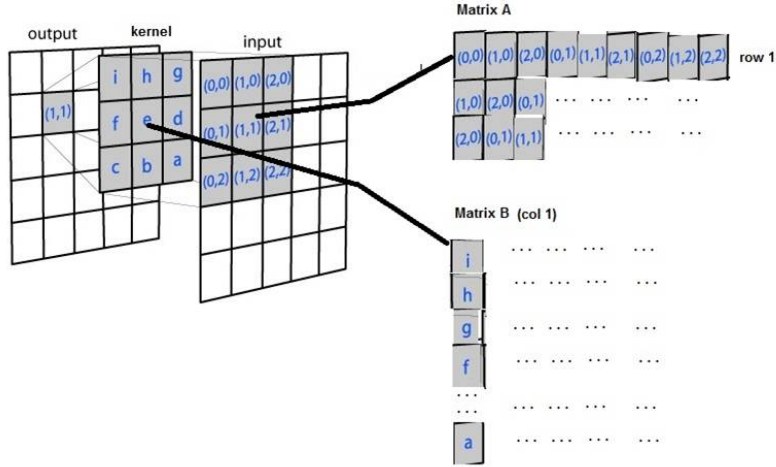


Figure 5.3: Dot product representation.

formally, the i th row in A is the flattened version of the sub-image in X sliced in the dimensions x -indices $i\%n : i\%n + m$ and y -indices $i/n : i/n + m$. Clearly, A has $m * m$ columns and $(n - m + 1)x(n - m + 1)$ rows. Similarly, B is the matrix whose i th column is the flattened k th kernel. Figure 5.3 shows an example representation of the these matrices.

$$H\beta = X \tag{5.4}$$

where β represent the set of kernels $k_{hidden_i} | i = 0, 1, \dots, a$, meant to reconstruct the input image. β can easily be solved using the regularized least-squares as in eq. 4.3. Finally, the new hidden filters are computed using the following equation,

$$H = A \cdot \beta^T \tag{5.5}$$

Running the algorithm in Figure 5.4 trains an ELM-CA and returns the hidden

filters as features.

<p>Input: a tuple $elm_{CA} = (W, b, H, \beta)$, a matrix X, k filters, and a target vector y</p> <p>Output: filters</p> <ol style="list-style-type: none">1 $kernels_inputs, b =$ Uniform Random Values between -1 and 1;2 $A \leftarrow$ Stack each flattened window in X as rows;3 $B \leftarrow$ Stack each flattened filter in $filters$ as columns;4 $H \leftarrow g(dot_product(A, B))$;5 $\beta \leftarrow regularized_least_square(H, A)$;6 $features \leftarrow dot_product(A, \beta^T)$;7 return $features.reshape_to_original_size()$;

Figure 5.4: Returns the hidden filters from training ELM-CA as features.

Line 1 assigns random values to the kernels and bias vector based on a uniform distribution between -1 and 1. Line 2 and 3 reshape X and $kernels_inputs$ to get A and B that allow for the "dot_product" representation of convolution. Lines 4 and 5 obtain the hidden filters followed by the solutions for β using regularized least-squares. Finally, the last line reshapes the extracted filters to their original 2D shape.

The algorithm in Figure 5.4 can be used to train CNNs consisting of a convolutional layer followed by a fully connected layer - like the CNN shown in Figure 5.5.

Consider an ELM-CNN consisting of W_1 that represents the kernel weights of the input layer and W_2 that represents the fully connected weights of the hidden layer. ELM-CNN assigns W_1 to $beta$, where $beta$ is the matrix obtained from training an ELM-CA on the input data. Next, ELM-CNN finds W_2 using the straightforward least-square solutions given in eq. 4.3.

The algorithm in Figure 5.6 describes the steps for training an ELM-CNN.

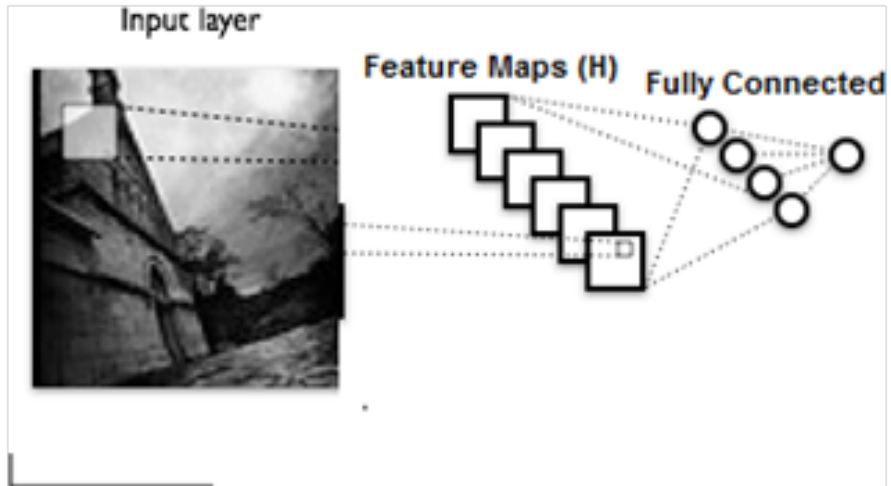


Figure 5.5: A CNN of one convolutional layer followed by a fully connected layer.

<p>Input: a tuple $elm_{CNN} = (W_1, b, H, W_2)$, $CA_{ELM} = (W, \beta)$ a matrix X and target vector y</p> <p>Output: a trained ELM-CNN</p> <ol style="list-style-type: none"> 1 Train ELM_CA on X; 2 $W_1 = ELM_CA.\beta^T$; 3 $H = g(W_1 \cdot X + b)$; 4 $W_2 \leftarrow regularized_least_square(H, y)$; 5 return (W_1, b, H, W_2)

Figure 5.6: Trains an ELM-CNN consisting of one convolutional layer followed by a fully connected layer.

Lines 1 and 2 trains an ELM_CA on the input data X and the obtained β is imputed to W_1 . Next, Lines 3 to 5 solve for W_2 using regularized least-square solutions, and then return the trained ELM-CNN, which can then be used for prediction.

Table 5.1: Dataset statistics.

Dataset	# Samples	Description
MNIST	500-8000	Arabic Digits from 0 to 9
AHDBase	500-8000	Hindi Digits from 0 to 9
MAHDBase	500-8000	Hindi Digits from 0 to 9

5.2 Experimental Results

5.2.1 Experimental Setup

The experiments were ran in a machine with 3.6 GHz quad-core CPU and 32 GB RAM operating a 64-bit Windows 7. For a fair, reliable assessment, the average of stratified 5-fold cross-validation is taken, with scores based on Accuracy.

The benchmarks involves the following four algorithms,

1. Support vector machines (SVM) with radial-basis function (RBF).
2. Extreme learning machines (ELMs)
3. ELM-based convolutional neural networks (ELM-CNN) with one convolutional Layer, followed by one fully connected layer

5.2.2 Experimental Design

The algorithms were evaluated against 3 datasets: MNIST, AHDBase, and MAHDBase.

Table 5.1 reports the statistics of these datasets.

To find good parameters, of the MNIST dataset, 8000 images were set as

training set and 2000 as the validation set. Using grid search for finding the parameters achieving the best accuracy for the validation set, SVM optimized its 'C' parameter which controls overfitting; Extreme Learning Machines optimized its number of hidden neurons; and ELM-CNN optimized its number of filters and kernel size. The parameters are given in Table 5.2

Table 5.2: Comparison between algorithms using the accuracy performance metric.

Algorithm	MNIST	AHDBase	MAHDBase
SVM	C=2	C=1.5	C=1.5
ELM	700 neurons	700 neurons	700 neurons
ELM-CNN	20 filters	120 filters	120 filters

5.2.3 Experimental Results

This section reports the performance of SVM, ELMs, and ELM-CNN on the three datasets with respect to accuracy.

For the first experiment, there are three tables reporting the accuracy and standard deviation of training the algorithms on the datasets with different sample sizes.

Table 5.3 and 5.4 report the algorithms' performance against the the MNIST dataset with increasing sample size.

ELM-CNN outperformed other algorithms for each dataset size, mainly because of its ability to preserve the spatial locality by extracting correlations between close pixels. SVM and ELM achieved close performance since they both have similar capabilities in constructing the non-linear function for prediction.

Table 5.3: Performance on the MNIST dataset for 500, 1K, and 2K samples.

Algorithm	500	1k	2k
SVM	0.74 ± 0.05	0.82 ± 0.03	0.87 ± 0.013
ELM	0.65 ± 0.043	0.66 ± 0.016	0.85 ± 0.009
ELM-CNN	0.9 ± 0.03	0.93 ± 0.01	0.95 ± 0.007

Table 5.4: Performance on the MNIST dataset for 4K and 8K.

Algorithm	4k	8k
SVM	0.9 ± 0.02	0.88 ± 0.013
ELM	0.91 ± 0.006	0.89 ± 0.007
ELM-CNN	0.96 ± 0.004	0.97 ± 0.03

The standard deviations are all very small to be considered significant, meaning the results are stable and safely represent their performance.

Table 5.5 and 5.6 report the algorithms' performance against the the MAHD dataset with increasing sample size.

Table 5.5: Performance on the MAHD dataset for 500, 1K, and 2K samples.

Algorithm	500	1k	2k
SVM	0.9 ± 0.034	0.91 ± 0.025	0.93 ± 0.01
ELM	0.77 ± 0.049	0.79 ± 0.022	0.94 ± 0.014
ELM-CNN	0.89 ± 0.039	0.94 ± 0.047	0.95 ± 0.026

Table 5.6: Performance on the MAHD dataset for 4K and 8K.

Algorithm	4k	8k
SVM	0.95 ± 0.004	0.95 ± 0.004
ELM	0.95 ± 0.008	0.95 ± 0.008
ELM-CNN	0.97 ± 0.013	0.97 ± 0.012

For MAHDBase, ELM-CNN performed worse than SVM in the 500 sample

subset. An explanation of this is that ELM-CNN is expected to overfit on small datasets, as it tends to construct highly complex functions. However, with larger sample sizes ELM-CNN consistently outperformed other algorithms with a more stable standard deviation.

Table 5.7: Performance on the AHDBase dataset for 500, 1K, and 2K samples.

Algorithm	500	1k	2k
SVM	0.83 ± 0.034	0.87 ± 0.025	0.89 ± 0.01
ELM	0.61 ± 0.037	0.61 ± 0.019	0.87 ± 0.013
ELM-CNN	0.88 ± 0.066	0.91 ± 0.038	0.94 ± 0.024

Table 5.8: Performance on the AHDBase dataset for 4K and 8K.

Algorithm	4k	8k
Support Vector Machines	0.87 ± 0.013	0.86 ± 0.011
Extreme Learning Machines (ELM)	0.9 ± 0.008	0.9 ± 0.005
ELM-CNN	0.97 ± 0.009	0.96 ± 0.009

Table 5.7 and 5.8 report the algorithms’ performance against the the AHD dataset with increasing sample size.

For AHDBase, ELM-CNN achieved the best performance with least standard deviation on the 500 sample subset. Further, with larger sample size datasets, ELM-CNN consistently outperformed other algorithms.

The time taken to train ELM-CNN was 164 seconds maximum.

It is clear that ELM-CNN is, on average, achieving the least error in all three datasets. The main reason is that convolution captures important information of 2D structures like images. The sliding window convolving with the image captures

the relationship between neighboring pixels in images which prove informative. After all, pixels close to one another are strongly correlated.

Another reason why ELM-CNN is successful is because the convolutional autoencoders based ELM used regularized least-squares for finding the weights of the decoding filters, extracting important features.

One observation is that the training time bottleneck of ELM-CNN is in computing the Moore-penrose pseudo inverse. Interestingly, Courrieu [68] proposed a fast method that computes Moore-penrose pseudo inverse of a matrix, which is five times faster than the method used in the experiments.

CHAPTER 6

CONCLUSION

This thesis presents three algorithms based on extreme learning machines (ELMs): recurrent ELMs (R-ELM), ELM based autoencoders (ELM-CA), and ELM based convolutional neural networks (ELM-CNN).

Developed for time-series datasets, R-ELM has an additional layer over ELMs, known as the context layer. This layer takes features of the past time step in order to improve the prediction accuracy. After all, features in each time step are correlated with those in the previous time steps. R-ELM not only uses past information for prediction, but also trains very quickly since it uses least-square solutions. Unlike backpropagation, least-square is known to be fast and efficient as it only involves few matrix operations for optimization.

One observation is that R-ELMs do not perform well with spatio-temporal datasets represented by only few features. This limitation is attributed to the fact that having a hidden layer would always cause overfitting on a dataset containing few features. However, for datasets with more than two features, R-ELMs

consistently outperform regular ELM.

Next, ELM-CA is proposed as a fast and efficient feature extractor. ELM-CA uses least-square solutions to extract new structural features from the input data. The idea is to randomize the input weights and use least-squares to solve the outgoing weights of the hidden layer. Empirically, it was shown that these weights are almost as efficient as that of sparse autoencoders (SAE) that trains using backpropagation. However, ELM-CA is a much faster learning algorithm than SAE.

The experimental results have shown that features extracted by ELM-AE has significantly improved regular ELM's and SVM's scores in classification. It was also shown that compared to deep belief networks (DBNs) of two layers, a similar network trained using ELM-AE achieved a better classification performance than a DBN trained using backpropagation.

Finally, ELM-CNN was proposed to train large convolutional networks. First, an ELM based convolutional autoencoder (ELM-CA) trains each convolutional layer in ELM-CNN. Training ELM-CA is simple, in that it maps the convolution formulation given in sparse convolutional autoencoder to an equivalent dot product formulation. ELM-CA can then be solved using least-square solutions using the same scheme as in ELM-AE. However, instead of learning fully connected weights, ELM-CA learns the kernel weights that can reconstruct the input data. In the experimental results, ELM-CNN achieved better accuracy than other state-of-the-art algorithms including support vector machines (SVMs) and ELMs.

On investigating the running time of ELM-CNN, the time bottleneck lies in computing the pseudo-inverse involved in one of the matrix operations. However, fast algorithms for computing the pseudo-inverse are available in the literature.

These three algorithms show that ELMs have a great promise in training large DBNs for huge datasets in a little amount of time. This allows researchers not only to carry out many different ideas and configurations - as training time is extremely low, but also find new perspectives that improves the algorithms in neural networks.

These algorithms have great potential for future work. R-ELMs, for example, can make use of two context layers in one network: the hidden and the output context layers, to establish better time step correlations. ELM-AE and ELM-CA could avail of sequential ELMs so they could learn features in real time while mitigating memory requirements. While ELM-AE is meant for gray-scale images, using complex functions like quaternions, ELM-AE would be able to extract features from color images. As such, features extracted are likely to be more meaningful than if they were extracted from gray-scale images. Similarly, ELM-CA could make use of quaternions to get better features for color images. These are interesting possibilities that are worth exploring.

REFERENCES

- [1] R. P. Lippmann, “An introduction to computing with neural nets,” *ASSP Magazine, IEEE*, vol. 4, no. 2, pp. 4–22, 1987.
- [2] F.-C. Chen, “Back-propagation neural networks for nonlinear self-tuning adaptive control,” *Control Systems Magazine, IEEE*, vol. 10, no. 3, pp. 44–48, 1990.
- [3] J. P. Bigus, *Data mining with neural networks: solving business problems from application development to decision support*. McGraw-Hill, Inc., 1996.
- [4] K. O. Stanley and R. Miikkulainen, “Efficient reinforcement learning through evolving neural network topologies,” *Network (Phenotype)*, vol. 1, no. 2, p. 3, 1996.
- [5] D. Ciresan, U. Meier, and J. Schmidhuber, “Multi-column deep neural networks for image classification,” in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3642–3649.
- [6] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

- [7] D. L. Ly, V. Paprotski, and D. Yen, “Neural networks on gpus: Restricted boltzmann machines,” see <http://www.eecg.toronto.edu/~moshovos/CUDA08/doku.php>, 2008.
- [8] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, “Extreme learning machine: theory and applications,” *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.
- [9] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain.” *Psychological review*, vol. 65, no. 6, p. 386, 1958.
- [10] S. J. Russell and P. Norvig, “Artificial intelligence: a modern approach (international edition),” 2002.
- [11] M. A. Hearst, S. Dumais, E. Osman, J. Platt, and B. Scholkopf, “Support vector machines,” *Intelligent Systems and their Applications, IEEE*, vol. 13, no. 4, pp. 18–28, 1998.
- [12] B. Schölkopf and A. J. Smola, *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT press, 2002.
- [13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.
- [14] M. Riedmiller and H. Braun, “A direct adaptive method for faster backpropagation learning: The rprop algorithm,” in *Neural Networks, 1993., IEEE International Conference on*. IEEE, 1993, pp. 586–591.

- [15] B. Scholkopf, “The kernel trick for distances,” *Advances in neural information processing systems*, pp. 301–307, 2001.
- [16] O. Chapelle, “Training a support vector machine in the primal,” *Neural Computation*, vol. 19, no. 5, pp. 1155–1178, 2007.
- [17] N. Morgan, D. Ellis, E. Fosler-Lussier, A. Janin, and B. Kingsbury, “Reducing errors by increasing the error rate: Mlp acoustic modeling for broadcast news transcription,” in *Broadcast News Workshop’99 Proceedings*. Morgan Kaufmann Pub, 1999, p. 167.
- [18] G. K. Venayagamoorthy, “Teaching neural networks concepts and their learning techniques,” in *39th ASEE Midwest Section Meeting*, 2004.
- [19] D. Scherer, H. Schulz, and S. Behnke, “Accelerating large-scale convolutional neural networks with parallel graphics multiprocessors,” in *Artificial Neural Networks–ICANN 2010*. Springer, 2010, pp. 82–91.
- [20] E. D. Sontag and H. J. Sussmann, “Backpropagation can give rise to spurious local minima even for networks without hidden layers,” *Complex Systems*, vol. 3, no. 1, pp. 91–106, 1989.
- [21] N. Le Roux and Y. Bengio, “Representational power of restricted boltzmann machines and deep belief networks,” *Neural Computation*, vol. 20, no. 6, pp. 1631–1649, 2008.
- [22] D. W. Hosmer Jr and S. Lemeshow, *Applied logistic regression*. John Wiley & Sons, 2004.

- [23] M. I. Jordan *et al.*, “Why the logistic function? a tutorial discussion on probabilities and neural networks,” 1995.
- [24] B. Karlik and A. V. Olgac, “Performance analysis of various activation functions in generalized mlp architectures of neural networks,” *International Journal of Artificial Intelligence and Expert Systems*, vol. 1, no. 4, pp. 111–122, 2010.
- [25] S. B. Kotsiantis, “Supervised machine learning: a review of classification techniques.” *Informatica (03505596)*, vol. 31, no. 3, 2007.
- [26] W. James and C. Stein, “Estimation with quadratic loss,” in *Proceedings of the fourth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 1961, 1961, pp. 361–379.
- [27] J. W. Miller, R. Goodman, and P. Smyth, “On loss functions which minimize to conditional expected values and posterior probabilities,” *Information Theory, IEEE Transactions on*, vol. 39, no. 4, pp. 1404–1408, 1993.
- [28] S. Suresh, N. Sundararajan, and P. Saratchandran, “Risk-sensitive loss functions for sparse multi-category classification problems,” *Information Sciences*, vol. 178, no. 12, pp. 2621–2638, 2008.
- [29] H. Yan, Y. Jiang, J. Zheng, C. Peng, and Q. Li, “A multilayer perceptron-based medical decision support system for heart disease diagnosis,” *Expert Systems with Applications*, vol. 30, no. 2, pp. 272–281, 2006.

- [30] M. K. Bhowmik, D. Bhattacharjee, M. Nasipuri, D. K. Basu, and M. Kundu, "Classification of polar-thermal eigenfaces using multilayer perceptron for human face recognition," in *Industrial and Information Systems, 2008. ICIIS 2008. IEEE Region 10 and the Third international Conference on*. IEEE, 2008, pp. 1–6.
- [31] P. Pandey and S. Barai, "Multilayer perceptron in damage detection of bridge structures," *Computers & Structures*, vol. 54, no. 4, pp. 597–608, 1995.
- [32] S. L. Phung, A. Bouzerdoum, and D. Chai Sr, "Skin segmentation using color pixel classification: analysis and comparison," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 1, pp. 148–154, 2005.
- [33] D. Pham and X. Liu, "Training of elman networks and dynamic system modelling," *International Journal of Systems Science*, vol. 27, no. 2, pp. 221–226, 1996.
- [34] A. Ng, "Sparse autoencoder," *CS294A Lecture notes*, p. 72, 2011.
- [35] R. N. Bracewell, "Fourier transform and its applications," 1980.
- [36] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *Computers, IEEE Transactions on*, vol. 100, no. 1, pp. 90–93, 1974.
- [37] M. Shensa, "The discrete wavelet transform: wedding the a trous and mallat algorithms," *Signal Processing, IEEE Transactions on*, vol. 40, no. 10, pp. 2464–2482, 1992.

- [38] Y. Wong, M. T. Harandi, C. Sanderson, and B. C. Lovell, “On robust biometric identity verification via sparse encoding of faces: Holistic vs local approaches,” in *Neural Networks (IJCNN), The 2012 International Joint Conference on*. IEEE, 2012, pp. 1–8.
- [39] H. Wersing and E. Körner, “Learning optimized features for hierarchical models of invariant object recognition,” *Neural computation*, vol. 15, no. 7, pp. 1559–1588, 2003.
- [40] R. Salakhutdinov and G. E. Hinton, “Deep boltzmann machines,” in *International Conference on Artificial Intelligence and Statistics*, 2009, pp. 448–455.
- [41] Y. Sun, X. Wang, X. Tang *et al.*, “Hybrid deep learning for face verification.” ICCV, 2013.
- [42] 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,” *The Journal of Machine Learning Research*, vol. 9999, pp. 3371–3408, 2010.
- [43] K. Labusch, E. Barth, and T. Martinetz, “Simple method for high-performance digit recognition based on sparse coding,” *Neural Networks, IEEE Transactions on*, vol. 19, no. 11, pp. 1985–1989, 2008.
- [44] R. Mazumder, J. H. Friedman, and T. Hastie, “Sparsenet: Coordinate descent with nonconvex penalties,” *Journal of the American Statistical Association*, vol. 106, no. 495, 2011.

- [45] C. Shen, M. Song, and Q. Zhao, “Learning high-level concepts by training a deep network on eye fixations,” *Deep Learning and Unsupervised Feature Learning Workshop, in conduction with NIPS, Lake Tahoe, USA*, 2012.
- [46] S. Wu and P. Flach, “A scored auc metric for classifier evaluation and selection,” in *Second Workshop on ROC Analysis in ML, Bonn, Germany*, 2005.
- [47] M. Cerf, E. P. Frady, and C. Koch, “Faces and text attract gaze independent of the task: Experimental data and computer model,” *Journal of vision*, vol. 9, no. 12, p. 10, 2009.
- [48] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, “Multimodal deep learning,” in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 689–696.
- [49] D. Cireşan, U. Meier, J. Masci, and J. Schmidhuber, “Multi-column deep neural network for traffic sign classification,” *Neural Networks*, vol. 32, pp. 333–338, 2012.
- [50] A. Coates and A. Y. Ng, “Learning feature representations with k-means,” in *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 561–580.
- [51] L. Rothacker, S. Vajda, and G. A. Fink, “Bag-of-features representations for offline handwriting recognition applied to arabic script,” in *Frontiers in Handwriting Recognition (ICFHR), 2012 International Conference on*. IEEE, 2012, pp. 149–154.

- [52] T. Wang, D. J. Wu, A. Coates, and A. Y. Ng, “End-to-end text recognition with convolutional neural networks,” in *Pattern Recognition (ICPR), 2012 21st International Conference on*. IEEE, 2012, pp. 3304–3308.
- [53] K. Kavukcuoglu, M. Ranzato, R. Fergus, and Y. LeCun, “Learning invariant features through topographic filter maps,” in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 1605–1612.
- [54] A. Coates, A. Karpathy, and A. Y. Ng, “Emergence of object-selective features in unsupervised feature learning.” in *NIPS*, vol. 25, 2012, pp. 2690–2698.
- [55] Q. V. Le, “Building high-level features using large scale unsupervised learning,” in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 8595–8598.
- [56] Y. Pang, W. Li, Y. Yuan, and J. Pan, “Fully affine invariant surf for image matching,” *Neurocomputing*, vol. 85, pp. 6–10, 2012.
- [57] J. Yang, K. Yu, Y. Gong, and T. Huang, “Linear spatial pyramid matching using sparse coding for image classification,” in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 1794–1801.
- [58] F. Bastien, Y. Bengio, A. Bergeron, N. Boulanger-Lewandowski, T. Breuel, Y. Chherawala, M. Cisse, M. Côté, D. Erhan, J. Eustache *et al.*, “Deep

- self-taught learning for handwritten character recognition,” *arXiv preprint arXiv:1009.3589*, 2010.
- [59] C. Geng and X. Jiang, “Face recognition using sift features,” in *Image Processing (ICIP), 2009 16th IEEE International Conference on*. IEEE, 2009, pp. 3313–3316.
- [60] A. Stolyarenko and N. Dershowitz, “Ocr for arabic using sift descriptors with online failure prediction.”
- [61] H. Bay, T. Tuytelaars, and L. Van Gool, “Surf: Speeded up robust features,” in *Computer Vision–ECCV 2006*. Springer, 2006, pp. 404–417.
- [62] Y. Ke and R. Sukthankar, “Pca-sift: A more distinctive representation for local image descriptors,” in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 2. IEEE, 2004, pp. II–506.
- [63] C. Grana, D. Borghesani, M. Manfredi, and R. Cucchiara, “A fast approach for integrating orb descriptors in the bag of words model,” in *IS&T/SPIE Electronic Imaging*. International Society for Optics and Photonics, 2013, pp. 866 709–866 709.
- [64] J. C. van Gemert, C. J. Veenman, A. W. Smeulders, and J.-M. Geusebroek, “Visual word ambiguity,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 32, no. 7, pp. 1271–1283, 2010.

- [65] O. A. Penatti, F. B. Silva, E. Valle, V. Gouet-Brunet, and R. d. S. Torres, "Visual word spatial arrangement for image retrieval and classification," *Pattern Recognition*, vol. 47, no. 2, pp. 705–720, 2014.
- [66] J. Yu, Z. Qin, T. Wan, and X. Zhang, "Feature integration analysis of bag-of-features model for image retrieval," *Neurocomputing*, vol. 120, pp. 355–364, 2013.
- [67] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: a new learning scheme of feedforward neural networks," in *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, vol. 2. IEEE, 2004, pp. 985–990.
- [68] P. Courrieu, "Fast computation of moore-penrose inverse matrices," *arXiv preprint arXiv:0804.4809*, 2008.
- [69] J. Kim, H. S. Shin, K. Shin, M. Lee *et al.*, "Robust algorithm for arrhythmia classification in ecg using extreme learning machine," *Biomed Eng Online*, vol. 8, p. 31, 2009.
- [70] A. Nizar, Z. Dong, and Y. Wang, "Power utility nontechnical loss analysis with extreme learning machine method," *Power Systems, IEEE Transactions on*, vol. 23, no. 3, pp. 946–955, 2008.
- [71] V. Malathi, N. Marimuthu, and S. Baskar, "Intelligent approaches using support vector machine and extreme learning machine for transmission line protection," *Neurocomputing*, vol. 73, no. 10, pp. 2160–2167, 2010.

- [72] Z. Sun, Q. Song, and X. Zhu, “Using coding-based ensemble learning to improve software defect prediction,” *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 42, no. 6, pp. 1806–1817, 2012.
- [73] R. Barandela, R. M. Valdovinos, J. S. Sánchez, and F. J. Ferri, “The imbalanced training sample problem: Under or over sampling?” in *Structural, Syntactic, and Statistical Pattern Recognition*. Springer, 2004, pp. 806–814.
- [74] W. Zong, G.-B. Huang, and Y. Chen, “Weighted extreme learning machine for imbalance learning,” *Neurocomputing*, vol. 101, pp. 229–242, 2013.
- [75] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, “A fast and accurate online sequential learning algorithm for feedforward networks,” *Neural Networks, IEEE Transactions on*, vol. 17, no. 6, pp. 1411–1423, 2006.
- [76] H. He and E. A. Garcia, “Learning from imbalanced data,” *Knowledge and Data Engineering, IEEE Transactions on*, vol. 21, no. 9, pp. 1263–1284, 2009.

Vitae

- **Personal Information**

- Issam Hadj Laradji
- Algerian nationality
- issam.laradji@gmail.com

- **Education**

- Master of Science degree in Computer Science, King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia. Expected graduation date, July, 2014.
- Bachelor of Science degree in Computer Science, King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia, August 2012.

- **Refereed Publications**

- **I. H. Laradji**, L. Ghouti, and E-H. Khiari. “PERCEPTUAL HASHING OF COLOR IMAGES USING HYPERCOMPLEX REPRESENTATIONS.” Published in *ICIP 2013 : IEEE International Conference on Image Processing*.
- **I.H. Laradji**, S. A. Mohammad, and L. Ghouti . “XML Classification using Ensemble Learning on Extracted Features.” To appear in *The 52nd Annual ACM Southeast Conference*.

- **I. H. Laradji**, L. Ghouti, F. Saleh, and M. AlTurki. “Sparse Single-hidden Layer Feedforward Network for Mapping Natural Language Questions to SQL Queries”. To appear in *ICANN 2014 : The 24th International Conference on Artificial Neural Networks*.